Software Support for Clock Synchronization over IEEE 802.11 Wireless LAN with Open Source Drivers

Aneeq Mahmood, Georg Gaderer, Patrick Loschmidt
Institute for Integrated Sensor Systems
Austrian Academy of Sciences
Viktor Kaplan Straße 2, 2700 Wiener Neustadt, Austria
Email: {Aneeq.Mahmood, Georg.Gaderer, Patrick.Loschmidt}@OEAW.ac.at

Abstract—**Between the wired and the wireless world a synchronization gap in terms of accuracy obviously exists due to the different possibilities of the technologies. This paper investigates means to access WLAN functionality in order to gain system-wide synchronization between access points and clients in order to establish a common notion of time in IEEE 802.11 systems. For this, a novel approach is presented, which uses beacons to transparently transport timing information even in high-loaded wireless LAN networks. Using the presented approach jitter accuracies in the microsecond range can be reached using the open Unix WLAN driver interfaces like madwifi or ath5k.**

I. Introduction

A common notion of time is of utmost importance for various applications. As pretty obvious due to the success of the IEEE 1588 standard in the wired world also a need for synchronized clocks in wireless domains can be seen. Applications for synchronized clocks of mobile nodes are as manifold as the wired ones such as test and measurement and audio-visual data transfer in real-time.

The motivation for this paper, however comes from another application domain, namely the factory automation. For this use-case it is mandatory to synchronize clocks in order to ease channel access schemes like TDMA and to use synchronization services for applications (such as timestamping of sensor data). Typical implementation structures foresee a hierarchical synchronization, where a usually wired and GPS-equipped master node distributes timing information via a wired network to wireless access-points, which themselves have to synchronize the wireless nodes. It is needless to mention that the backhaul network infrastructure has to provide certain support for synchronization. A practical implementation for such a system can be observed in [1] where network controllers inside a factory oversee a large wireless network beneath them. The wired controllers transit a reliable clock source to the underlying wireless networks which should be able to synchronize to the same clock source. However, IEEE 802.11 [2] WLAN standard currently implements its own synchronization scheme in a wireless network using an internal clock, which makes it incompatible for use in aforementioned hybrid networks for factory automation.

Consequently, in this study[1], application layer clock synchronization over wireless channel is investigated when the underlying communication protocol is IEEE 802.11. The goal of the study is to achieve precise clock synchronization in a WLAN using only commercial-off-the-shelf (COTS) hardware. These components pave the way for synchronization investigations using open source Linux drivers. WLAN device drivers have normally been proprietary material of the companies who develop the WLAN hardware and their extensive use in Linux has not been possible without employing tools such Network Driver Interface Specification (NDIS) wrapper [3]. Such tools are only designed to make Windows device drivers usable for Linux without paying attention to timing constraints for synchronization. Therefore, carrying out clock synchronization with such devices/drivers is hardly possible because of the presence of large jitter in timestamps. Some chipset manufactures provided open source drivers but this software development stalled as soon as the chipsets were not available in the market anymore. However, open source WLAN driver development has again picked up the pace in the last few years and some companies now support completely open source driver with which one can communicate directly with the hardware with dedicated application programming interfaces (APIs). Therefore, this paper targets the case where these open source drivers are employed to achieve synchronization over WLAN with software timestamping.

The remainder of this paper is structured as follows: The next section provides a brief introduction into currently available open source WLAN drivers and also deals with different *modi operandi*, which will be used to provide synchronization in this study. Section III provides the state-of-the art for clock synchronization over WLAN and points out what to expect from upcoming amendment to WLAN standard as far as synchronization is concerned. Section IV highlights the issue of software timestamping and its accuracy along with the challenges associated with it. Implementation setup for

[1]This project was partly financed by the province of Lower Austria, the European Regional Development Fund, the FIT-IT project ε-WiFi under contract 813319 and the EU under the FP7 STREP ^{flex}WARE contract number 224350.

experiments and measurements are discussed in Section V. Section VII discusses the measurement and results while the last section is about conclusions and possible future activities and extensions to this study.

II. Open Source WLAN Drivers and Synchronization Means

To understand the role of current open source drivers in WLAN, it is important to look at the advancements in design of WLAN chipsets. Figure 1 provides a block diagram of a typical WLAN transceiver where the incoming signal comes to the antenna, is converted from 2.4 GHz to 11 MHz. This analog signal is sampled, digitized and passed through a signal detector and demodulator to retrieve the original bit stream, which has been transmitted in the first place. Once the MAC layer has received the entire packet, it strips the WLAN header off and then sends the packet payload to the host computer through a dedicated interface for further processing. Previously, the RF mixer block and the baseband transceiver block have been implemented in the WLAN cards as two separate chips. In recent years, with the help of technological advancements, the IC manufacturers have been able two combine these two blocks into a single chip. As a consequence, they have encountered another problem, which is the large footprint of WLAN MAC layer lying in the baseband transceiver as firmware.

To reduce the size of this firmware, chip manufacturers have collaborated with Linux open source community and as a result of that, *mac80211* [4] software-infrastructure is provided. With the help of mac80211, the WLAN *Mac Layer Management Entity* (MLME) has been removed from the baseband transceiver and is moved to the host PC as shown in figure 1. The MLME inside WLAN is responsible for carrying out major management tasks such as device authentication, sending probe requests and responses, power saving etc. The mac80211 extension has shifted the control of management tasks from the actual hardware to Linux user space. In addition to that, Atheros has released the source code of their WLAN driver, which has finally resulted in the development of completely open source *Multiband Atheros Driver for Wireless Fidelity* (MadWiFi) driver [5] and ath5k driver [6]. It is intended that madwifi driver will be completely replaced in the future by ath5k who will support more features and will have better community support. Hence, the more future-proof ath5k driver has been selected to be used in this study.

Consequently, with the help of open source drivers, now only major tasks such as handling of final queues inside WLAN for transmission and reception, channel sensing and random backoff implementation, and communicating with the physical layer for packet transmission and reception is done by the MAC firmware. Tasks such as client association, authentication, beacon handling, assembling for transmission packets etc. is now done inside the driver, which sits on top of the base provided by mac80211.

Another task of MLME, now featured by mac80211, is the higher-layer synchronization supported by IEEE 802.11. With its help, it is possible to send synchronization (`SYNC`) packets containing timestamps from an Access Point (AP) to the client-stations (STAs). The `SYNC` packets are followed by `FollowUp` packets, which contain the lower layer timestamp of the previously sent `SYNC` packet. In this way, WLAN can be used to provide higher layer clock synchronization for the application using WLAN for communication. An obvious shortcoming of this scheme is that the scheme only calculates the offset between the timestamps and delay calculation and compensation is not done. Thus, in this study, on top of WLAN, a synchronization application will be implemented, which will use both offset and delay compensation for synchronization. Another scheme, which the current study investigates for synchronization in WLAN, is to exploit the beacons for carrying timestamps from AP to STAs for synchronization. Such a scheme can be useful in situations such as in [1] where the network is considered saturated by normal traffic and additional traffic for synchronization can not be supported. As beacons are a part of a WLAN and can be modified with device drivers, their usage as `SYNC` packets carrying accurate timestamps will serve the purpose easily.

III. State of the Art

In [7], the clock synchronization over WLAN has been implemented by using a proprietary WLAN card driver for windows. The timer used for synchronization is the internal *Timing Synchronization Function* (TSF) timer present inside the WLAN beacon packets. The value of this timer is sent from master to the slave. The slave uses this time to adjust a virtual clock on the application layer to achieve synchronization with an end-to-end accuracy of about 200 µs. In [8], it is investigated whether high-precision clock synchronization over WLAN is possible or not in the presence of a suitable hardware without any actual implementation for synchronization. Different timestamping delays have been investigated but the focus has been set on timestamping with hardware means and not software. The madwifi driver has been used to implement a software based TDMA scheme in [9]. The TSF timer has been used to implement clock synchronization in this study. However, the achievable accuracy for synchronization is not mentioned but it has been mentioned that the microsecond range accuracy has been achieved using TSF timer for the multimedia applications targeted in the study.

From the IEEE 802.11 standard's point of view, the intended purpose of TSF timer is to provide internal synchronization between APs and STAs. Based on this timer, the STAs can, for example, estimate the next expected beacon arrival and can wake up in advance from their low power state to receive the beacon. However, the standard, till now, does not provide any means to provide application layer synchronization except for the MLME contribution mentioned in the previous section. This issue is supposed to be solved in the IEEE 802.11v amendment [10]. In addition to several other changes a UTC time offset field will be introduced in the beacons. This offset will indicate the difference between the UTC time and TSF timer. Based on this information, the STAs can

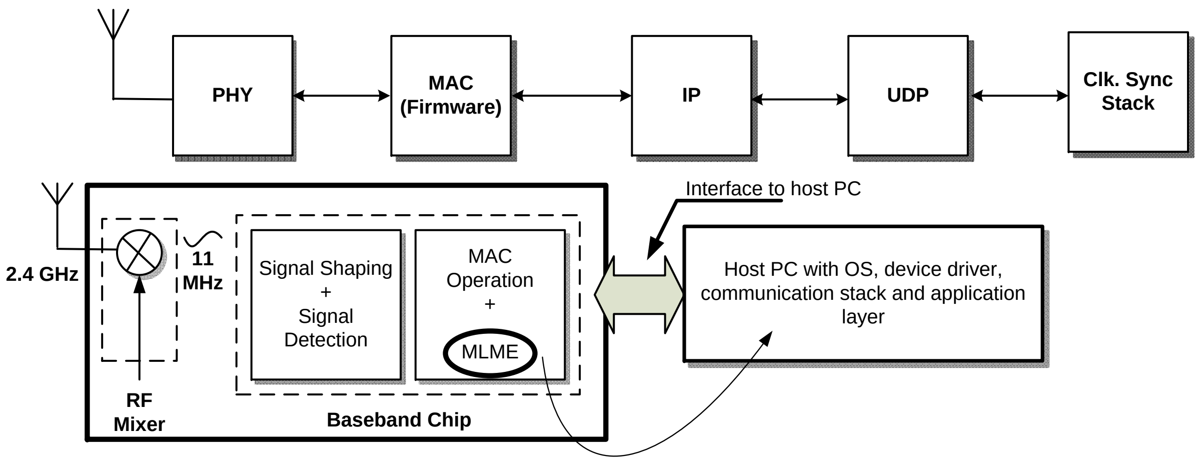

Fig. 1. Comparison between functional blocks and actual implementation of an IEEE 802.11 WLAN chipset

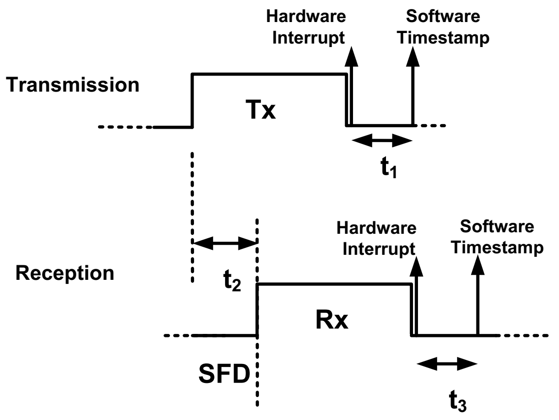

Fig. 2. Overview of delay associated for packet transmission by AP and reception by the STA

achieve synchronization on the system level and not only for typical WLAN operations. In addition to that, support for synchronization protocols such as IEEE 1588 [11] will also be made available. IEEE 802.11 action frames will replace the `SYNC` messages from AP to STAs and the acknowledgment packets from the STAs can then be used as `Delay-Req` for delay calculations. However, 802.11v suggests to use unicast messages for carrying these timestamps which can cause an increase in the network load depending upon how often these packets are sent for synchronization and to how many STAs in the network. The details of the 802.11v amendments are currently not yet available and hence the details about synchronization procedure and accuracy can not be provided.

IV. Timestamping

The achievable synchronization accuracy is directly affected by the precision of timestamps. With software timestamping, it is understood that compared to low-level hardware timestamps the accuracy is compromised because of the additional jitter introduced by the communication stack. To understand the various factors causing the degradation of timestamping accuracy, it is important to understand the data flow from the transmitter (Tx) to the receiver (Rx) side. Figure 2 indicates this flow and also highlights various considerable jitter sources. Once the packet is assembled by the MAC on the Tx side, it is sent out on the air after discovering that the channel is free. As IEEE 802.11 WLAN has a contention based channel access scheme, the packet must stay in the queue inside the WLAN chipset until the channel becomes inactive and becomes available for use by the device.

Once the channel is obtained, the packet is sent out and, in the case of successful transmission, the hardware notifies the device driver with a hardware interrupt. In the interrupt handling routine, the device driver draws the timestamp using *getnstimeofday()* function call and the rest of the packet handling operations are then carried out in the transmission tasklet. This entire time between hardware notifying the driver and the driver drawing the timestamp is shown as t_1 in figure 2. On the receiver side, the same process takes place when the reception process is finished and the time between hardware notification to the driver at the receiver and drawing of timestamp is highlighted as t_3. It should be noted that channel sensing and back-off time does not affect the software timestamps because hardware only notifies the device driver once the packet has been completely sent.

Another major source of timestamping jitter highlighted in figure 2 is t_2. This jitter is the time between the packet leaving the transmitter and beginning of being received at the receiver. This start of process of reception mainly includes the data travelling time from Tx to Rx side, frequency and symbol synchronization at the receiver, settling time of the adaptive gain control, training of the equalizers and symbol detection [12]. Out of all these, the travelling time from transmitter and

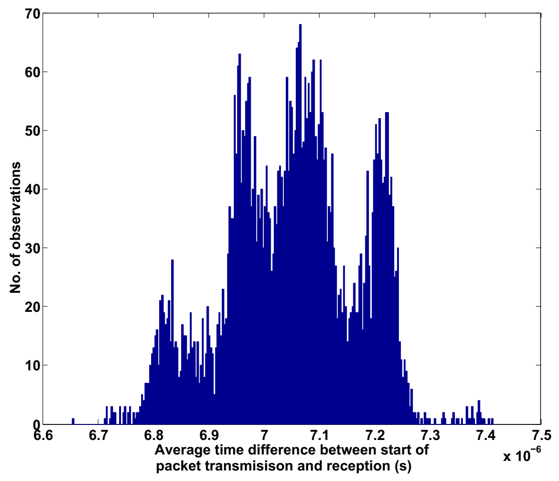

Fig. 3. Histogram of the delay between the start of packet transmission and the detection of SFD at the receiver

receiver in a line-of-sight (LoS) can be considered constant while the rest are random and act as sources of jitter.

The two random delays t_1 and t_3 depend on the system load and process scheduling and are not easy to quantify. However, according to [7], for a 166 MHz processor, the time needed to draw a timestamp is 100 ns. For interrupt handling delays, the exact values for ath5k driver in Linux on testbed under consideration in this study are not yet available. However, studies dealing with measurement of interrupt handling delays for Linux version such as [13], indicate that interrupt handling delay does not exceed 18.4 µs in no-load condition and 67.7 µs in very high loaded condition. This indicate that achieving synchronization in low microsecond range can be possible.

To calculate the delay t_2, it is important to access the hardware interface between the RF mixer and the baseband transceiver. The output of the RF mixer (and input of the baseband transceiver) is a baseband signal with known phase and frequency and is ready to be passed through the analogue digital converters for digital detection and demodulation. As indicated in figure 1, the current WLAN chipsets no longer have this interface as both RF mixer and baseband chip are co-located inside one IC.

Therefore, an older WLAN chipset is used, which is capable of toggling a pin on the baseband chip as soon as packet is to be transmitted in the air. On the reception side, the WLAN chipset toggles a pin when the 16 bit "start of the frame delimiter" (SFD) is detected. The SFD follows a 128 bit preamble and both preamble and SFD are sent at a basic rate of 1 Mbps. Hence, the minimum possible delay after which a packet is detected is 144 µs.

Figure 3 provides the mean delay and jitter between "the start of the packet transmission at the transmitter" and "successful detection of the SFD" for 5000 samples. The mean delay has actually been found to be 151.05 µs while the standard deviation is found to be 120 ns. However, the 144 bits of preamble and SFD are excluded from this delay, and therefore the delay, as shown in the figure 3, is 7.05 µs. This means that 7.05 µs are required for frequency and symbol phase detection and settling of the gain controller on average. It should be noted that Tx and Rx are places in a LoS for these measurements and, in the case of non LoS, signal reflections will be present. This will prolong the convergence of the equalizer coefficients in the receiver chipset, which can increase both delay and jitter.

V. Implementation Setup

Figure 4 provides an implementation setup for carrying out synchronization in this study. The AP acts as the master clock in the system while an STA acts as a slave clock. Both, AP and STA, from hardware point of view, are dedicated firmware development boards with a 533 MHz processor each and running Openwrt [14], which consists of Linux for embedded systems. The mini PCI WLAN cards are supported by bleeding edge ath5k drivers and are interlinked with mac80211. The AP and STA have been put in direct LoS to minimise the impact of signal reflections. To measure the synchronization accuracy, 1 pulse per second (1 PPS) is generated from the serial port of the boards and is fed into an oscilloscope. The difference between rising edges of the pulses from two boards gives a measure of the difference between their respective clocks.

A. *Normal Synchronization Procedure*

The basic synchronization cycle is similar to what protocols like IEEE 1588 employ. The AP sends a `SYNC` packet, containing an application layer timestamp, to the STA. The packet goes through the communication stack in the kernel and is handed to the hardware by the driver. The hardware sends the packet out and notifies the device driver by an interrupt. The driver then draws a 64 bit timestamp from the system clock and sends the timestamp back to the application in the user space. This timestamp is then sent out by the AP in a `FollowUp` packet for the STA. The STA draws a timestamp when the `SYNC` packet is received in the kernel space and the timestamp is then sent to the "timestamp accumulator" in the user space. The accumulator, later on, also collects the timestamp coming in the `FollowUp` packet for master-to-slave offset. To calculate the delay from slave clock to the master clock, the STA sends the `Delay-Req` packet, which draws a `Delay-Resp` message from the AP. From these messages, the slave-to-master difference is calculated. The final offset is calculated by using the "timestamp-offset" and path delay compensation as it is the case in IEEE 1588. Further, it is handed on to the synchronization application, which employs a simple control loop to steer the system clock and is also responsible for making up for frequency drift.

B. *Beacon Modifications*

As discussed in section II, the beacon packets can be utilized to carry application layer timestamps and not only timestamps for TSF timer. The beacon assembly is carried out inside mac80211 and the beacon payload is handed to the hardware by the driver, which appends WLAN header to the payload and transmits it. In ath5k, a beacon alarm is present, which

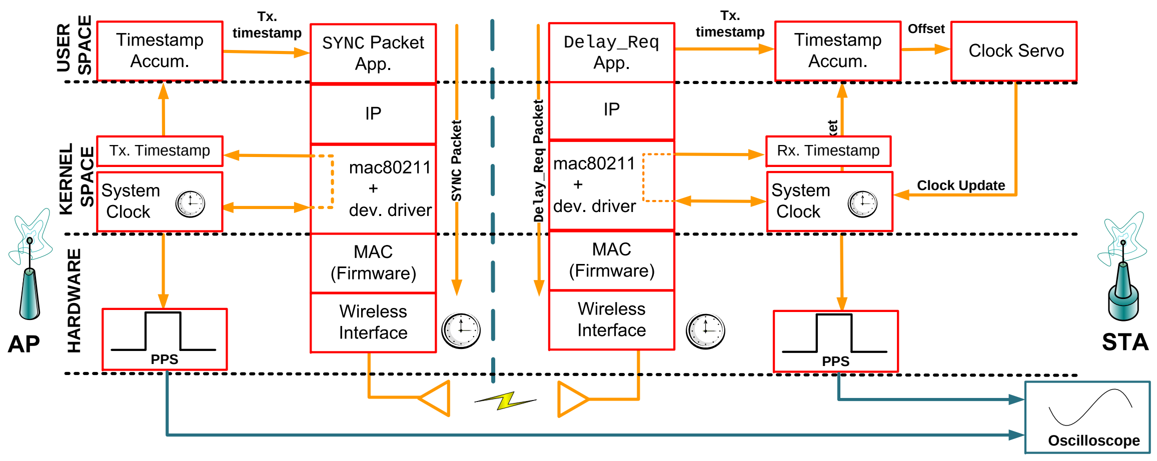

Fig. 4. A block diagram for the system showing activities happening on AP side, acting as master, and on STA side, acting as slave.

raises a software interrupt to ensure that the next beacon is sent out at the exact time according to the chosen beacon interval. The ath5k driver has no dedicated hardware interrupt for transmission of beacons and hence the raising of the software interrupt is taken as the instant to draw the timestamps.

The tasks performed by the driver, once the software interrupt is raised, include stopping all transmission queues, beacon packet assembly and handling the beacon transmission itself. The beacon queue is the highest priority queue present in WLAN chipsets and the contention window for channel access in WLAN is also set to its minimum values for beacons. Hence, with the use of beacons, the most obvious source of jitter is, in fact, the channel accessing delay. However, as the timestamping instant is not really close to the hardware, there will be some additional jitter, which will degrade the accuracy. The timestamp itself is masked as "Vendor" related information inside the packet and hence the WLAN cards from same vendor can communicate after minor modifications in the stack but other cards without these modifications will not be able to be the part of WLAN.

VI. Measurements and Results

For the case where normal `SYNC` packets have been used for synchronization, `SYNC` and `Delay-Req` packets are sent every second to calculate the actual offset while the clock is only adjusted after every four seconds. To minimise the impact of the jitter, the four offset values are saved in a non-linear filter which discards the minimum and maximum offsets. This filtering, thus, tends to provide an average offset while eliminating the effect of those measurements which are outliers. Additionally, all the packets carrying timestamps have been broadcast packets, which removes the necessity to send acknowledgment packets. The final clock offset between the master and the slave is measured only when the initial transient phase of synchronization is over.

Figure 5 shows the clock offset between AP and STA for 10000 samples. The final clock offset achieved has been $-23.6\,\mu s$ with a standard deviation of $6.1\,\mu s$. One major source for the mean offset is the assumption made in the delay calculation for timestamps that delay from master-slave and slave-master is symmetric. However, it has not been the case as the delay from slave to master side is greater than the other way around and hence a constant offset is present at the end. If this offset is known beforehand, it can be taken care of by the synchronization application. However, this offset can not be determined *a priori* and hence the use of `Delay-Req` packets is critical for determining this offset. For beacon based `SYNC`

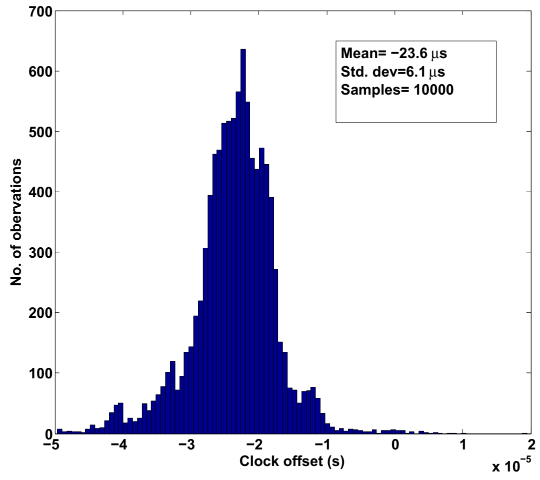

Fig. 5. Synchronization accuracy between AP and STA while using dedicated packets for synchronization and delay calculations

packets, the delay calculation can not be done and the clock offset is calculated as the direct difference between timestamps drawn at the transmission and the reception time. However, the beacon interval is adjusted to be 1 s and the clock is adjusted every four seconds to have the same behaviour as in the case of dedicated synchronization packets. Figure 6 provides the histogram of the clock offset between the master clock and slave clock when beacons are used as `SYNC` packets. As stated in the earlier sections, timestamping for beacons is carried

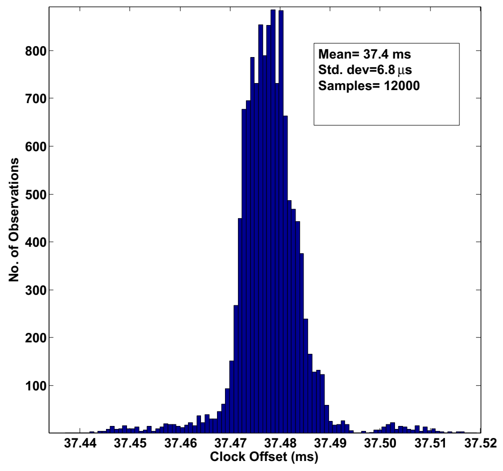

Fig. 6. Synchronization accuracy between AP and STA when using beacons as SYNC packets

out higher in the protocol stack as compared to hardware-interrupt-driven timestamping. This is evident from the mean offset value of 37.4 ms between master and the slave clock. As with the other case, this offset can be taken care of *a posteriori* but is difficult to predict beforehand and may vary slightly with time. Thus, a better timestamping instant has to be chosen if the beacons are to be used for carrying timestamps from AP to STA for clock synchronization.

VII. Conclusion

A software timestamping based synchronization scheme is studied in this scheme for the COTS WLAN cards. Open source WLAN drivers are used to provide software time-stamping low down in the protocol stack. Additionally, the use of beacons is advocated to act as `SYNC` packets and carry timestamps from master to the slave clocks. However, the WLAN hardware does not indicate the transmission of a beacon with a hardware interrupt contrary to the transmission of normal data packets. Hence, the final achievable accuracy with beacons ends up in the millisecond range as compared to a few microseconds with normal `SYNC` data packets. Hence, further investigations have to be done to find better time-stamping instant for software timestamping with beacons. For software timestamping driven by interrupts in the operating system, interrupt-handling delays assert a lot of influence on the accuracy of the timestamps used for synchronization. This random delay must be analysed and quantified under different traffic and system loads before establishing any final verdict on the accuracy of WLAN based clock synchronization.

Acknowledgements

The authors wish to thank Oregano Systems for their kind support within the project as well as Mr. Henning Trsek from the Institute Industrial IT for his support with the Linux drivers.

References

[1] G. Gaderer, P. Loschmidt, and A. Mahmood, “A novel Approach for Flexible Wireless Automation in Real-Time Environments,” in *Proceedings of the 2008 IEEE International Workshop on Factory Communication Systems*, G. Cena and F. Simonot-Lion, Eds., IEEE. IEEE, May 2008, pp. 81–84.

[2] “IEEE standard 802.11 - Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. The Institute of Electrical and Electronics Engineers, Inc., 2007.”

[3] NDISwrapper, “Network driver interface specification (NDIS)wrapper,” http://sourceforge.net/apps/mediawiki/ndiswrapper Last accessed: 22/06/2010.

[4] J. W. Linville, “Tux on the Air: The State of Linux Wireless Networking,” in *Proceedings of the Linux Symposium*, vol. 2, Ottawa, Ontario, Canada, Jul. 2008, pp. 39–46.

[5] Madwifi, “the madwifi project.” [Online]. Available: http://madwifi-project.org/

[6] ath5k, “the madwifi project,” http://madwifi-project.org/wiki/About/ath5k Last accessed: 28/02/2010. [Online]. Available: http://madwifi-project.org/wiki/About/ath5k

[7] M. Mock, R. Frings, E. Nett, and S. Trikaliotis, “Clock Synchronization for Wireless Local Area Networks,” in *Proc. 12th Euromicro Conference on Real-Time Systems Euromicro RTS 2000*, 2000, pp. 183–189.

[8] T. Cooklev, J. Eidson, and A. Pakdaman, “An Implementation of IEEE 1588 over IEEE 802.11b for Synchronization of Wireless Local Area Network Nodes,” *IEEE Trans. Instrum. Meas.*, vol. 56, no. 5, pp. 1632–1639, 2007.

[9] F. Guo and T. Chiueh, “Comparison of QoS Guarantee Techniques for VoIP over IEEE802.11 Wireless LAN,” in *Proc. of 15th Annual Multimedia Computing and Networking Conference (MMCN 2008)*, January 2008.

[10] IEEEP802, “Status of Project IEEE 802.11v, Wireless Network Management,” Last accessed: 22/06/2010. [Online]. Available: http://www.ieee802.org/11/Reports/tgv_update.htm

[11] J. McKay, Ed., *IEEE 1588 (tm) 2.1 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. IEEE, March 2006.

[12] R. Exel, J. Mad, G. Gaderer, and P. Loschmidt, “A Novel, High-Precision Timestamping Platform for Wireless Networks,” in *ETFA’09: The 14th International Conference on Emerging Technologies and Factory Automation*, Sep. 2009, pp. 1–8.

[13] P. Regnier, G. Lima, and L. Barreto, “Evaluation of Interrupt Handling Timeliness in Real-Time Linux Operating Systems,” *Operating system review*, vol. 42, no. 6, pp. 52–63, 2008.

[14] OpenWrt, “OpenWrt, Linux Distribution for Embedded Devices,” http://openwrt.org Last accessed: 28/02/2010.