RT-WiFi on Software-Defined Radio: Design and Implementation

Zelin Yun†, Peng Wu†, Shengli Zhou†, Aloysius K. Mok‡, Mark Nixon§, Song Han†
†University of Connecticut
†Email: {zelin.yun, peng.wu, shengli.zhou, song.han}@uconn.edu
‡University of Texas at Austin
‡Email: mok@cs.utexas.edu
§Emerson Automation Solutions
§Email: mark.nixon@emerson.com

2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS) | 978-1-6654-9998-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/RTAS54340.2022.00028

***Abstract*—Applying high-speed real-time wireless technologies in industrial applications has the great potential to reduce the deployment and maintenance costs compared to their wired counterparts. Wireless technologies enhance the mobility and reduce the communication jitter and delay for mobile industrial equipment, such as mobile collaborative robots. Unfortunately, most existing wireless solutions employed in industrial fields either cannot support the desired high-speed communications or cannot guarantee deterministic, real-time performance. A more recent wireless technology, RT-WiFi, achieves a good balance between high-speed data rates and deterministic communication performance. It is however developed on commercial-of-the-shelf (COTS) hardware, and takes considerable effort and hardware expertise to maintain and upgrade. To address these problems, this paper introduces the software-defined radio (SDR)-based RT-WiFi solution which we call SRT-WiFi. SRT-WiFi provides full-stack configurability for high-speed real-time wireless communications. We present the overall system architecture of SRT-WiFi and discuss its key functions which achieve better timing performance and solve the queue management and rate adaptation issues compared to COTS hardware-based RT-WiFi. To achieve effective network management with rate adaptation in multi-cluster SRT-WiFi, a novel scheduling problem is formulated and an effective algorithm is proposed to solve the problem. A multi-cluster SRT-WiFi testbed is developed to validate the design, and extensive experiments are performed to evaluate the performance at both device and system levels.**

***Index Terms*—Software-defined radio (SDR), RT-WiFi, full-stack configurability**

I. Introduction

A recent trend in smart factory automation is to employ high-speed real-time wireless technologies to interconnect heterogeneous industrial assets to perform various sensing and control services, and support mobile equipment to conduct designated tasks in a collaborative fashion [1]. Most of these industrial applications have stringent requirements on both high data throughput and deterministic real-time performance (*e.g.*, latency and jitter) [2], [3].

The existing efforts on the design and implementation of real-time wireless solutions can be summarized in four main categories. The first category includes those works focusing on low-speed low-power real-time communication solutions, such as WirelessHART, ISA 100.11a, WISA and 6TiSCH [4]–[7]. Although those solutions can achieve deterministic communication performance and have ultra-low energy footprint, they cannot support high-speed communications, constrained by the underlying IEEE 802.15.4/802.15.4e [8] physical layer (PHY) and data link layer (DLL). The works in the second category [9]–[14] achieve the real-time performance based on IEEE 802.11e standard, including the hybrid coordination function (HCF) controlled channel access (HCCA) which enables the polling method [15]–[17] and the enhanced distributed channel access (EDCA) which enables priorities in the transmissions and uses the highest priority for the real-time transmissions to guarantee their access to the channel. However, when EDCA is applied, the downlinks may compete for the highest priority queue on the access point (AP) side which may cause unnecessary delay and the ensuing timing violations. The polling method in HCCA is not time-efficient when the channel usage is high compared to assigning communication schedules to the devices directly and it is also subject to coexistence issues in the scenarios when multiple APs use the same HCCA access function [1]. The works in the third category study the applications of 5G and Long Term Evolution (LTE) technologies in real-time industrial applications [18]–[20]. However, the deployment of LTE and 5G equipment do not exploit the license-free bands and therefore misses the economic advantage and the flexibility afforded by the extra bandwidth required for the anticipated applications in the industrial automation field such as robotics. For the last category, existing works [21], [22] focus on modifying IEEE 802.11 standards and implementing the systems on COTS hardware. For example, [21] proposes a configurable real-time WiFi system, called RT-WiFi, based on Qualcomm Atheros AR9285. It modifies the driver and implements a network manager for scheduling deterministic real-time communications. For the above works using COTS hardware, a major issue is that COTS hardware is usually not open-source, and many functions are not accessible which makes it difficult to maintain and upgrade such system to support frequently updated OS kernels and wireless protocols.[1]

To address the aforementioned issues in existing work, we present in this paper the design and implementation of a software-defined radio (SDR)-based RT-WiFi solution which we name SRT-WiFi. SDR [24] is a radio communication system where components that have been traditionally implemented in hardware are instead implemented by means of software on a PC or an embedded system. We design SRT-

[1]For a more detailed discussion on the related work, please refer to our technical report [23].

978-1-6654-9998-9/22/$31.00 ©2022 IEEE
DOI 10.1109/RTAS54340.2022.00028

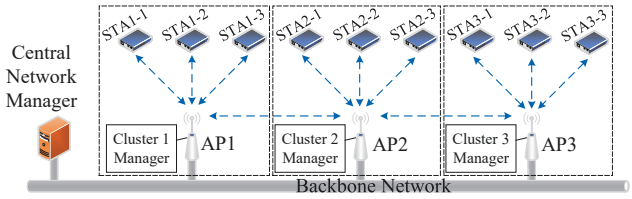

Fig. 1: Overview of the multi-cluster SRT-WiFi network.

WiFi based on an advanced SDR platform (ZC706 development board with Zynq-7000 and AD9364) where the radio functions are programmed on field programmable gate array (FPGA). This advanced SDR system can run in real time since the radio functions are achieved by the logic blocks in FPGA running at the speed as driven by an oscillator. With such a programmable real-time radio system, SRT-WiFi can achieve the key functions required to support high-speed real-time communications, and also provide an open-source platform to support ever-evolving IEEE 802.11 standards.

Fig. 1 gives the overview of a multi-cluster SRT-WiFi network where multiple APs are synchronized and connected to a backbone network. A central network manager (CNM) manages all the network resources and allocates them to the cluster managers (CMs) running on individual APs. In each cluster, high-speed real-time point-to-point wireless communications with rate adaptation are supported to deal with the interfered environments. The clusters operate on multiple channels meaning that one channel has one or multiple clusters operating on it. Compared to COTS hardware-based existing works, SRT-WiFi leverages the programmability of the SDR-based PHY and DLL to provide full-stack configurability.[2] By taking advantage of this full-stack configurability, it is possible to add three major features in SRT-WiFi: i) more precise time synchronization which leads to a smaller slot size for packet transmission and higher sampling rate; ii) efficient queue management which reduces possible downlink latency caused by the limited number of queues in COTS hardware; iii) more accurate signal-to-noise ratio (SNR) measurement, based on which we propose a novel rate adaptation mechanism to dynamically change the data rates based on the SNR measurement of the links to guarantee the desired packet delivery ratio (PDR) of each link; this adaptation outperforms the Minstrel algorithm [25] employed in regular WiFi network. Based on the proposed rate adaptation mechanism, we further formulate and solve the multi-cluster SRT-WiFi network scheduling problem (MSNS-RA) based on the dynamic rates determined at run time. We implement the SRT-WiFi protocol and the multi-cluster network management solution on a real testbed, and validate the effectiveness of the designs through extensive experiments at both device and system levels.

The remainder of this paper is organized as follows. Section II presents the overall system architecture of SRT-WiFi. Section III and Section IV describe the design of the programmable logic (PL) component and the processing system (PS) component of SRT-WiFi, respectively. Section V introduces the multi-cluster network management framework to support rate adaptation in SRT-WiFi to guarantee the timing requirement of real-time tasks even in the presence of severe interference. Section VI evaluates the performance of SRT-WiFi at both device and system levels. We conclude the paper in Section VII and discuss the ongoing and future work.

[2]The current version of SRT-WiFi system supports IEEE 802.11a/g. It can be further extended to support emerging IEEE 802.11 standards, such as 802.11n/ac/ax. See the ongoing and future work in Section VII).

II. System Architecture

SRT-WiFi is based on the Openwifi project [26], [27] which is a SoftMAC IEEE 802.11 design compatible with Linux MAC80211. In this section, we first introduce Openwifi, and then describe the SRT-WiFi architecture in detail (see Fig. 2).

A. Openwifi Architecture

Openwifi has two major components: the Processing System (PS) and the Programmable Logic (PL). PS is an operating system (OS) running the major part of the data link layer (DLL) and all the other higher layers. PL is an FPGA-based embedded system running the real-time part of the DLL and the physical layer (PHY). Both PL and PS are implemented on an integrated System-on-Chip (SoC) which consists of an FPGA (for PL) and an ARM processor (for PS). PL and PS exchange data through the Advanced eXtensible Interface (AXI) [28]. In addition, PL connects to a radio terminal for packet transceiving.

In Openwifi, PL is designed as the wireless adaptor. As shown on the right side of Fig. 2, PL has three main modules: the TX interface (TXI), the XPU (application-specific processing unit) and the RX interface (RXI). The TXI and RXI modules handle packet transmission and reception, respectively. The XPU module runs the state machine of IEEE 802.11 channel access methods. To process general packet transmissions, TXI first holds the packet passed from PS in its queues and waits for the transmission trigger from XPU. The carrier-sense multiple access (CSMA) block in XPU senses the channel and runs the backoff mechanism. Once the channel is available, XPU triggers TXI which in turn fires the packet to the modulation block (OFDM TX).The modulated signal is then passed through radio interface and finally emitted from the antenna by the radio terminal. After sending the packet, the XPU module waits for the acknowledgement (ACK) packet from RXI if ACK is required. If ACK is correctly received or not required, the transmission success is reported to PS. If XPU does not receive the correct ACK after a pre-defined time threshold, it triggers retransmission(s) until reaches the limit of transmission attempts and then reports the failure to PS. For a general packet reception, the signal from radio terminal is demodulated by OFDM RX and pushed in a queue in RXI. At the same time, XPU reads the packet header and applies a packet filter to decide if this packet is destined for PS. If so, RXI then fires the packet to PS. If the packet requires an ACK, XPU generates the ACK packet in TXI and triggers the transmission. Both TX and RX of the radio terminal run

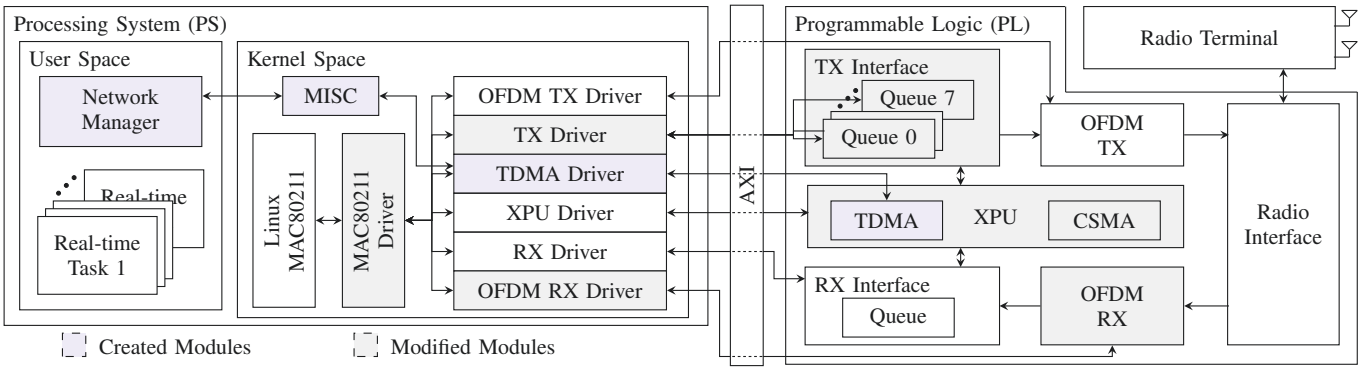

Fig. 2: Overview of the SRT-WiFi system architecture design based on the Openwifi project.

in parallel, and all PL modules have registers to be used for configuring the operation mode and parameters.

The PS component is a Linux OS running on an ARM processor. As a SoftMAC wireless device in Linux, the major part of DLL is integrated in Linux kernel (MAC80211 subsystem [29]). Thus the MAC80211 driver is needed to provide the interface of the wireless adaptor (PL) for the Linux MAC80211 subsystem. The data exchanges between the MAC80211 driver and PL rely on the sub-drivers (see the left side of Fig. 2). All the sub-drivers are designed to provide APIs for register operations to the MAC80211 driver so that it can configure the wireless adaptor (PL). The TX and RX drivers also handle TX and RX data packet transfer between PS and PL, respectively.

B. SRT-WiFi Architecture

The key design goal of SRT-WiFi is to support precise time synchronization and multi-cluster real-time communications with effective rate adaptation at run time. For this purpose, we present below the SRT-WiFi architecture, by modifying PL and PS in Openwifi to add the required functions.

SRT-WiFi PL: The PL component of SRT-WiFi is designed to i) achieve the real-time transmissions with high synchronization time precision, ii) provide more efficient queue management and iii) measure the reception SNR of the links more precisely in order to provide reference for rate adaptation.

To achieve real-time transmissions, we design a TDMA block in XPU to supplement the CSMA block. The TDMA block triggers the PHY and DLL activities with high time precision. It runs either according to the local timer or synchronizes with another device in the SRT-WiFi network. According to our measurements (to be elaborated later), the synchronization time error and standard deviation in the multi-cluster SRT-WiFi are as low as 0.03 µs and 0.07 µs, respectively. Different from the CSMA block which triggers the transmissions following the DCF mechanism, the TDMA block triggers the transmissions according to a schedule constructed by the network managers in PS. The schedule is stored in the TDMA block and updated at run time through a TDMA driver that is added in PS (see Fig. 2). The TDMA and CSMA modes in SRT-WiFi can be switched during the run time seamlessly.

For an AP working in the TDMA mode, it needs to handle the links to all the connected stations. The transmissions on those links have to follow the order of a schedule. With limited number of queues, COTS hardware [21] must manage the issue that the queued packets may block the transmissions of upcoming packets that may cause unmanageable congestions. SRT-WiFi provides an effective queue management which supports more customized queues and significantly increases the number of supported links for growing network scale.

By leveraging the capability of SRT-WiFi to have direct access to the received signals, we are able to design novel methods to measure the SNR precisely and implement it in the OFDM RX module. The SNR information provides a reference for the rate adaptation mechanism to adapt the TX data rates and adjust the communication schedules at run time.

SRT-WiFi PS Kernel: As shown in Fig. 2, we add the TDMA driver and modify the MAC80211 driver, TX and OFDM RX drivers to provide an interface for exchanging the schedule, queue and SNR information between PS and PL in the TDMA mode. The TDMA driver is registered in the kernel as a miscellaneous character driver (MISC). It provides APIs for the network managers in the user space. The network manager configures the schedule and queue information in PL through the TDMA driver and updates the data rates in MAC80211 driver as well. The TX driver is modified to support queue management and the OFDM RX driver is enhanced to support reading the SNR values measured in PL.

SRT-WiFi Network Management: We call the network managers running on individual APs cluster managers (CM) and the ones running on the stations device managers (DM). These network managers are designed for two purposes, i) to exchange information at the application layer among all the devices including the schedule, data rates and SNR of links, ii) to manage the TDMA DLL on each device such as configuring the schedule for the TDMA block and reading the SNR measurement from the PL. All network managers run in the user space so they are easy to maintain and upgrade.

III. SRT-WiFi Programmable Logic (PL) Design

We first present the PL design of SRT-WiFi and focus on the new functions in the XPU, TXI and OFDM RX modules.

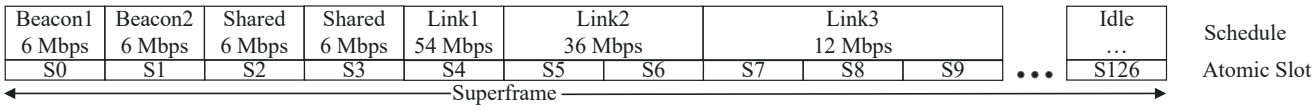

Fig. 3: The timing diagram of an example superframe in multi-cluster SRT-WiFi with 127 atomic slots.

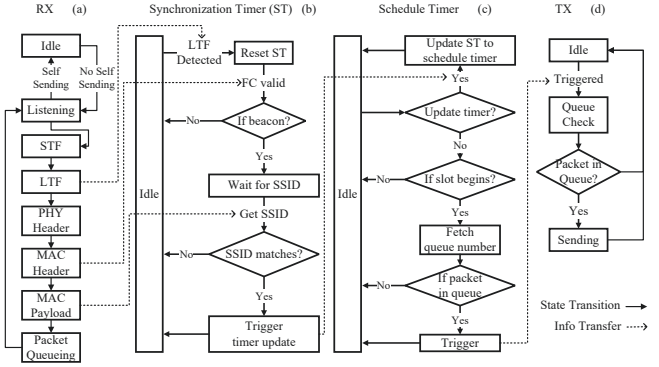

Fig. 4: State machines of the schedule timer and the synchronization timer (ST) in the TDMA block of SRT-WiFi.

A. TDMA Block Design

In the TDMA mode of SRT-WiFi, all transmissions follow a schedule to coordinate the communications among devices and avoid collision. The schedule describes the transmitting times and orders of the links in a time period called superframe which is a sequence of consecutive time slots. Each time slot specifies the radio activities (TX, RX or Idle) and the associated sender/receiver. At run time, the superframe is repeated *ad infinitum* to generate schedule. To support rate adaptation, the length of the time slot varies along with the rate, since with the same packet length, a lower rate requires longer time to transmit. The time slots use atomic slots (ASs) as the basic time unit (to be elaborated later). In SRT-WiFi, the lengths of superframe, time slot and AS are all configurable.

Fig. 3 shows the timing diagram of an example superframe in an SRT-WiFi network. It has 127 ASs where Slot0 and Slot1 are used by AP1 and AP2 to send beacons, respectively. Slot2 and Slot3 are shared slots for any link and usually used for the association process. The other ASs are either assigned to links for dedicated communications or left idle. The links using the same MTU (Maximum Transmission Unit) but different data rates require different slot lengths in terms of the number of ASs. For example, Link1, Link2 and Link3 use 1, 2 and 3 ASs for their transmissions, respectively.[3]

To enable real-time communications in SRT-WiFi and implement the schedule in PL, a TDMA block is added in the XPU module (see Fig. 2). In the TDMA block, a register page is implemented. Some registers are used to configure TDMA parameters, such as the superframe length; the other registers are assigned to keep the schedule information. Based on the schedule information, the TDMA block employs a set of timers called schedule timers to trigger the transmissions. At the beginning of a time slot, the TDMA block fetches the link information associated with that slot and triggers the transmission. The corresponding queue in the TXI module sends a frame if it is not empty. The frame is then modulated and sent through the radio terminal.

[3]See Section VI for the detailed data rates and required slot lengths.

B. TDMA Time Synchronization Design

Another key function of SRT-WiFi is to achieve precise time synchronization among the devices in the network. In our design, we have multiple clusters in the same SRT-WiFi network. Each cluster consists of an AP and multiple stations, and the clusters may share the same channel. For those clusters operating on the same channel, the devices need to be well synchronized to avoid potential collision. The synchronization mechanism of existing work using COTS hardware [30] is to connect and synchronize the APs through an Ethernet backbone network using the IEEE 1588 protocol [31]. The stations are then further synchronized with the APs using the beacon packets. For the synchronization among the APs, the OS of each AP first updates its system timer with IEEE 1588 and then updates the timer in its wireless adaptor, which is used to send packets at run time. For the synchronization on the station side, they listen to the beacon packets and update the system timer. Since the time synchronization on both APs and stations are done by non-real-time OS, it may cause an average time drift between the devices as high as 20 μs [21].

To address this problem, in SRT-WiFi, we propose a new synchronization method based on SDR which is performed at the physical layer (PHY). It is worth noting that this method is only suitable for the devices operating on the same channel. For two APs operating on different channels to synchronize, IEEE 1588 will still be employed. For the APs operating on the same channel, we first designate a master AP (MAP) and let the other APs be the slave APs (SAPs). We assume that all SAPs can hear from the MAP, which provides the reference clock. The SAPs synchronize with the MAP, and all the stations synchronize to their corresponding APs. The key design goal of SRT-WiFi synchronization is to avoid using the timer in non-real-time OS but leverage the timer in hard real-time PL. For this aim, timers are added in the TDMA block with nanosecond precision for synchronization. We call them TDMA timers, and they are set and run in hard real-time. TDMA timers on MAP are set by its OS to unify the time on MAP. TDMA timers on SAPs and stations synchronize with the TDMA timers on MAP using PHY beacon signal and their OS time are synchronized accordingly.

We now introduce the synchronization procedures. In SRT-WiFi, PHY demodulation is achieved in the OFDM RX module in PL. The demodulated symbols are passed to RXI and XPU. In the TDMA block, a synchronization function is added to utilize the baseband signal demodulation to synchronize

TABLE I: PDR with varied payload sizes and slot lengths

Payload (bytes)	50	100	150	200	300	400	500
Slot Length (μs)	110	118	126	130	146	162	174
Sampling Rate (Hz)	9090	8474	7936	7692	6849	6172	5747
PDR (%)	99.7	99.6	99.6	99.6	99.3	99.7	99.4

with a specific AP. More specifically, two TDMA timers are added, one is called schedule timer and the other is called synchronization timer (ST). ST is used to track the arrival time of a beacon packet and the schedule timer is to run the schedule. When a new packet arrives and the long training field (LTF) of the PHY signal is detected in the OFDM RX module, ST is reset. Next, the synchronization function waits for the DLL packet header from the OFDM RX module. It checks whether the packet is a beacon packet. If so, it continues to wait for the service set ID (SSID) in the packet payload. Once SSID is read, the synchronization function compares it with the target SSID. If they match, ST is updated to the schedule timer; otherwise, the synchronization function waits for the next packet and the schedule timer runs as usual with no update. This timer update procedure is summarized in Fig. 4 where Fig. 4 (a) shows how a packet is received and passes the information to ST and Fig. 4 (b) shows how ST synchronizes accordingly and triggers the update of the schedule timer. Fig. 4 (c) shows how the schedule timer changes states to update the time or trigger real-time transmissions as shown in Fig. 4 (d). It is worth noting that this synchronization method also works with higher bit rates in IEEE 802.11n/ac/ax standards. With this method, our experiments show that the synchronization time drift of the SRT-WiFi devices can be maintained within 0.2 μs which is much better than the 20 μs [21] time draft observed on the COTS hardware. This more precise time synchronization can help reduce the guard time which is to avoid collisions between slots due to the synchronization error and support smaller time slot length which further improves the sampling rates. Table I presents the packet delivery ratio (PDR) test results with varied application layer payload sizes, the corresponding slot lengths and achievable sampling rates. The guard time used in the experiments is set at 10 μs. From the results, we can observe that with a payload size of 50 bytes, the slot length can be set at 110 μs and the sampling rate can be as high as 9 kHz. The detailed experimental results can be found in Section VI.

C. Queue Management

In SRT-WiFi, the packets from PS are first pushed in queues before transmission. For COTS hardware-based solutions, the queue implementation is not configurable. For example, AR9285 used in the RT-WiFi implementation [21] uses only 8 queues. To support real-time transmissions in SRT-WiFi, queues are assigned to individual links to guarantee the desired timing performance. However, when the number of stations increases beyond the number of queues in the AP, the packets belonging to different links may share a queue, leading to unexpected timing violations. For example, as shown in Fig. 5 (a), an AP has 10 associated stations while it only has 8 queues

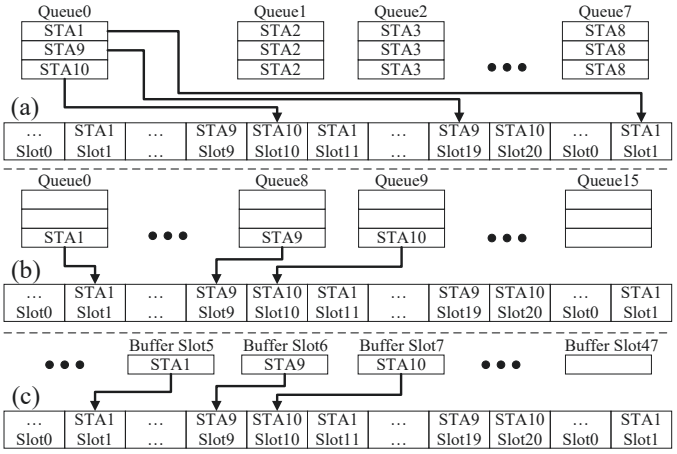

Fig. 5: Queue management issues in RT networks with shared queues.

TABLE II: Max. and avg. delay (slot number) of packets in assigned and dynamic queue management methods with 16 links.

Number of Queues	8	10	12	14	16
Assigned Maximum Delay (slot)	2816	2358	1707	1125	82
Assigned Average Delay (slot)	336	236	159	87	16
Dynamic Maximum Delay (slot)	591	162	106	104	103
Dynamic Average Delay (slot)	271	42	16	16	16

so STA9 and STA10 have to share a queue with other links. When two or more packets belonging to different links are sharing the same queue, they have to wait until the packet at the queue head being sent, although their assigned time slots in the superframe may come first. This issue happens mainly on the AP side when handling transmissions for multiple stations.

In SRT-WiFi, we assign the queues to different links and the packets belonging to different links are pushed to the corresponding queues as shown in Fig. 5 (b). The schedule in the TDMA block stores the information on which queue to be triggered for every slot. This is a feature of SDR-based system since the number of supported queues can be extended as long as the FPGA has enough resources.

This SDR-based queue implementation can eliminate the queuing delay when the number of supported stations is smaller than the number of implemented queues on the AP. However, as the number of stations increases, the number of queues cannot be increased infinitely. To address this issue, we propose a dynamic buffer design in SRT-WiFi as shown in Fig. 5 (c) where we use a buffer to replace the previous queues and the buffer is composed of buffer slots and each buffer slot only stores one packet at most. When a packet is passed from the driver, TXI selects an unused buffer slot and pushes the packet into it. At the beginning of each time slot, the TDMA module checks the link information for that slot. It goes over the buffer to check whether there is a packet belonging to that link. If so, it transmits that packet. Since a buffer slot only stores one packet, with the same FPGA resources, more buffer slots can be implemented than queues.

Table II presents the performance comparison between the assigned and dynamic queue management methods with 16 links and the number of queues. We assume that each link

generates a packet periodically and a packet only requires one atomic slot to transmit. If the corresponding queue is available, the packet is pushed into queue. All transmissions follow a randomly generated schedule where the throughput of each link is guaranteed and the length of superframe is fixed. The time between the packet being generated and transmitted is recorded as the packet delay, and packets are not dropped due to the delay. From the results, we can observe that in the assigned queue management method (queues are assigned to links), a few gap between the number of queues and links may cause significantly large max./avg. delay. On the other hand, with the same number of queues, the dynamic queue management method is able to handle more links and keep both max and avg. packet delay small. It however cannot eliminate the delay since all the queues are shared.

D. SNR Measurement

To support the rate adaptation function in SRT-WiFi, we propose two practical methods to achieve precise SNR measurement in PL. Both methods utilize the short training field (STF) in the preamble of 802.11 PHY signal. The first method computes the cross-correlation [32] of the STF. It is known that STF consists of 10 same short symbols corresponding to 160 samples with 20 MHz sampling rate. So the samples in STF repeat every 16 samples [33]. After the detection of STF, it is buffered. We use the chips from the 33$^{\text{rd}}$ to the last one (in total 128 chips) and divide them into two groups each of which has 64 chips. We compute the cross-correlation of the two groups of chips as the ρ, and the SNR value (dB) can be computed as follows:

$$\mathrm{SNR} = 10\log_{10}\left(\frac{\rho}{1-\rho}\right) \tag{1}$$

where we assume that $\rho < 1$. The reason that we use two groups of 64 chips is to exclude the chips at the beginning due to the problems caused by the transient effects of initiating a transmission in the hardware of the sender.

For the second method, after the STF detection, the STF and a piece of background noise before the STF are buffered. The STF signal is added by the background noise. We measure the power of the background noise before the STF and the power of the STF signal which is noise power plus the signal power. Then the SNR (dB) can be computed as:

$$\mathrm{SNR} = 10\log_{10}\left(\frac{P_{\mathrm{STF}} - P_{\mathrm{noise}}}{P_{\mathrm{noise}}}\right) \tag{2}$$

where P_{STF} is the signal power of the STF part and P_{noise} is the power of the background noise signal before the STF. We assume that P_{STF} is larger than P_{noise}.

Both SNR measurement methods are implemented in the OFDM RX module in SRT-WiFi, and their performance is discussed in Section VI. An SNR value is computed every time when a packet arrives. The computed SNR value is buffered together with the source address if applicable (not all the packets have the source address, if not the SNR value is discarded) so that we know which link the SNR value belongs to since there could be multiple packets being processed during that time. This information is obtained by the device manager on each device through the drivers. The device manager then sends the SNR information to the central network manager to determine the data rate of each link and the corresponding schedule in the network. The performance of both SNR measurement methods are discussed in Section VI.

IV. SRT-WiFi Processing System (PS) Design

We now introduce the SRT-WiFi PS design including the drivers in the kernel and the network managers in user space.

A. SRT-WiFi Drivers

The SRT-WiFi drivers are the interface between PL and kernel with two purposes: i) configure parameters in the PL modules to support different operation modes and functions and ii) handle the packet exchange between PL and kernel.

We first present the PL configuration and the structure of the drivers. As shown in Fig. 2, each module in PL is connected to a corresponding driver in the kernel to operate its registers. We call these drivers sub-drivers. They encapsulate the register operation functions into APIs to be called in the MAC80211 driver. For the TDMA block in XPU, it also has registers which are divided into two parts, i) to keep information of the network such as atomic slot length, superframe length and the SSID of the AP that it synchronizes with, ii) to keep the TDMA schedule including the link and queue information of each time slot. To configure these registers, we add a TDMA driver in the kernel. Since the functions of the TDMA mode are not compatible with the MAC80211 subsystem, it is difficult to configure the TDMA block through MAC80211. Instead, we make the TDMA driver a miscellaneous character driver (MISC). It provides reading and writing functions for the user space. In the user space, the network manager calls the APIs of the TDMA driver to configure the TDMA block so that it can modify the schedule, set the parameters and switch the working mode when necessary.

When the MAC80211 sub-system sends a packet, the packet is passed to the MAC80211 driver and handled by the TX operation function (TXO). In SRT-WiFi, we specify the TX rate and queue for each link. The MAC80211 driver has access to the TDMA driver so TXO fetches the assigned queue and TX rate for the current packet using the destination address of the packet as the key. The queue assignment and rate selection are decided by CNM and each device stores that information in a table in the TDMA driver. The packet is finally passed to PL for transmission with the rate and queue information.

B. Network Manager

In SRT-WiFi, we have three types of network managers forming a network management hierarchy, including the central network manager (CNM), cluster managers (CM) running on the APs and device managers (DM) running on the stations. CNM determines and updates the schedules and assigns resources to the DMs through their associated CMs.

To maintain proper operations of the SRT-WiFi network, CNM acquires the global knowledge from all devices including the APs and stations. When a SRT-WiFi network starts,

CNM runs first and waits for CMs to connect and determines the schedules to be assigned to the links. CMs then start its cluster and the slave APs synchronize with the master AP in the same channel and wait for the stations to connect. For the convenience of synchronization and the joining process, all beacon slots and shared slots are fixed during the system operation, and this information is shared with all APs and stations. When a station is powered on, it scans the channels, synchronizes with the designated AP and joins the network. After that, the DM on that station connects to the CM on the AP to obtain and update the schedule. Before receiving the schedule, the station can only use the shared slots to complete the joining process in the CSMA mode.

A unique feature of SRT-WiFi network management is to enable dynamic slot length in the schedule to support run-time rate adaptation. For an individual link, the MTU is fixed while the data rate changes along with the interference level. With a lower data rate, a packet of the same length requires longer time to transmit which may exceed the boundary of a time slot and cause collision. In this work, we apply dynamic slot length in the schedule to solve this issue. In the schedule, we define an atomic slot (AS) to be a slot that has the minimum length to support transmitting a packet with a size of MTU at the highest rate. For a packet to transmit at a lower rate, it can use multiple consecutive ASs in a non-preemptive fashion. Thus by choosing different rates in the run time, the packet transmission can take different number of ASs. With this dynamic slot size assignment mechanism, we will formulate and solve the multi-cluster SRT-WiFi network scheduling problem with rate adaptation (MSNS-RA) below.

V. Network Management

We now formulate the multi-cluster SRT-WiFi network scheduling with rate adaptation (MSNS-RA) problem, prove its NP-hardness, and present the design detail of a heuristic scheduler.

A. System Model

Consider a set $C = \{C_1, C_2, ..., C_m\}$ of clusters in a multi-cluster SRT-WiFi network. Each cluster consists of one SRT-WiFi AP and multiple SRT-WiFi stations forming a star network topology. As the SRT-WiFi network is a time-slotted system, we define an *atomic slot (AS)* as the minimal uninterruptible time unit in the system. For each packet to be transmitted in the SRT-WiFi network, it takes one or multiple *transmission units*. A transmission unit is configured to be one or multiple atomic slots based on the selected data rates.

Let $\Pi_i = \{\tau_{i,1}, \tau_{i,2}, ..., \tau_{i,n}\}$ be a set of tasks that transmit the packets periodically in cluster C_i. Each task $\tau_{i,j}$ is characterized by $\tau_{i,j} = (B_{i,j}, U_{i,j}, D_{i,j}, T_{i,j})$, where $B_{i,j} \in \mathbb{N}$ represents the size of the transmission unit for $\tau_{i,j}$ (in number of atomic slots), $U_{i,j}$ is the number of transmission units required by $\tau_{i,j}$. The deadline and period of $\tau_{i,j}$ are denoted as $D_{i,j}$ and $T_{i,j}$, respectively.

We assume that each task $\tau_{i,j}$ is released in a periodic fashion with a set of instances $\{\mathcal{I}_{i,j,k}\}_{k=1}^{\infty}$. For a transmission unit $l \in [1, U_{i,j}]$ of an instance $\mathcal{I}_{i,j,k}$, let $s_{i,j,k,l}$ and $f_{i,j,k,l}$ represent its *start time* and *finish time*, respectively. Accordingly, let $r_{i,j,k,l}$ and $d_{i,j,k,l}$ be the release time and deadline of the l^{th} transmission unit of $\mathcal{I}_{i,j,k}$. The release time $r_{i,j,k,1}$ of the first transmission unit is the release time of $\mathcal{I}_{i,j,k}$ and the deadline $d_{i,j,k,U_{i,j}}$ of the last transmission unit is the deadline of $\mathcal{I}_{i,j,k}$. In addition, it holds that $r_{i,j,k,p} = f_{i,j,k,p-1}$ with $p \in [2, U_{i,j}]$ and $d_{i,j,k,q} = d_{i,j,k,q+1} - B_{i,j}$ with $q \in [1, U_{i,j} - 1]$.

We assume that the sizes of atomic slots for all the clusters are the same, and the number of available channels in the SRT-WiFi network is H. We assign each cluster C_i to a channel $h_i \in [1, H]$, and introduce the *conflict condition* as follows.

Definition 1. *For any two instances $\mathcal{I}_{i,j,k}$ and $\mathcal{I}_{i',j',k'}$ with $i = i'$ or $h_i = h_{i'}$ hold, we say that they conflict with each other if the following condition satisfies:*

$$[s_{i,j,k,l}, f_{i,j,k,l}] \cup [s_{i',j',k',l'}, f_{i',j',k',l'}] \neq \emptyset \tag{3}$$

where $l \in [1, U_{i,j}]$, $l' \in [1, U_{i',j'}]$, and it cannot hold that $i = i'$, $j = j'$, $k = k'$ and $l = l'$ at the same time.

Based on the conflict condition above, we define the feasible condition of scheduling an instance in multi-cluster SRT-WiFi.

Definition 2. *For any instance $\mathcal{I}_{i,j,k}$, it is feasibly scheduled in a multi-cluster SRT-WiFi network if it does not conflict with any other instance and the following condition holds:*

$$[s_{i,j,k,l}, f_{i,j,k,l}] \subseteq [r_{i,j,k,l}, d_{i,j,k,l}] \tag{4}$$

where $f_{i,j,k,l} = s_{i,j,k,l} + B_{i,j}$ and $l \in [1, U_{i,j}]$.

B. Problem Formulation

The MSNS-RA problem considers assigning channels to individual clusters and then schedule the transmissions of the packets to eliminate the schedule conflict.

Definition 3. *Consider a set of clusters $\{C_i\}_{i=1}^{m}$ each executing a set of tasks $\{\tau_{i,j}\}_{j=1}^{n}$, the MSNS-RA problem is to assign a channel $h_i \in [1, H]$ to each cluster C_i and to find a feasible schedule for all the tasks assigned with rate adaptation on the same channel so that any instance of a task can be feasibly scheduled based on Condition (3) and (4).*

Theorem 1. *MSNS-RA is NP-hard in the strong sense.*

Proof. Our NP-hard proof uses the 3-Partition problem which is known to be NP-hard [34]. An instance of 3-Partition consists of a collection $A = (x_1, x_2, ..., x_{3n})$ of positive integers such that $\sum x_i = nM$, $\frac{M}{4} < x_i < \frac{M}{2}$ for each $1 \leq i \leq 3n$, there exists a partition of A into A_1, A_2,...,A_n such that $\sum_{x_i \in A_k} x_i = B$ for each $1 \leq k \leq n$ [34].

From any instance of the 3-partition problem, we may construct an equivalent instance of MSNS-RA. Assuming that the total number of available channel in MSNS-RA is 1. Let Π be the set of tasks running in all the clusters. For each integer x_i, we map a corresponding task in Pi with its size of transmission unit as x_i, its number of transmission units as 1, its period and deadline as M. Also, we construct an extra task with its size of transmission unit as M, its number of

transmission units as 1, its period and deadline as nM. The above reduction is a polynomial reduction.

Since the extra task has taken the intervals in $[k \times M, (k+1) \times M]$ where $k \in [0, 2n-2]$ is an even number, if there exists a partition of A into A_1, A_2,...,A_n in the 3-partition problem, we can construct the corresponding solution to schedule each 3 tasks corresponding to the integers in A_i in an unused interval $[(k+1) \times M, (k+2) \times M]$. Also, if we could find a feasible schedule for the MSNS-RA problem, we must schedule every 3 tasks in an unused interval $[(k+1) \times M, (k+2) \times M]$ since $\frac{M}{4} < x_i < \frac{M}{2}$. This shows that there is a feasible schedule if and only if there is a 3-Partition, which proves that the MSNS-RA is NP-hard in the strong sense. □

C. Heuristic Scheduler Design

To address the MSNS-RA problem in the general case, we propose an effective heuristic scheduler to perform the channel and task assignment. The proposed scheduler design contains a cluster scheduler and a set of task schedulers. The cluster scheduler assigns the channels for individual clusters by balancing the network utilization of channels. Once the channel assignment is completed, the task schedulers are employed to schedule the tasks in each cluster. The details of the two schedulers are presented below.

1) Cluster scheduler: Given m clusters $\{C_i\}_{i=1}^{m}$, the cluster scheduler first computes and sorts the clusters according to their network utilization in descending order. Specifically, for each cluster C_i with the corresponding task set $\{\tau_{i,j}\}_{j=1}^{n}$, the network utilization of the cluster is computed as the sum of its task utilization. To reduce the search space, we introduce a heuristic to assign the clusters to each channel. We define the network utilization of a channel as the sum of the network utilization of all the tasks assigned to this channel. For a cluster C_i, with H available channels, we always select the channel with the least network utilization and assign it to C_i. For example, we assign C_1 to channel 1, C_2 to channel 2, ... , C_H to channel H. For cluster C_{H+1}, we assign it to channel H as the utilization of channel H is the lowest. For cluster C_{H+2}, we compare channel $H-1$ and channel H and select the one with the lowest network utilization.

2) Task scheduler: After assigning the channels for individual clusters, the task scheduler aims to find a feasible schedule for all the tasks assigned to the same channel. Given n tasks $\{\tau_i\}_{i=1}^{n}$ assigned to a channel $h \in [1, H]$ which may be from different clusters with a hyper-period $\mathcal{H}$, we utilize the release times and deadlines of the transmission units of all the instances of every task from the task set to build the interval set $\mathcal{T}$. For any interval $I \in \mathcal{T}$ with $I = [s, e]$, s is a release time of a transmission unit and e is the deadline of that transmission unit. Let $\mathcal{D}_I$ be the *demand* of the interval I, which is defined as the sum of $B_{i,j}$ of any transmission unit of $\mathcal{I}_{i,j,k}$ with its release time and deadline included in I.

Following the EDF (Earliest Deadline First) scheduling policy, we schedule the transmission units based on their deadlines. However, the sizes of transmission units from different tasks might be different and a transmission unit cannot be interrupted during execution. In this non-preemptive case, EDF is known to be non-optimal. To improve the schedulability, we consider the technique of inserting idle time. The key idea is that for each instance popped from the ready queue we utilize the future release patterns of tasks to decide whether or not to insert the idle time to delay its execution. This prevents a non-preemptive transmission unit from being scheduled in an interval such that its demand plus part of this transmission unit becomes larger than the length of the interval, thus jeopardizing the schedulability. To overcome this problem, we employ the following rule to insert the idle time in the constructed schedule. For any transmission unit of a task instance $\mathcal{I}_{i,j,k}$ to be scheduled at time t. If there exists an interval $I = [s, e]$ satisfying the following two conditions:

- Condition 1: $[s, e] \subset [t, d_{i,j,k,l}]$
- Condition 2: $t + B_{i,j} > e - \mathcal{D}_I$.

then the release time of the transmission unit is set to s.

Since there may exist multiple intervals that satisfy the above conditions, we change the release time of the transmission unit to be the latest one. In addition, deferring the release time to a later time can change the interval set $\mathcal{T}$. We therefore update the interval set once a release time is updated.

With the above rule to insert idle time in the EDF schedule, we describe the operation of the task scheduler in Alg. 1. The task scheduler first computes the hyper-period $\mathcal{H}$ of the task set assigned to the channel and initializes the interval set $\mathcal{T}$ and the time t based on the timing parameters of the task instances. It then utilizes a ready queue Q to schedule the tasks based on EDF. Specifically, for each instance $I_{i,j,k}$ popped from the ready queue, we employ the rule of inserting idle time to decide if its release time will be deferred (Line 6-11). In addition, we check if the current transmission unit can be scheduled (Line 12-13). If the release time of the current transmission unit is not modified, i.e., $r_{i,j,k,l} = t$, we schedule it in time $[t, t + B_{i,j}]$. Otherwise, we push it back to the ready queue and update the interval set $\mathcal{T}$ (Line 19-20). Let $N = \sum_{i=1}^{n} \mathcal{H}/T_i$ be the total amount of instances of all the tasks where T_i is the period. As computing the interval set takes $O(N^2)$ time, the total time complexity of the task scheduler is $O(N^3)$.

VI. Performance Evaluation

In this section we report our performance evaluation on the SRT-WiFi design, at both component and system levels.[4] Fig. 6 presents the devices used in our SRT-WiFi testbed. We have two hardware platforms. ZC706 consists of Z7045 SoC and AD9364 radio chip. It is used as the hardware for both AP and stations. ADRV9364-Z7020 consists of Z7020 SoC and AD9364 radio chip. It is only used for some stations due to its limited FPGA resources. USRP2900 is a traditional SDR device. It is only used for testing purpose in the SRT-WiFi testbed such as signal analysis and interference generation.

[4]More experimental results can be found in our technical report [23].

Algorithm 1: Task Scheduler

Input : A task set $\Pi = \{\tau_i\}_{i=1}^{n}$
Output: A schedule $\mathcal{S}$ or reports failure

Compute the hyper-period $\mathcal{H}$ of the tasks
Initialize a ready queue $Q = \emptyset$, the interval set $\mathcal{T}$ and time t
while *An instance* $\mathcal{I}_{x,y,z}$ *is released or* $Q \neq \emptyset$ **do**
  $Q = Q \cup \{\mathcal{I}_{x,y,z}\}$
  Get the earliest instance $\mathcal{I}_{i,j,k}$ in Q at current time t with its l^{th} transmission unit to be scheduled
  for $I \in \mathcal{T}$ *with* $I = [s, e]$ **do**
    Compute the demand $\mathcal{D}_I$ of the interval I
    if $I \subset [t, d_{i,j,k,l}] \wedge t + B_{i,j} + \mathcal{D}_I > e$ **then**
      $r_{i,j,k,l} = \max(r_{i,j,k,l}, s)$
    end
  end
  if $r_{i,j,k,l} + B_{i,j} > d_{i,j,k,l}$ **then**
    return None `// reports failure`
  end
  if $r_{i,j,k,l} = t$ **then**
    $[s_{i,j,k,l}, f_{i,j,k,l}] = [t, t + B_{i,j}]$
    $t = f_{i,j,k,l}$
  else
    insert $\mathcal{I}_{i,j,k}$ to Π
    update the interval set $\mathcal{T}$
  end
end
return $\mathcal{S}$

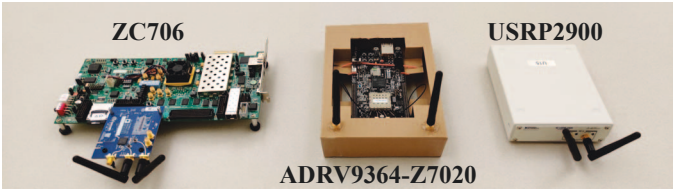

Fig. 6: SDR hardware used in the SRT-WiFi testbeds.

A. Synchronization

We first evaluate the effectiveness of the proposed time synchronization mechanism in SRT-WiFi. To support multi-cluster SRT-WiFi, we let the slave APs (SAPs) synchronize with the master AP (MAP) and the stations synchronize with either the MAP or SAP. In the experiments, we first test the beacon interval of the MAP by configuring it to send the beacon packets periodically. We use USRP2900 to capture the beacon signal and use the COTS hardware (AR9285) working in the monitoring mode to sniff the beacon packets.

In the tests, we set the slot length at 500 µs and the superframe length at 127 slots so the expected beacon interval is 63.5 ms (one beacon per superframe). Fig. 7 (a) and (b) show the time drift of the beacon interval measured by USRP2900 and AR9285, respectively. The time drift is measured as the error between the inter-arrival time of two consecutive beacons and the expected superframe length (63.5 ms). The SDR result

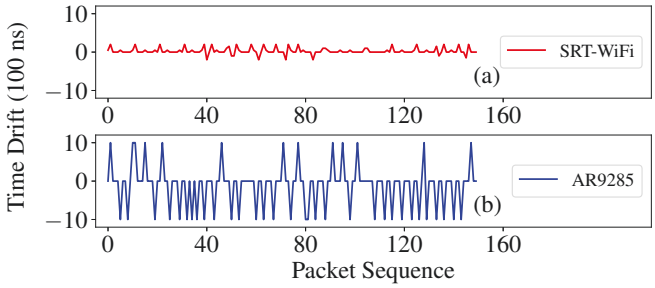

Fig. 7: Time drifts in beacon interval by USRP2900 and AR9285.

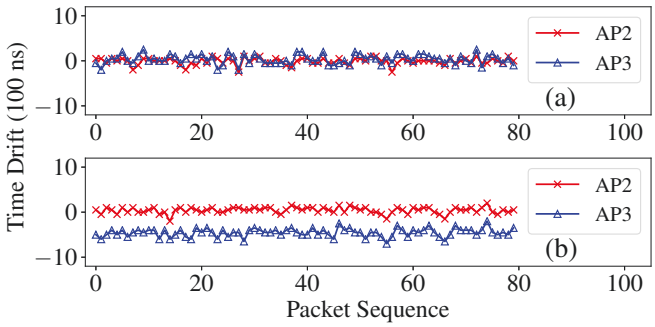

Fig. 8: Synchronization performance.

is measured directly from the captured base band signal and the average error, maximal error and the standard deviation are 0.03 µs, 0.2 µs and 0.07 µs, respectively. On the other hand, the result measured from AR9285 has the average error, maximal error and the standard deviation as 0.13 µs, 1 µs and 0.54 µs, respectively. From the comparison, we observed that SRT-WiFi has much more accurate timer than the COTS hardware. Thus the implementation using COTS hardware needs a larger guard time in the slot design to avoid potential collision.

Next, we test the synchronization performance of multi-cluster SRT-WiFi networks. In SRT-WiFi, the MAP provides the reference clock. For the SAPs and stations to connect to the MAP, they listen to the beacons from the MAP. We call it level-1 synchronization. For the stations connecting to the SAPs, we call it level-2 synchronization. We use three APs in the experiments. To test level-1 synchronization performance, we set AP1 as the MAP sending beacons in slot 0, and AP2 and AP3 as SAPs to synchronize with AP1. We use USRP2900 to measure the beacon sending time of the three APs. Fig. 8 (a) shows the sending time errors of AP2 and AP3. AP2 uses slot 115 to send beacons. The average sending time error is 0.01 µs with a standard deviation of 0.08 µs. AP3 uses slot 117, and its average error is 0.03 µs with a standard deviation of 0.1 µs. These results show that the accuracy of level-1 synchronization can be well maintained within 1 µs. Next, we let AP2 synchronize with AP1 and AP3 synchronize with AP2 to test level-2 synchronization performance. We configure AP1 to send beacons in slot 0 and AP2 to send beacons in slot 2. The measured results in Fig. 8 (b) show that the average error is 0.04 µs. We further let AP3 synchronize with AP2 and send beacons in slot 119 and the average error is 0.5

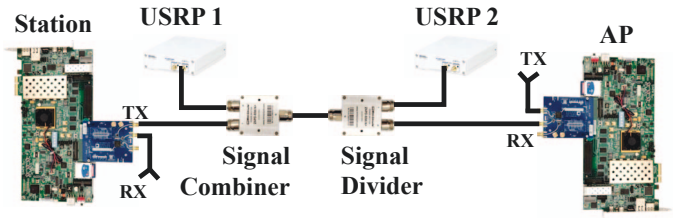

Fig. 9: Setup of the SRT-WiFi testbed for SNR measurement.

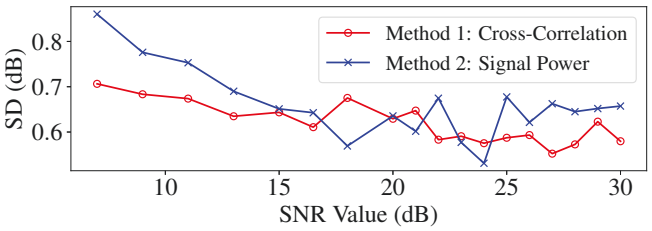

Fig. 10: Standard deviation of the two SNR measurement methods.

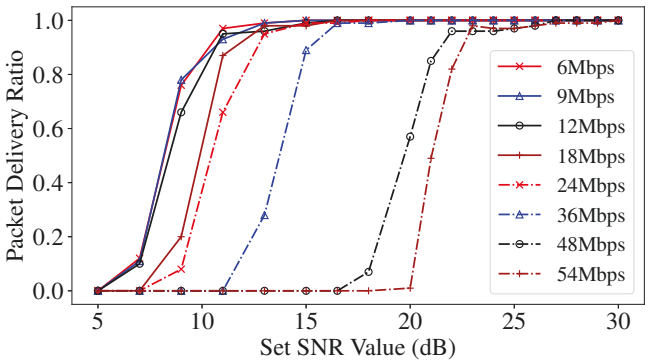

Fig. 11: PDR under different rates against SNR.

TABLE III: TX rates and slot lengths under different SNR values.

SNR threshold (dB)	25	22	19	17	15	13	10	7
Rate (Mbps)	54	48	36	24	18	12	9	6
Slot length (μs)	174	186	218	282	342	470	594	846
Atomic slot usage	1	2	2	2	2	3	4	5

μs and the standard deviation is 0.09 μs. This confirms that although level-2 synchronization is slightly worse than level-1 synchronization, the error can still be within 1 μs. It is worth noting that the error comes from not only the SRT-WiFi devices, but the SDR measurement since the sampling rate we use in the experiment is 20 MHz. With a higher sampling rate of SDR, more accurate results are expected.

B. SNR Measurement

In the second set of experiments, we test the SNR measurement performance of SRT-WiFi, and the setup of the testbed is shown in Fig. 9. We use two SRT-WiFi devices, one for AP and one for station. The TX connector of the station connects to the input of a signal combiner. The other input of the signal combiner connects to a USRP device (USRP1). The combined signal is then divided by a signal divider into two ways, one to another USRP (USRP2) and the other connected to the RX of the AP. Both RX of the station and TX of the AP use the antennas. During the experiment, the signal from AP to the station goes on air while the signal from station to AP goes through the cable. We use USRP1 to add controllable noise to the signal from the station to the AP. The AP measures the SNR. At the same time we use USRP2 to record the same signal as the one received at the AP. We then compute the SNR value from USRP2 as the ground truth and compare the results from AP to evaluate the SNR measurement performance.

Fig. 10 shows the standard deviation measured from both the cross-correlation method and the signal power method. We only test the SNR from 7 dB to 30 dB since when the SNR is lower than 7 dB, the connection between the AP and station is hard to maintain due to the high packet loss rate. The experimental results show that the cross-correlation method outperforms the signal power method in general and thus it is used in all the following experiments.

Fig. 11 shows the packet delivery ratios of the SRT-WiFi device under different SNR measured in the testbed. We vary the SNR values by configuring USRP1 to add noise and let the station send 500-byte UDP packets and count the number of received packets on the AP side. This result gives us the reference to perform rate adaptation.

Table III presents the data rates applied under different channel SNR values and the corresponding slot lengths when transmitting a 500-byte UDP packet. We give the SNR value threshold for each data rate to be used only when the measured SNR value is no smaller than the corresponding threshold. The slot length includes the length of the data packet, the SIFS (16 μs), the ACK and the guard time (10 μs). The settings in Table III are applied in all the following experiments.

C. Rate Adaptation

In this subsection, we demonstrate the effectiveness of the rate adaptation function in SRT-WiFi. In the experiment, we add interference to the channel and measure the data rates and PDR for both SRT-WiFi and regular WiFi networks. In SRT-WiFi, the reception SNR is measured at each device and sent to CNM. CNM then decides the data rate and constructs the schedule for the devices. In this experiment, we set up one AP and one station. We add the interference at the AP side and let the station send UDP packets to the AP and measure the PDR and SNR. The level of interference is not fixed but varied every 0.5 second, meaning that in the first half of each second, the interference rises to a set level while in the next half of that second, the interference shuts down so that we change the interference fast. Fig. 12 (b) shows the measured SNR of the channel and Fig. 12 (a) zooms in part of the measured SNR to show how the interference varies. The minimum SNR values first decrease from 27 to 12 dB and then gradually increase back. The data rates of both SRT-WiFi and WiFi are shown in Fig. 12 (c). Regular WiFi uses the Minstrel algorithm [25] for rate adaptation, which adapts to the interference according to the transmission history. The corresponding PDR is shown in Fig. 12 (d). From the figure it can be observed that when the SNR value is lower than 20 dB,

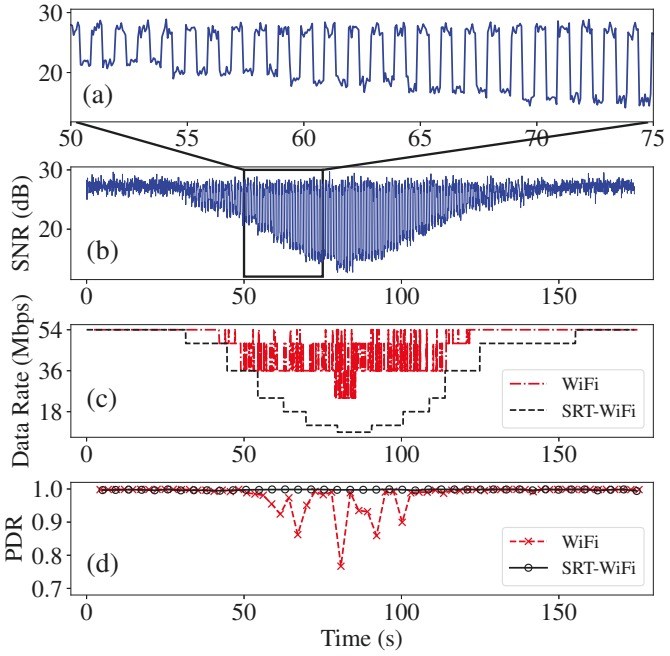

Fig. 12: Data rate and PDR comparison between SRT-WiFi and WiFi.

TABLE IV: Comparison of schedulability among HTS, EDF and Z3.

Utilization	0.3	0.4	0.5	0.6	0.7	0.8	0.9
EDF (%)	45.8	19.5	10.4	5.2	2.6	0.8	0.5
HTS (%)	75.4	48.6	32.5	19.5	9.8	4.2	1.5
Z3 (%)	75.6	49.4	33.9	21.0	11.1	5.0	1.9
EDF & RCS (%)	20.9	11.6	6.9	2.8	1.1	0.3	0
HTS & RCS (%)	30.4	17.7	9.6	4.3	2.1	0.6	0.1
HTS & HCS (%)	52.7	30.1	15.8	8.1	4.5	1.1	0.1

regular WiFi cannot keep stable transmissions. In SRT-WiFi, we employ a conservative rate adaptation method. The CNM buffers the measured SNR values for a time window and uses the rate according to the lowest SNR value in the buffer. Once a lower SNR is measured, the data rate is reduced immediately. The rate does not go back until all the SNR values in the buffer are higher than the SNR threshold of a higher rate (see Table III). Although this method wastes some resources when the channel condition is good, it provides stable transmissions. The performance of this method in the presence of interference is shown in Fig. 12 (c) and it is a step shape without fast changes. Fig. 12 (d) shows the PDR of SRT-WiFi during the test. It is always stable because it measures the lowest SNR and applies the corresponding rate to improve the reliability.

D. Schedule Management

We now present our simulation results and a case study to show the effectiveness of the proposed heuristic method to solve the MSNS-RA problem. In the simulation studies, we evaluate the performance of the proposed heuristic task scheduler (HTS) and heuristic cluster scheduler (HCS). For HTS, we compare it with EDF and an efficient satisfiability modulo theories (SMT) solver Z3 [35]. All the three algorithms are implemented in Python and run in a CPU cluster node with

TABLE V: Comparison of the computational overheads among HTS, EDF and Z3 on large task sets.

Scheduler	Average time cost (s)	Schedulability	Termination ratio
EDF	0.019	0.75%	0%
HTS	2.609	18.37%	0%
Z3	2732.706	5.35%	45.9%

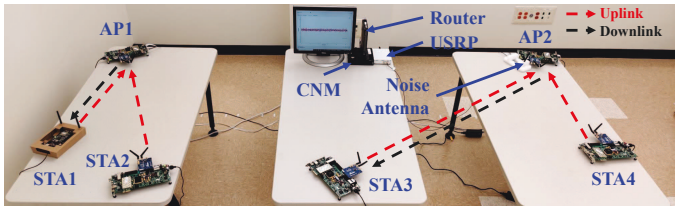

Fig. 13: An overview of the multi-cluster SRT-WiFi testbeds.

Xeon E5-2690 v3 2.6 GHz CPU. The scheduling problem on each channel is formulated as a constraint programming problem, which can be solved by Z3.

We first simulate random task sets to evaluate the schedulability (% of schedulable task sets among all the generated ones) of the three methods under the single-channel single-cluster scenario. For each task set, we randomly generate around 10 tasks with the total channel utilization varied from 0.3 to 0.9. Each schedulability value is generated with the simulation of 2000 task sets. The schedulability comparison of the three methods is shown in row 2 to 4 of Table IV. The results show that HTS is significantly better than EDF while slightly lower than Z3. In the results we keep the infeasible task sets for comparison to show the trend of how the utilization affects the schedulability. We further compare the time costs of the three methods. Here we generate large-scale task sets with 100 to 150 tasks in each task set and use random channel utilization from 0.3 to 0.9. Each schedulability value is still generated with the simulation of 2000 task sets. The results are shown in Table V and it is clear that Z3 costs much more time than HTS. Besides, we set a timeout for Z3 as 5000 second and it reports 45.9% terminated cases. The above results show that HTS can achieve a good balance between performance and time cost.

For HCS, we compare it with the random cluster scheduler (RCS) which randomly assigns the clusters to the channels under the multi-cluster multi-channel scenario. In each task set, we randomly generate 4 to 8 channels and 2 to 10 clusters with 5 to 15 tasks in each cluster. The average channel utilization is also varied from 0.3 to 0.9. The tasks of clusters are assigned to channels by HCS or RCS and then each channel is scheduled by HTS or EDF. If all the channels are schedulable we count it as a schedulable case and finally compute the schedulability. From row 5 to 7 in Table IV, we observe that with RCS, HTS keeps the advantage comparing to EDF. With the proposed HTS scheduler and HCS cluster working together, the schedulability is further improved.

In addition to the simulation studies, we also implement HTS on a multi-cluster SRT-WiFi testbed and perform a case study. The network is configured with two APs (AP1 and AP2

TABLE VI: Parameters of tasks used in the case study.

Task Number	1	2	3	4	5	6
Period (AS)	15	15	15	15	30	30
Deadline (AS)	10	10	10	10	29	30
Transmission Unit Number	1	1	1	2	2	1

TABLE VII: Comparison of SNR, PDR and data rates.

Scheduler & Stage	SNR (dB)			PDR (%)			Rate (Mbps)		
	AP2	STA3	STA4	AP2	STA3	STA4	AP2	STA3	STA4
Heu.@S1	20.8	21.1	26.2				36	36	54
Heu.@S2	20.6	15.9	19.0	98.98	98.94	97.24	36	18	24
Heu.@S3	20.5	14.2	16.8				36	12	18
EDF@S1	20.8	21.1	26.2		98.26	98.20	36	36	54
EDF@S2	20.6	15.9	19.0	98.15	98.42	24.5	36	18	54
EDF@S3	20.5	14.2	16.8		92.31	0	36	18	54

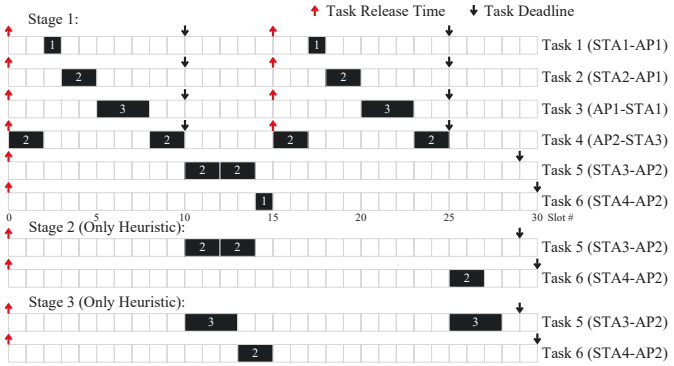

Fig. 14: The task schedule constructed by the MSNS-RA heuristic.

for Cluster1 and Cluster2, respectively) in one channel and each AP is connected with two stations (STA1 and STA2 in Cluster1, STA3 and STA4 in Cluster2). Fig. 13 gives an overview of the testbed where CNM and APs are connected to a router to form a backbone network. A USRP device is used to generate interference and it is placed next to AP2. As shown in Fig. 14, we assign a task to each link and the task specifications are summarized in Table VI. Each instance of the tasks sends a 500-byte UDP packet. With a fixed packet length, the transmission time of the packet depends on the data rate. In the experiment, we set the atomic slot (AS) length as 174 μs which is for transmitting a packet at 54 Mbps.

In the experiments, STA1 and STA2 in Cluster1 transmit to AP1 in the uplink and AP1 transmits to STA1 in the downlink. In Cluster2, AP2 synchronizes with AP1 and transmits to STA3 in the downlink. At the same time, STA3 and STA4 transmit to AP2 in the uplink. During the experiments, we add interference with three levels as three experiment stages so that links need to adapt to proper rates to achieve good PDR. We first apply HTS in Stage 1 and Fig. 14 shows the constructed schedule. We then increase the noise level to Stage 2 and Stage 3, respectively. The measured SNR, PDR and applied data rates of links in Cluster2 (sender name is used to mark a link) are summarized in Table VII (a PDR of multiple rows is the average PDR). The link quality of STA3 and STA4 drop significantly in each stage because the interferer is placed close to AP2. The SNR of STA3 and STA4 drop to 14.2 dB and 16.8 dB, respectively, therefore the rates of STA3 and STA4 drop to 12 Mbps and 18 Mbps, respectively, to adapt to the interference. On the other hand, STA1, STA2 and AP1 are barely affected by the interference and their average PDR are 98.59%, 98.53% and 98.24%, respectively.

We then evaluate the performance of EDF under the same experiment settings. EDF is only able to generate a feasible schedule in Stage 1 and devices cannot require more atomic slots when the interference level increases. With this constraint, in Stage 2 and Stage 3, the data rate of STA3 can only drop to 18 Mbps while the rate of STA4 keeps the same. This causes the PDR of STA4 to drop significantly in Stage 2 and its connection breaks in Stage 3. These results confirm that our proposed heuristic method can generate feasible schedules and keep higher PDR in noisy scenarios compared to EDF.

VII. Conclusion and Future Work

This paper presents the design, implementation and performance evaluation of SRT-WiFi, a high-speed real-time wireless system with full stack configurability that is based on software-defined radio (SDR) platform. We discuss the design principles of the programmable logic and processing system of the SRT-WiFi system and show the advantages of SRT-WiFi on high-precision synchronization, queue management and SNR measurement-based rate adaptation compared to existing real-time wireless solutions. We further formulate the multi-cluster SRT-WiFi network scheduling problem with rate adaptation (MSNS-RA) and propose an effective heuristic solution to solve it. The performance of the system and the proposed algorithm are thoroughly evaluated in our SRT-WiFi testbed.

For the ongoing and future work, we shall keep improving SRT-WiFi to support evolving features like higher bandwidth, multiple-input and multiple-output (MIMO), beamforming and orthogonal frequency division multiple access (OFDMA) in IEEE 802.11n/ac/ax. Two features under development include 1) higher throughput with 40 MHz bandwidth and single-user MIMO (SU-MIMO) in IEEE 802.11n/ac supporting multiple data streams to be transmitted simultaneously and providing more choices on data rate selection for SRT-WiFi, 2) multi-user multiple-input and multiple-output (MU-MIMO) in IEEE 802.11ac enabling the AP to transmit packets to multiple stations in one time slot which enhances the flexibility of scheduling and further reduces the jitter and latency. Besides the throughput, we also consider more complex network topology like the ad hoc mode. And we will deploy our system in real industrial testbeds to test the performance with interference pattern of real industrial environments. We shall evaluate how these features will affect the resource management in SRT-WiFi to improve the throughput and further reduce the transmission latency in SRT-WiFi networks.

VIII. Acknowledgement

This work was partly supported by the National Science Foundation under Award CNS-2008463 and TI-1919229.

REFERENCES

[1] F. Tramarin, A. K. Mok, and S. Han, "Real-time and reliable industrial control over wireless lans: Algorithms, protocols, and future directions," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1027–1052, 2019.

[2] X. Guo, S. Han, X. S. Hu, X. Jiao, Y. Jin, F. Kong, and M. Lemmon, "Towards scalable, secure, and smart mission-critical iot systems: Review and vision:(special session paper)," in *2021 International Conference on Embedded Software (EMSOFT)*. IEEE, 2021, pp. 1–10.

[3] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE transactions on industrial informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[4] J. Song, S. Han, X. Zhu, A. K. Mok, D. Chen, and M. Nixon, "A complete wirelessHART network," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, 2008, pp. 381–382.

[5] The International Society of Automation, "ISA 100.11a." [Online]. Available: https://www.isa.org/standards-and-publications/isa-standards/isa-standards-committees/isa100

[6] R. Steigmann and J. Endresen, "Introduction to WISA: WISA-wireless interface for sensors and actuators," *White paper, ABB*, 2006.

[7] D. Dujovne, T. Watteyne, X. Vilajosana, and P. Thubert, "6TiSCH: deterministic IP-enabled industrial internet (of things)," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 36–41, 2014.

[8] "IEEE 802.15 WPAN™ Task Group 4." [Online]. Available: https://www.ieee802.org/15/pub/TG4.html

[9] Y. Cheng, D. Yang, H. Zhou, and H. Wang, "Adopting IEEE 802.11 MAC for industrial delay-sensitive wireless control and monitoring applications: A survey," *Computer Networks*, vol. 157, pp. 41–67, 2019.

[10] G. Cena, L. Seno, A. Valenzano, and C. Zunino, "On the performance of IEEE 802.11 e wireless infrastructures for soft-real-time industrial applications," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 3, pp. 425–437, 2010.

[11] L. Seno, S. Vitturi, and F. Tramarin, "Tuning of IEEE 802.11 MAC for improving real-time in industrial wireless networks," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*. IEEE, 2012, pp. 1–8.

[12] G. Tian, S. Camtepe, and Y.-C. Tian, "A deadline-constrained 802.11 MAC protocol with QoS differentiation for soft real-time control," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 2, pp. 544–554, 2016.

[13] H. Trsek, J. Jasperneite, and S. P. Karanam, "A Simulation Case Study of the new IEEE 802.11 e HCCA mechanism in Industrial Wireless Networks," in *2006 IEEE Conference on Emerging Technologies and Factory Automation*. IEEE, 2006, pp. 921–928.

[14] R. Moraes, F. Vasques, P. Portugal, and J. A. Fonseca, "VTP-CSMA: A virtual token passing approach for real-time communication in IEEE 802.11 wireless networks," *IEEE Transactions on Industrial Informatics*, vol. 3, no. 3, pp. 215–224, 2007.

[15] D. K. Lam, K. Yamaguchi, Y. Shinozaki, S. Morita, Y. Nagao, M. Kurosaki, and H. Ochi, "A fast industrial WLAN protocol and its MAC implementation for factory communication systems," in *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, 2015, pp. 1–8.

[16] J. Lin, W. Liang, H. Yu, and Y. Xiao, "Polling in the frequency domain: a new MAC protocol for industrial wireless network for factory automation," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 20, no. 4, pp. 211–222, 2015.

[17] Y. Zheng, A. Xu, Y. Song, W. Zhao, and M. Liu, "Industrial wireless deterministic communication based on WLAN: Design, implementation and analysis," in *2009 IEEE International Conference on Communications Technology and Applications*. IEEE, 2009, pp. 274–278.

[18] B. Holfeld, D. Wieruch, T. Wirth, L. Thiele, S. A. Ashraf, J. Huschke, I. Aktas, and J. Ansari, "Wireless communication for factory automation: An opportunity for LTE and 5G systems," *IEEE Communications Magazine*, vol. 54, no. 6, pp. 36–43, 2016.

[19] S. A. Ashraf, I. Aktas, E. Eriksson, K. W. Helmersson, and J. Ansari, "Ultra-reliable and low-latency communication for wireless factory automation: From LTE to 5G," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2016, pp. 1–8.

[20] A. Aijaz, "Private 5G: The future of industrial wireless," *IEEE Industrial Electronics Magazine*, vol. 14, no. 4, pp. 136–145, 2020.

[21] Y.-H. Wei, Q. Leng, S. Han, A. K. Mok, W. Zhang, and M. Tomizuka, "RT-WiFi: Real-time high-speed communication protocol for wireless cyber-physical control applications," in *2013 IEEE 34th Real-Time Systems Symposium*. IEEE, 2013, pp. 140–149.

[22] W. Liang, M. Zheng, J. Zhang, H. Shi, H. Yu, Y. Yang, S. Liu, W. Yang, and X. Zhao, "WIA-FA and its applications to digital factory: A wireless network solution for factory automation," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1053–1073, 2019.

[23] Z. Yun, P. Wu, S. Zhou, A. K. Mok, M. Nixon, and S. Han, "RT-WiFi on Software-Defined Radio: Design and Implementation," 2022. [Online]. Available: https://arxiv.org/abs/2203.10390

[24] M. Dillinger, K. Madani, and N. Alonistioti, *Software defined radio: Architectures, systems and functions*. John Wiley & Sons, 2005.

[25] D. Xia, J. Hart, and Q. Fu, "Evaluation of the Minstrel rate adaptation algorithm in IEEE 802.11 g WLANs," in *2013 IEEE International Conference on Communications (ICC)*. IEEE, 2013, pp. 2223–2228.

[26] J. Xianjun, L. Wei, and M. Michael. (2019) open-source IEEE802.11/Wi-Fi baseband chip/FPGA design. [Online]. Available: https://github.com/open-sdr/openwifi

[27] X. Jiao, W. Liu, M. Mehari, M. Aslam, and I. Moerman, "openwifi: a free and open-source IEEE802. 11 SDR implementation on SoC," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*. IEEE, 2020, pp. 1–2.

[28] S. S. Math, R. Manjula, S. Manvi, and P. Kaunds, "Data transactions on system-on-chip bus using AXI4 protocol," in *2011 International Conference on Recent Advancements in Electrical, Electronics and Control Engineering*. IEEE, 2011, pp. 423–427.

[29] D. C. Mur, "Linux Wi-Fi open source drivers-mac80211, ath9k/ath5k."

[30] Q. Leng, W.-J. Chen, P.-C. Huang, Y.-H. Wei, A. K. Mok, and S. Han, "Network management of multicluster RT-WiFi networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 15, no. 1, pp. 1–26, 2019.

[31] J. C. Eidson, *Measurement, control, and communication using IEEE 1588*. Springer Science & Business Media, 2006.

[32] E. A. Lee and D. G. Messerschmitt, *Digital communication*. Springer Science & Business Media, 2012.

[33] "IEEE 802.11 working group." [Online]. Available: https://www.ieee802.org/11/

[34] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990.

[35] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.