Traffic analyzer front-end for complex IEEE 802.15.4 applications

Dario Duendar, Lukas Hegetschweiler, Marcel Meli
Zurich University of Applied Sciences
Institute of Embedded Systems
Winterthur, Switzerland
Contact: Marcel.Meli@zhaw.ch

Abstract— **The last years have seen a proliferation of the use of wireless communication in different applications. The systems range from simple 2-nodes communication to complex mesh systems capable of covering vast areas. Debugging such systems, especially large mesh networks can be a daunting task. There are few tools that can help. In this paper, we present and discuss the results of a monitoring tool we are developing. The system is modular, based on a deterministic multicore processor. In the proof of concept, each monitoring probe is equipped with several IEEE802.15.4 transceivers, making it possible to monitor several wireless channels at the same time and to implement a mitigation of diversity issues in the monitoring. The transceivers could also be used to generate test frames for the system under test if necessary. The parallel architecture makes it easy to add new modules and to synchronize the sniffers with the most appropriate methods. In this phase of the work, we used DCF77 to synchronize the nodes. The collected data is sent to a common host for analysis with appropriate tools. The results show that the architecture is appropriate and that synchronization should be improved.**

Keywords—802.15.4; ZigBee; microcontroller; XMOS; parallel processing; antenna diversity; sniffer; mesh networks;6LoWPAN;

I. Introduction and Motivation

Properly debugging wireless mesh networks such as those based on IEEE802.15 can be a very challenging task. Reliable observation of the communication frames is needed. Near a good software analysis tool, a front-end for the reliable capture of the traffic is vital. Several tools exist, but they are limited to the immediate surroundings of the transmitting nodes. Mesh networks can cover large areas. The propagation difficulties and variations associated with wireless communications dictate the use of several monitoring devices at the right places. The monitoring probes must be fast enough to capture as many important details as possible. To allow a correct time interpretation, those monitors should also have the same time reference.

We present a monitoring tool we are presently developing and discuss the first results. The system is based on a multicore processor. Each sniffer is equipped with many transceivers, giving it the ability to monitor several wireless channels at the same time and to mitigate RF diversity issues. The transceivers can also be used to generate test frames for the system under test. The parallel architecture makes it easy to add new modules and to synchronize the sniffers with the most appropriate methods. The collected data is sent to a common host for analysis with appropriate tools.

In our teaching and R&D work in the past years, we have often been confronted to the problem of good, easy to deploy and reliable wireless sniffers. With existing tools one also often misses important details because monitoring is slow or because of interference or multipath issues.

- It is important to equip our teaching laboratories with a device where students can observe the traffic of the different wireless systems that are used. The nodes can be placed at crucial locations and the sniffed data sent on a main server for visualization by all that have the authorization. This will obsolete the need of student having an individual sniffer tool. It will also enable all to see the same thing.
- In the case of our research activities, a tool that can easily be configured to deal with new wireless systems will have much value. The coverage of large areas will also help in the developing of mesh network strategies and help make the right decisions for the energy optimization of low power nodes.
- Because of the nature of wireless, repeaters are often needed. Determining their position is no easy task. A scaled down version of the distributed sniffer will also be useful for the installation/debugging of wireless networks in different industrial and commercial applications.

The rest of this document is divided as follows:

- A short paragraph will give references to similar works or tools.
- We will present the design in general terms, showing how modularity has been achieved.
- We will describe the proof-of-concept system.
- The last part will deal with the results of some tests.

II. PREVIOUS WORKS

There have been several attempts to develop tools for monitoring wireless traffic. An interesting analysis can be found in reference [14]

There are some commercial tools that can be used to monitor wireless WPAN traffic. However, we do not know of any commercial instrument that can really achieve a distributed monitoring of WPAN with the flexibility that we need.

References [12, 18] list some systems and analysis tools that can sniff several 802.15.4 (or Ble) channels. They rely on existing USB dongles. Several dongles can be connected on a hub in order to allow a local monitoring of many channels. The time synchronization is achieved by using a dongle to generate a master time. These tools might do for monitoring devices that are in the vicinity of the hub. But they do not support distributed monitoring.

Various chip manufacturers such as Texas Instruments, Atmel or Microchip also have tools that can be used to monitor 1 channel of 802.15.4 traffic, or Ble traffic [3,4,11].

The commercial device in [19] allows a broadband monitoring of all Ble channels, but does not allow distributed sniffing.

In some research projects, elements that go in the direction that interests us have been developed. However, they do not have the flexibility or reliability or precision we require [14,15,16].

Some projects have been done at our own Institute, to prepare the way for this work [13,20,21,22].

The closest to what we need comes from [17]. They have built elements that can be used for distributed sniffing, achieving a synchronization in the order of 1µs with the use of PTP. This however requires appropriate network switches in the Ethernet network. The monitoring probes rely of FPGA hardware, which somehow restricts the flexibility.

III. REQUIREMENTS

Some requirements have been identified as important for the whole system.

Needed is a modular architecture allowing monitoring probes to be placed wherever is needed, in order to “sniff the communication” and report it back. 3 main elements are required for each monitoring probe:

- One or several elements capable of reading the information exchanged by communicating nodes. This will normally be a receiver. In order to deal with different WPAN systems, it should be possible to easily replace the radio or even to mix radios to allow the simultaneous monitoring of different WPAN. This could be useful to observe the impact of interferences in the same environment.
- The system should allow the easy integration of one or more elements for time synchronization. This flexibility will make it possible to synchronize the sniffers using the most appropriate method in a given setting.
- A way of sending data to the central analysis tool. Optionally, a way of receiving data/commands from that tool. This can be wired, wireless or even be a logger.

Each probe should support the monitoring of several channels and allow a channel to be monitored on several receivers. This will help counter some RF effects like diversity.

IV. GENERAL CONCEPT

The general concept can be summarized as a “network of sniffers” (NS). Each sniffer (SF) is made up of one or more monitoring probes (MP), allowing it to monitor identical channels, different channels or a mixture. The probes are built as peripherals of a parallel processor (sometimes called XMOS here). Several XMOS processor boards can be connected together on the same sniffer in order to increase the number of monitoring probes. Each sniffer has a local timer that keeps time for each probe. Each sniffer also incorporates a network time synchronizer (TS) that allows the different timers to be synchronized. Each sniffer sports a communication channel that is used to send collected data to the data processing server (PS) for computing and visualization. The processing server can also send configuration information and commands to the different elements in the sniffer network. In many cases, the data will be transported via Ethernet. It is however also possible to use another communication interface for the host or even to save data on local mass storage.

The monitoring probes are communication transceivers. They can be exchanged to support different wireless standards or even mixed to allow the simultaneous monitoring of different WPAN communication systems. Especially targeted are systems in ISM bands (2.4 GHz and 868 MHz), 802.15.4 based schemes (6LoWPAN, ZigBee …) Ble, proprietary systems.

The synchronization module can also be adapted according to the environment. For instance, GPS modules could be used if the reception of the signals is good enough. One could also implement PTP (IEEE1588) in each sniffer if this is fitting with the network that is used for the data transfer. In this work, we decided to try the use of the DCF77 radio clock. Thanks to the use of low frequency carrier, it is a good option for areas that are poorly covered by GPS.

Central to the whole concept is the parallel processor architecture developed and commercialized by the firm XMOS [5]. That architecture allows a deterministic use of the CPU resources in parallel processes. This is crucial for the tasks needed in the sniffer and the flexibility that we want. There are several versions of that processor architecture that would allow sniffers with more or less resources to be built. It is also possible to connect several XMOS together and use their special features for data or time sharing.

The distributed sniffer architecture described above allows many options. For a practical proof of concept, we built a system to monitor IEEE802.15.4 networks. We based it on a development board from XMOS, available on the market [6].

That board has some restrictions, but is good enough for the first steps in implementing the network sniffer. That system is continually improved. The results will be used at a later stage for the implementation of a better version.

We will first describe the main elements of the basic sniffer and then present some tests results.

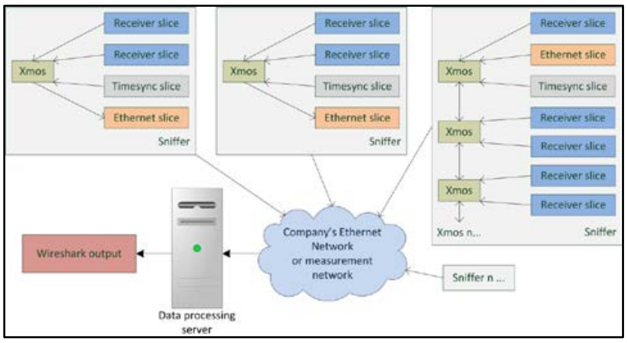


Fig. 1. Block diagram showing the monitoring system as a whole.

A. *The XMOS processor board*

Every sniffer is built in a modular way, equipped with 1 or more slicekit boards (SK). The slicekit we used integrates a XS1-L16-128 XMOS processor [5,6]. It can accommodate 4 slices. A slice is a hardware part that can be plugged into a SK board. Every SK board has 4 slots to extend the board with slices. There is also an additional interface for programming or debugging. In this project we have used the following slices:

- The Ethernet slice (max one per sniffer). This is used to enable an Ethernet connection and send the sniffed data to the server.

- The time synchronization slice (max one per sniffer). This is used to synchronize the different sniffers. In a first version, the time synchronization uses a DCF77 receiver.

- The radio slice (any number per sniffer). Radio slices contain an 802.15.4 radio, used to listen to the wireless communication traffic. Every radio slice can monitor only 1 channel at the time. The radio slice in this project use the Atmel RF233 transceiver [2]. They were developed in an earlier work [10].

SK boards can be cascaded. The second SK can be plugged into the special slot of the first one through its debugging / programming jack. In this way, it is possible to chain an arbitrary number of boards. Since each of these boards has free slots for two more radio slices, it is possible to add as many radios to a sniffer as channels that should be sniffed. It is also possible to sniff a channel with more than one radio slice.

A large number of sniffers can be connected to the server. In practice, we can have 255 since we use 1 byte in the protocol for sniffer identity. This can easily be increased by modifying the protocol and allowing for say 2 address bytes. It is evident that the network needs to be fast enough to allow the transfer of all sniffed data.

The user is completely free to set a single sniffer with several radio slices up and to place sniffers wherever he wants, as long as the needed Ethernet connection and signal for synchronization are available.

An IPv6 network is used to deliver the sniffed data to the server [9]. IPv4 is also possible for the XMOS processors. This should be good enough for most cases where an Ethernet network is available. It will also be possible to implement another physical communication channel for the link between sniffers and the host controller (CAN, WLAN, Wireless …). What would be needed is the corresponding slice and appropriate firmware.

A Linux computer can be used as server. It should run python and Wireshark. The data received from the sniffers is processed and piped. In Wireshark, this pipe is set as network interface and every processed packet appears on the well-known Wireshark GUI. Wireshark can handle large amounts of data, comes with powerful filter functions and will present the data in an acceptable way.

B. *The XMOS processor*

The 32-bit microcontroller we used has two physical cores, each with 8 logical cores. Every core can execute independent instructions.

Fig.2 shows the block diagram of the XS1-L16-128-QF124. The two physical cores effectively run in parallel, while the eight logical cores on them run in time slices, organized by a hardware scheduler. This guarantees a minimum number of MIPS per logical core, which defines the XMOS as a deterministic system. The minimal number of MIPS per logical core is equal to the total available 1000 MIPS divided by the number of active logical cores. However, no logical core can get more than 125 MIPS (8 of 16 cores active).

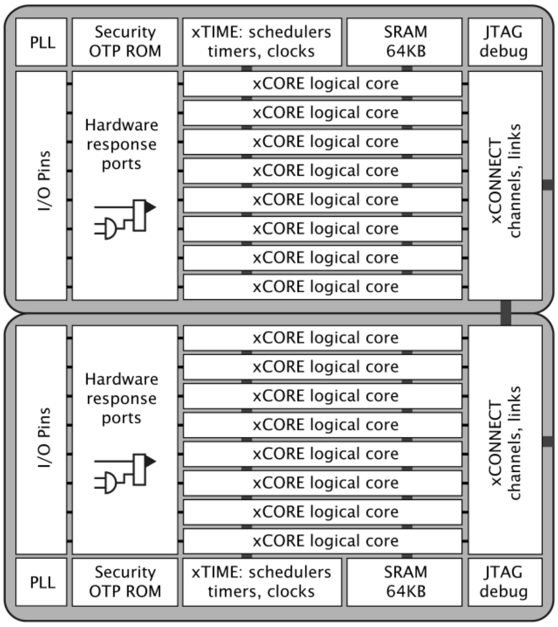


Fig. 2. Block diagram of the XMOS processor used

C. Start-up

When a sniffer is powered on, the time synchronization part starts receiving the DCF77 time signal. While waiting for a valid DCF time to be received, a local timer is started. It is used to timestamp the arriving packets. At the same time, the Ethernet part connects to the server whose IP address is read from the settings on the sniffer.

Upon receiving a wireless frame, the receiver gets a time stamp from the time synchronization slice and delivers the received data together with the time stamp to the Ethernet slice. The Ethernet slice sends all the received data to the processing server using TCP. Finally the server converts the raw data and the corresponding time stamp to blocks that can be understood by Wireshark. This is written in a pipe on which Wireshark is listening.

D. Synchronisation

Synchronizing the sniffers is an important aspect. The goal is a time synchronisation with 1µs (or less) accuracy. Although this could be achieved with GPS, we chose to work with DCF77 in order to gain some experience with that method and compare it with others. The main advantage of DCF77 is the fact that the signal can be received in places where it is not possible to get GPS signals. DCF77 is basically very accurate (1 second lost in every 20’000’000 years [8]). It is not worldwide available, but covers most of Europe (2’000 kilometers around Frankfurt am Main [7]). Similar systems exist in other areas. The DCF signal delivers a rising edge every second.

Several problems appeared during the implementation of the DCF time synchronisation. One of them is the jitter of about 10ms on the rising edges of the second pulse that is used to adjust the internal timer of a sniffer (our own measurements) Reference [8] gives 30ms while considering other parameters. Despite its accuracy, the DCF77 signal reception can be marred with delays. One reason is the architecture used by low cost receivers. Propagation issues also play a role. It is clear that this affects the synchronisation of sniffers.

There are ways to minimize the effects of this problem. For example with a digital signal processing (DSP) based solution. This solution delivers an accuracy of ± 250 µs [23].

We tried an averaging method to reduce the effect of jitter. However, in this phase of the work, we decided to first concentrate on the architecture issues of the distributed sniffer network.

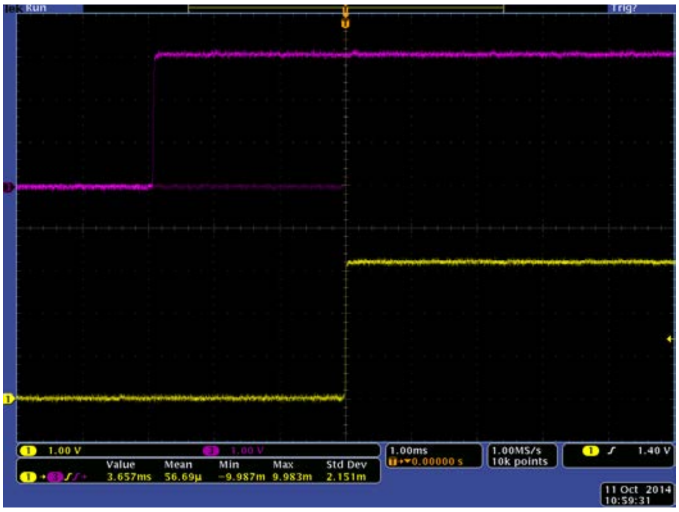


Fig. 3. Difference of the second start edge between two independent DCF receivers (Min: -9.987 ms, Max: 9.983 ms).

E. Data structure

When data is received it will be saved in a local buffer immediately, in order to remain ready for the next packet. The radio process can receive 3 frames (3 receivers are connected). Data are kept in a buffer that is large enough to allow the use of TCP. (See Fig.6 Data flow)

Every received packet is stored in the frame buffer with the structure shown in Tab. IX (packet structure table).

The timestamp contains the whole DCF time of the received packet. The value of the internal timer is used to get to the microsecond level. ED is the Energy Detection information that can be read from the radio. LQI gives information about the quality of the reception. As it is possible to add more sniffers in a distributed network, a sniffer and radio identity is added to have information about who received the displayed frame. The whole results in an overhead of 13 Bytes for every packet.

V. Measurements and Results

In order to test the system, measurements were made at several levels.

In a first category, tests were done on the SK board to measure the performance of the embedded system. Since time stamping and the discharge of the receiver FIFO is done under firmware control, it is important to establish the limits of the system. This helps to determine the local time stamping delays in function of the number of receivers connected and shows how many receivers one could realistically expect to attach to current hardware.

In a second category of tests, different configurations of the sniffers and monitoring probes were tried. Data was captured and sent to the monitoring PC for display using Wireshark.

A. General set up

For testing purposes a wireless (802.15.4) node was programmed to send 2 types of payloads. In one case, frames as short as 5 bytes were used. In a second case, frames built with the maximum of 127 bytes were used. In both cases, test could be made with a minimal Inter Frame Spacing of 228 µs.

Although frames as of 5 bytes do not seem to make much sense, we used them to observe the reaction of the system to frames coming at high speed. The minimal frame spacing of 228 µs was dictated by the limits of embedded system node used to generate the frames.

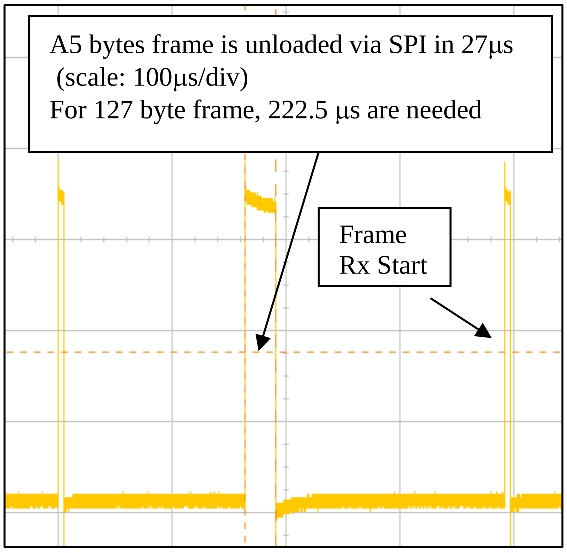
B. *Single radio unloading performance.*

The receiving process runs on one soft core of the SK processor. It can react directly to the end-of-frame event. It immediately starts unloading the frame, over SPI. The SPI clock speed was set to 6.25 MHz. This speed is sufficient to get the packet and additional information before a next frame arrives (receiving time includes the writing of data in the local buffer).

Payload	IFS (µs)	unloading time (µs)
5 Bytes	228	27
127 Bytes	228	222.5

TABLE I.

At maximal payload length of 127 Bytes the XMOS needs almost all the IFS time to read out the packet via SPI. An overlap with the next incoming packet is not a problem as long the readout is faster.



A5 bytes frame is unloaded via SPI in 27µs
(scale: 100µs/div)
For 127 byte frame, 222.5 µs are needed

Fig. 4. Block diagram showing the monitoring system as a whole.

C. *Single radio unloading multi-frame performance.*

Multi-frame performance tests were made by sending several frames containing a payload of 5 bytes. The same was also done with frames containing a payload of 127 bytes. Frames were separated by 228µs. The packets were sent to the sniffer on channel 26. 10'000 frames were sent and the number of received frames counted.

We used our sniffer to collect the results. 2 other sniffers found on the market were used to monitor the same channel.

The Atmel AVR RZ USBstick sniffer [3] and the TI CC2540 sniffer [11]

The results are shown below (Table II).

Frames sent	Payload (bytes)	xmos Sniffer	TI sniffer	Atmel sniffer
10000	127	10000	**5000**	**9998**
10000	5	10000	10000	**1179**

TABLE II.

It can be clearly seen that the 2 other sniffers (from TI and Atmel) are not capable of unloading the received frames fast enough. For that reason, some frames are lost.

At maximal payload, the TI CC2540 seems to drop every second packet. The AVR RZ USB Stick has troubles with the short inter frame interval when small frames are sent. The XMOS sniffer received all packets as expected.

D. *Multi-radio configuration.*

Extending the sniffer with one additional radio is easily done. One additional logical core is needed to run the corresponding process. It runs independently of the other processes.

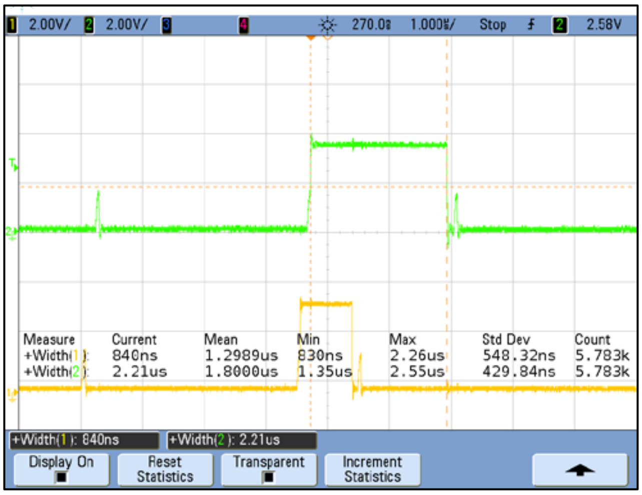
In the worst case configuration, two (or more) radio on the same sniffer will receive the same packet at the same time (same 802.15.4 channel). The two receiving processes influence each other on two points. Firstly, both have to take the timestamp at (nearly) the same time. Secondly, they have to copy the packet into the data buffer. This can be a source of delays.

E. *Time delay in time stamping for multi-radio configuration.*

When a frame arrives on a sniffer, the start-of-frame interrupt is activated and a timestamp is immediately saved. It is later attached to the radio frame. To save the timestamp immediately, the receiving module asks the synchronization module for time information. Saving the actual timestamp takes less than 1µs. What happens if several radios receive a packet at exactly the same time and the receiving tasks ask the synchronization module for a timestamp? We attempted to create such a scenario and investigate the behavior of the system.

Two receivers on the same sniffer were configured to monitor the same channel so that every packet on this channel should be registered twice (once on each radio). The measurements on Fig. 5 shows the duration from the request to save the actual time (rising edge) until the time is saved (falling edge).

Receiver0 (yellow) requests the timestamp before receiver1 (green). The timestamp for receiver0 is saved immediately, but receiver1 has to wait until the synchronisation module has processed all data for receiver0. The delay between the time stamp request of receiver1 to the completion of the request amounts to about 2.55µs.



Measure	Current	Mean	Min	Max	Std Dev	Count
+Width(1):	840ns	1.2989us	830ns	2.26us	548.32ns	5.783k
+Width(2):	2.21us	1.8000us	1.35us	2.55us	429.84ns	5.783k

Fig. 5. Block diagram showing the monitoring system as a whole.

This can lead to a difference of 1 to 3 us (worst case, because of microsecond rounded values) between the recorded timestamps for the two packets. It could also happen that receiver0 requests the timestamp first; this depends on the interrupt signal of the radio. The rising interrupt signals of the receiving radios are delayed by up to 1µs (different physical cores). This leads to the following worst case timestamp shift for the same packet on the same sniffer:

MAX_SHIFT = Max time stamp completion time + Max interrupt delay = 3µs + 1µs = 4µs

A measurement with 10'000 sent packets that were sniffed by two radios on the same sniffer is shown on table III

Timestamp shift (µs)	**Number of packets**	**Percentage**
± 0	0	0.00
± 1	7185	71.85
± 2	2814	28.14
± 3	0	0.00
± 4	1	0.01

TABLE III.

The table shows the local delay of time stamps taken for the same packet with the same sniffer but different radio modules. The results prove the calculation for the MAX_SHIFT.

It seems logical, that the duration of the second timestamp should not take more time than two times the duration of the first timestamp. In fact, there is an overhead due to the XMOS internal communication which claims some time delay. Receiver1 is on another physical core than the synchronization module and receiver0. Therefore more internal communication and synchronization of the two processes across the physical cores is necessary for the timestamp of receiver1. Consequently, the timestamp completion of receiver1 takes 1.35µs in minimum and that of receiver0 only 830ns.

These delays can be reduced by optimizing the time stamping routine and by placing all radios on the same logical core. This will be done as work on the project progresses.

F. *Multi-radio configuration performance*

Timestamp that is saved at a start-of-frame event is copied and attributed to a frame only if the equivalent end-of-frame interrupt is seen. The packet is read out of the radio. The timestamp, sniffer information and information about reception quality are added to the data (13 Bytes overhead). The whole information block is then copied into the frame buffer (Fig.8).

The packet is copied in the buffer before a new packet arrives. In the case of 127 bytes the start-of-frame interrupt arrives just after the copy (more time needed for larger frames). If more radios are added, the process will overlap an incoming packet. This is not a problem as long the end-of-frame does not occur before the packet is copied. In the case of 127 bytes, that happens more than 4ms later.

The receiving procedure was tested with good results. 10'000 packets were sent and received in both radios. This is shown by the total of received packets which is twice the number frames sent by the test generator.

Packets sent	Payload (Bytes)	XMOS Sniffer (reception on both radios)
10000	127	20000
10000	5	20000

TABLE IV.

G. *Multi-radio configuration: TCP Buffer performance*

All the data that is unloaded from the radios must be saved in a bigger buffer. The data buffer is a process that can be called by the radio software module. It gets all the packets and stores them in a 32kB buffer. The data buffer then calls the TCP process to send it out as fast as possible.

Moving the packet in the data buffer must be faster than the receiving of new packets in the radios. Fig. 9 shows the time needed to copy packets received from 2 radios at the same time. The packets are updated with the extra information described in Packet structure.

The XMOS memcpy instruction is very fast as soon as it is started. That is useful in the case of large frames. In the case of small packets, the processor is also busy communicating and waiting for the corresponding process to be ready.

H. *Sniffer total time performance*

The total time performance of the sniffer, from the end-of-frame interrupt until the packet is in the Data Buffer ready to be sent by the TCP was measured. TCP is considered fast enough to handle the throughput of two receiving radios (2*250 Kbit/sec).

The tests were made with the testing node sending as fast as possible (IFS of 228µs). All radios are listening on channel 26. The mean of all measurements were taken. In the case of two radios the worse value was taken.

Frame size (Bytes)	Number of Radios on Sniffer	Mean Process time	Time between frame end interrupts
5	1	26.8 µs	393 µs
127	1	222.5 µs	4.28 ms
5	2	34 µs	393 µs
127	2	232.4 µs	4.28 ms

TABLE V.

In both cases, the sniffer has no problem to receive the next incoming packet. It is possible to add more radios to the system without having troubles reading the packet and storing it in the Data Buffer.

Sending data to the host for display could also be a bottleneck. The data sender is connected to the TCP Slice of the XMOS processor. As soon as a radio delivers a packet in the Data Buffer, it will take the data and send it to the server. In case of heavy wireless traffic, problems can occur if the server is slow answering the request (> 1 ms). Especially in the case small payload (tested with 5 Bytes) the data buffer will overflow. This problem was solved by collecting the data and sending bigger packets of up to 1024 Bytes together to the server.

I. Sniffer performance: Time difference between different probes.

A frame received by different sniffers should theoretically show the same timestamp. However, due to the jitter of the DCF synchronization and the delay at time stamping, there is a difference. In order to verify this, 4 sniffers were placed in 4 different rooms (Fig. 10). After all of the 4 sniffers received a valid DCF77 time, some traffic between the nodes in room0 and room1 was recorded. All of the 4 sniffers received the sent packet. According to the timestamp, the frame was first received by sniffer3, then sniffer2. The displayed order relates to the way data were processed by the TCP part and piped to Wireshark. The last sniffer that received the package was sniffer1. The maximum difference between all the generated timestamps for this frame amounts to 2.532ms. This difference can be as high as 10ms as mentioned earlier in this document. It is mostly caused by the jitter in the DCF77 modules.

Timestamp [s]	Delta last [ms]	Frame number	Sniffer identity
433.082443	…	0	X
459.981012	26’898.569	1	2
459.979224	-01.788	1	3
459.981720	02.496	1	0
459.981756	00.036	1	1
460.017856	36.100	2	Y

Min. timestamp	459.979224 s
Max. timestamp	459.981756 s
Delta for frame 1	2.532 ms

Delays between sniffers

J. Diversity

Since the sniffers can all accommodate several monitoring probes, it is possible to monitor one RF band on several probes of the same sniffer. If the distance between the antennas of the probes is properly chosen, it is possible to see frames that could otherwise not be seen by one probe because of multipath RF effects.

In an example (Table VI), both radios of Sniffer0 are set to the same channel. One node sends a COAP request. The other node responds with an acknowledgment that is only received correctly on one radio. Together with the time information the packet can be still identified. The energy level also shows that the signal of the Node in Room1 is very weak (data from Node1 to Node0).

Delta time	Src.	Dest.	Size	Info	ED	FCS
0.1106300	Node0	Node1	94	Get	33	True
-0.000001	Node0	Node1	94	Get	38	True
0.055322	Node1	Node0	121	Ack	0	True
0.000002	Node1	Node0	126	data	0	False

TABLE I.

K. Using one sniffer to monitor several channels

The different probes that are on one sniffer can also be used to monitor a network working on different channels or networks active on different channels.

In an example (Table VII), channel 25 has ZigBee traffic while channel 26 is used by a 6LoWPAN device sending a COAP information. Data from both networks are seen on the monitoring PC.

Delta time	Src.	Dest.	Info	Ch	ED
0.208902	0.0001	Broadcast	ZigBee Command	25	48
0.050025	0.0003	Broadcast	ZigBee Command	25	12
0.944524	bbbb:: 20c:29..	1111:: e2c9..	Coap, Confirmable	26	31

TABLE II.

L. Monitoring a large area with several sniffers

Table VIII shows measurement data recorded with 4 sniffers. In the upper half of the table, there is a request that is send from a node in room0 to a node in room 1. This request is seen three times, namely by every sniffer that is in the vicinity. The energy detection (ED) value is much higher for the sniffer in the room where the sending node was placed.

In the lower half of the table, one can see the answer of the receiving node. The answer was also seen by the sniffer in its room and the sniffers in neighbour rooms. Here also, it is logical that the sniffer in the same room records the highest ED value.

This measurement shows not only the traffic between the nodes, but also where they could be placed. If a mesh network is operating, it is possible to find a bottle neck in the network and of course it is possible to sniff the whole range of the mesh network because an arbitrary number of sniffers could be placed where needed.

Delta time	Src.	Dest	size	Info	ED	FCS
0.106108	Node0	Node1	94	Confirmable Get	0	True
0.005096	Node0	Node1	94	Confirmable Get	7	True
-0.003968	Node0	Node1	94	Confirmable Get	31	True
0.051889	Node1	Node0	121	ACK, 2.05 Content	0	False
0.003451	Node1	Node0	121	ACK, 2.05 Content	0	True
0.003967	Node1	Node0	121	ACK, 2.05 Content	12	True

TABLE III.

VI. CONCLUSIONS AND FUTURE WORK

We have designed an architecture based on a parallel processor that has the needed flexibility for implementing a network of distributed sniffers. This architecture should allow the reliable monitoring of wireless traffic in simple and complex low power communication networks. The flexibility of the system opens the door to easy addition of modules suitable to different WPAN protocols. Likewise, different timing systems can be used for synchronization in order to match the requirements of the environment of use. The DCF77 system used here needs some optimization for a better time synchronization (ideally under the microsecond).

Future work will focus on 2 aspects:

- We will first optimize the system that has been built, such as to understand the best trade-offs for a future version.
- We will then rebuild the architecture to take advantage of the lessons learned.

ACKNOWLEDGMENTS

Thanks to XMOS for providing us with Slikekits for test.

Thanks to Mr. C. Schlittler for discussions on the XMOS.

We are thankful to SBT in Zug, CH for their help

We are thankful for the support of the CH Federation with CTI grant KTI 14763.1 PFES-ES

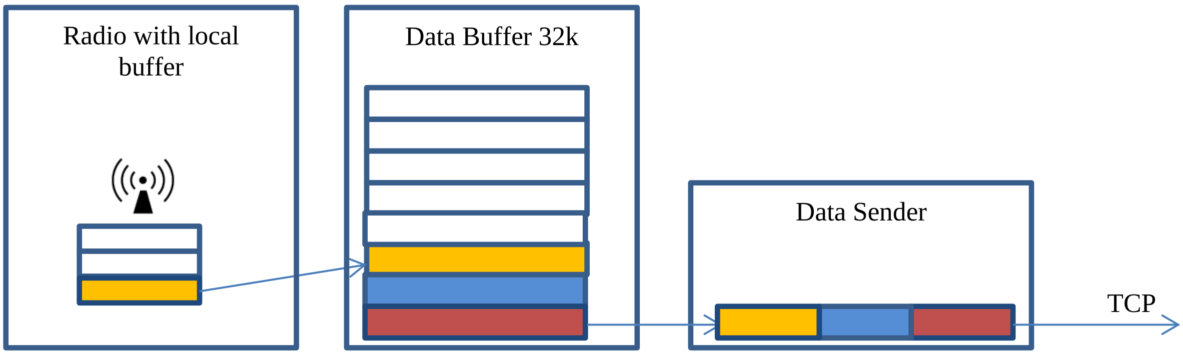


Fig. 6. Data flow

TABLE IV. PACKET STRUCTURE TABLE

Lenth	Timestamp	Data	LQI	Sniffer Nr	Radio & Channel Nr	ED and FCS
1 byte	8 bytes	<128 bytes	1 byte	1 byte	1 byte	1 byte

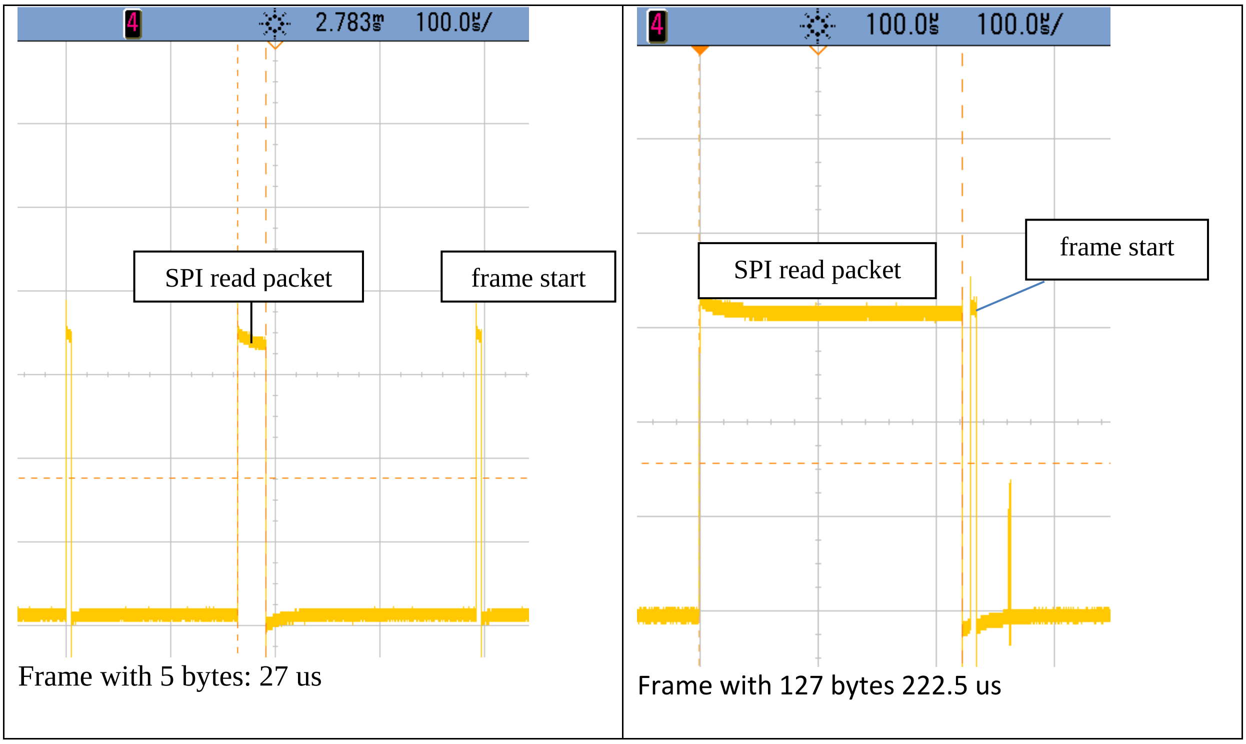
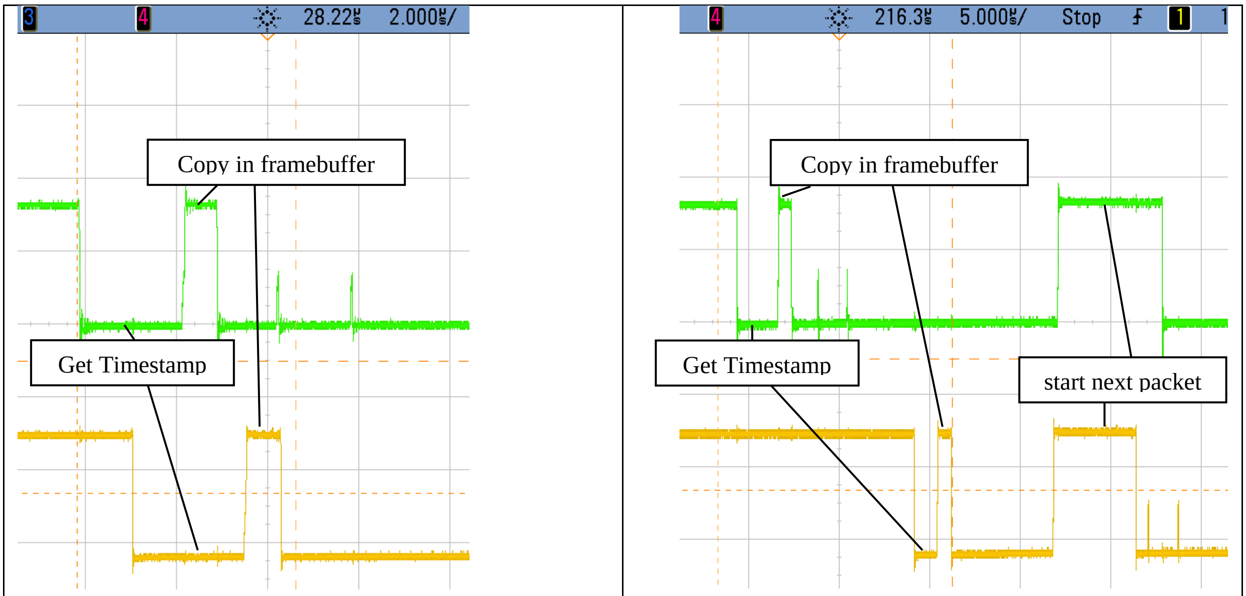


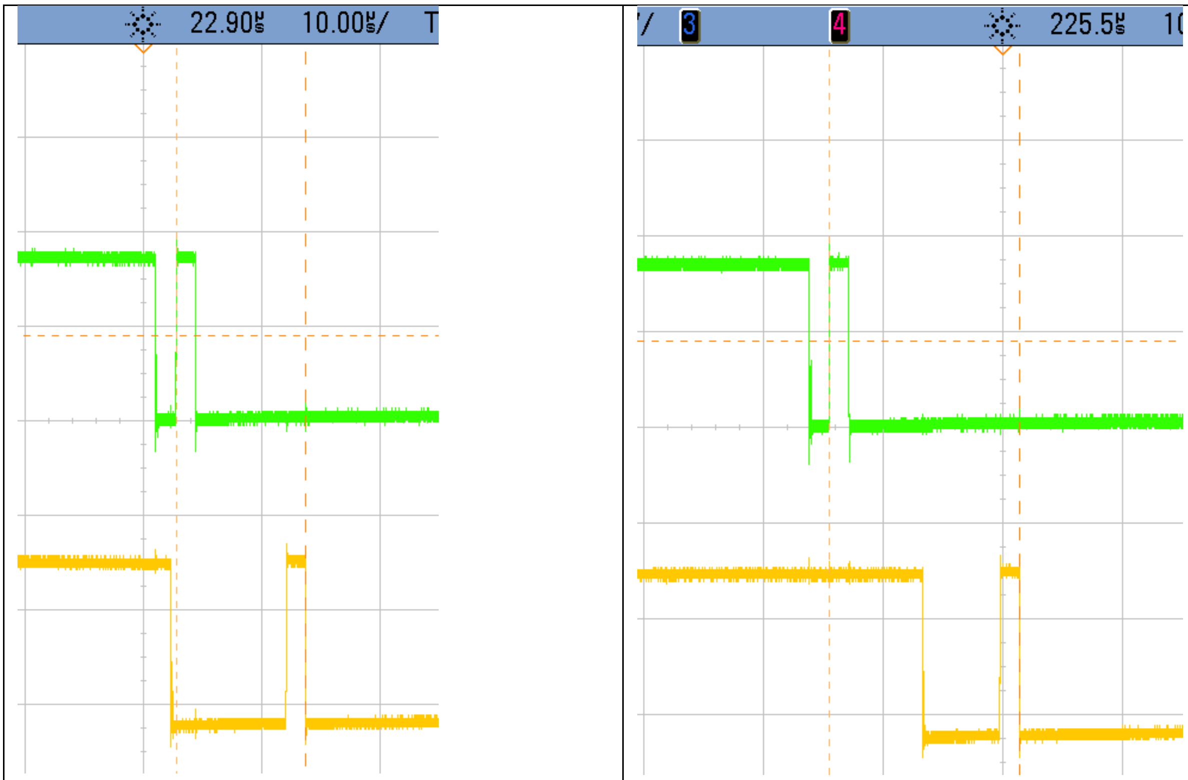
Fig. 7. Timing for radio receiving of short or long frames.



Copy of a 5 bytes frame on both radios: 4.8 us

Copy of a 127 bytes frame on both radios: 12.95 us

Fig. 8. Timing performance with 2 radios



5 bytes + 13 bytes Overhead: 10.9 us

127 bytes + 13 bytes Overhead: 15.9 us

Fig. 9. Buffer timing performance

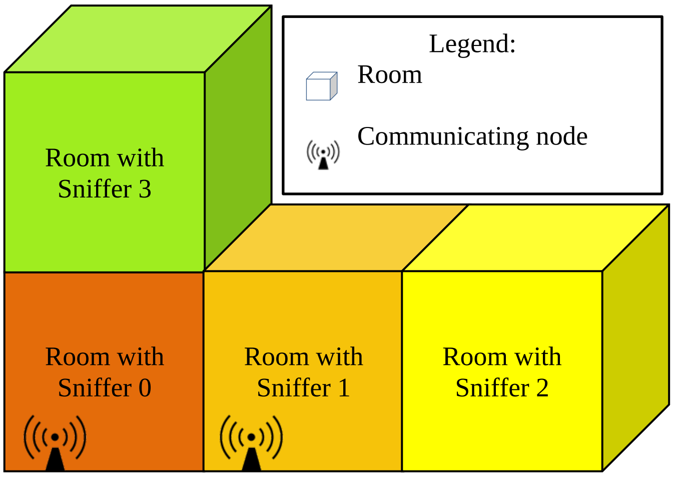


Fig. 10. Test rooms and placement of sniffers and nodes

Fig. 11. Xmos slice kit

REFERENCES

[1] 802.15.4 specifications

[2] AT86RF233Atmel radio: http://www.atmel.com/devices/at86rf233.aspx

[3] Atmel sniffer: http://www.atmel.com/tools/AVRRZ541AVRZ-LINK2_4GHZPACKETSNIFFERKIT.aspx?tab=devices

[4] Microchip zena sniffer: http://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=AC182015-1

[5] xmos firm. http://www.xmos.com/about

[6] Xmos board https://www.xmos.com/products/xkits/slicekit

[7] DCF77 receiver: http://de.wikipedia.org/wiki/DCF77

[8] Information about DCF77: http://tf.nist.gov/general/pdf/2429.pdf (table2 says 30ms with signal propagation time)

[9] Christian Schlittler,6LoWPAN node based on the XMOS architecture, Master thesis, ZHAW 2014

[10] Christian Schlittler, XMOS and 6LoWPAN, Semester project, ZHAW 2013

[11] SmartRF Protocol Packet Sniffer: http://www.ti.com/tool/packet-sniffer

[12] Peryton sniffer: http://www.perytons.com/

[13] Ramon Clematide, Markus Hutzler, Platform for multichannel wireless network sniffer, PA 2008, ZHAW,

[14] Multi-Channel Packet-Analysis System Based on IEEE 802.15.4 Packet-Capturing Modules; Seongeun Yoo, International Journal of Distributed Sensor Networks, Volume 2014, Article ID 216504

[15] A. Koubaa, S. Chaudhry, O. Gaddour et al., “Z-monitor: monitoring and analyzing IEEE 802.15.4-based wireless sensor networks,” in Proceedings of the 36th Annual IEEE Conference on Local Computer Networks (LCN '11), pp. 939–947, October 2011.

[16] L. Choong, Multi-Channel IEEE 802.15.4 Packet Capture Using Software Defined Radio, CLA Networked & Embedded Sensing Laboratory, Los Angeles, Calif, USA, 2009.

[17] P. Ferrari, A. Flammini, D. Marioli, S. Rinaldi, and E. Sisinni, “On the implementation and performance assessment of a wirelessHART distributed packet analyzer,” IEEE Transactions on Instrumentation and Measurement, vol. 59, no. 5, pp. 1342–1352, 2010.

[18] Frontline sniffers: www.fte.com/

[19] Elisys Ble sniffer: http://www.ellisys.com/products/bex400/index.php

[20] Remo Ritzmann, Master thesis on Ble sniffer, 2010, ZHAW

[21] Dominic Ast, Markus Hutzler, Scalable Multichannel Wireless Sniffer ECC2012 : http://www.embeddedcomputingconference.ch/pdf_2012/1C1_AstHutzeler.pdf

[22] Lorenz Sulzberger, Multi channel sniffer for wireless PAN (based on parallel processor), Bachelor thesis 2009, ZHAW

[23] DCF77 Timecode Receiver, Elektor january 2012, Steve Marchant