LABORATORY: WIRELESS SOFTWARE RADIO IMPLEMENTATION (Mini Project report)

Name: Prashanth Kumar Herooru Sheshagiri

Mtr Nr: 383470

1. INTRODUCTION

This report details about implementation of project on developing reliable video link on wireless WARP board. The aim of the project was to develop link layer interface to implement the project with a minimum throughput of 1Mbps and maximum jitter of about 15%, with 0 packets lost or invalid. This project details about the algorithm and programming model developed for both transmitter and receiver. The project is built on WARP v2 FPGA kit which is programmed using standard C language in Xilinx Studio IDE. The link layer algorithm built in this project is based on, and optimised over, CSMA MAC protocol. In this report we start from WARP hardware architecture, and peripherals used in this project and move along to the algorithm model including flowchart and finally conclude with results obtained using a dedicated evaluation program.

1. WARP HARDWARE

Wireless Open-Access Research Platform (WARP), which is the platform that we are using in this lab. It is a FPGA-based SDR platform with an open source repository developed by Rice University. WARP v2 contains FPGA board, several peripheral daughter cards and clock board. RF board built around direct conversion transceiver structure supporting both 2.4GHz and 5GHz band thus providing extensibility to provide customized peripheral cores, interfaces and even hardware interfaces to meet the demands of new applications. It offers high degree of flexibility and programmability for rapid prototyping of any layer in wireless communication. WARP open source repository is also capable of implementing advanced wireless algorithms and offers scalability to flexibly assign computational resources according to the demands of the operating applications (high speed DSP algorithms). WARP Power PC is also compatible with regards to updating on board processors while preserving the validity of other components.

We are using “OFDM reference design” as our PHY layer framework for our project. The main setup of our project consists of a transmitter, a receiver and an interferer all operating in the same channel and in the same operating frequency (i.e., 2.4GHz for our project).

1. LINK LAYER DESIGN

We are designing an operational MAC, developed over CSMA MAC protocol. We are optimising the CSMA protocol to achieve the desired throughput and jitter rate. The WARP repository has 4 reference design framework which we are employing.

1. WARPPHY: Provide PHY-specific functions for interfacing WARPMAC (and MAC code) to the OFDM PHY peripherals.
2. WARPMAC: This framework sets reasonable default values for many of the parameters of the MAC, configures interrupts and exceptions, configures ethernet and initializes the custom peripherals such as the radio controller, the PHY, the packet detector and the automatic gain control block.
3. WARPNET: To configure network packets.

Additionally, we have a custom framework which I am using to prepare EMAC packet by copying the sequence number to the source IP for packet loss and invalid packets verification.

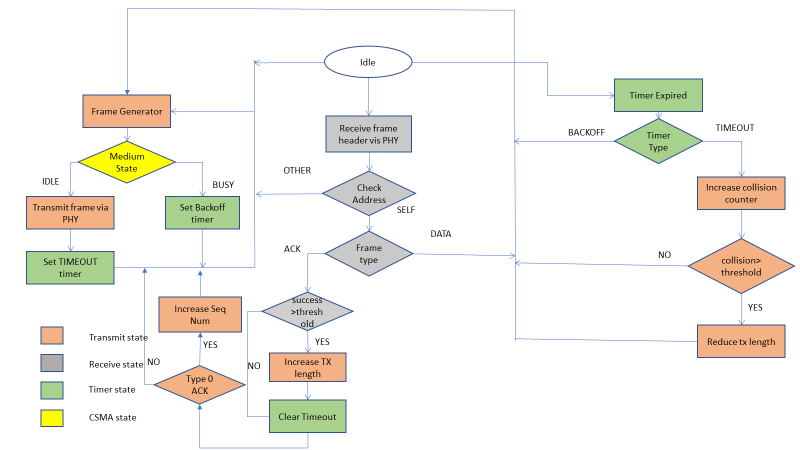


Fig 1: WARP Transmitter Flowchart

The transmitter is designed as shown in Fig 1. The transmitter is designed to generate random packets but with a well-defined header using the frameworks mentioned earlier. Before transmitting, carrier sensing is performed and upon sensing idle channel, the transmission is started and upon encountering busy channel a back off is initiated and after the back off interval, we are transmitting again. To handle error control, the transmitter receives ACK for each transmitted frame and jumps to next packet only after completely receiving the ACK of previous packet. For each successful transmission, we have a success counter and once this counter reaches a certain value, length of the subsequent packet is increased. For each transmitted packet, we set up a timer and a failure to successfully transmit is registered upon reaching timeout stage. Similar to success counter we also have a failure counter and upon reaching corresponding failure threshold, we decrease the length of the packet. To ensure a successful transmission, we initiate packet generator after each transmission attempt, irrespective of failed or successful transmission.

The receiver design is as shown in Fig 2.

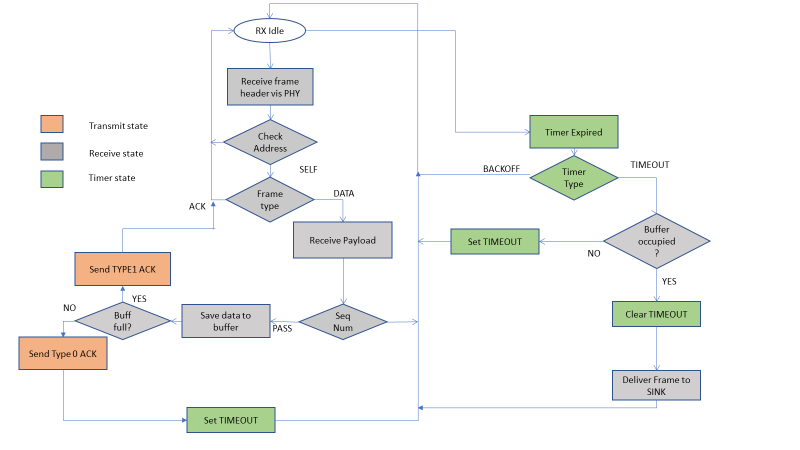


Fig 2: WARP Receiver Flowchart

I performed address verification upon receiving each packet. We process the data once we come to know that the packet is addressed to the same node. If the frame is data, then we proceed to send an ACK and then we verify the sequence number. If the sequence number of the current packet is 1 greater than sequence number of previous packet, then we proceed to save the packet. Otherwise the packet is discarded. To understand how the data is uploaded, we need to first understand the data structure of the way we save the payload. This is illustrated in Fig 3.

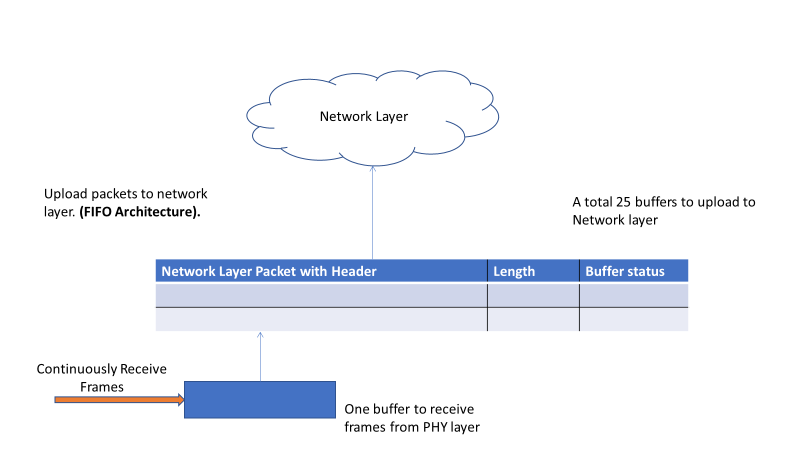


Fig 3: Data structure before uploading

Before uploading packets to network layer, we create a data structure as shown to successfully upload frames. It contains the payload, length of the payload and status of the buffer containing the payload. Length is necessary attribute to prepare packets for uploading. The buffer status turns 0 when data has been successfully uploaded, so that we can effectively reuse the buffer address for next incoming packets.

The receiver creates this data structure for each packets it receives, and upon encountering timeout, releases the payload to network layer. This timer-based operation ensures low jitter rate, provided the timeout duration is very large. But this creates problem with respect to throughput as it increases packet arrival time. The throughput is then compensated by increasing the success counter and optimising the transmitter and receiver code to ensure minimum time is consumed for other loop-based operation.

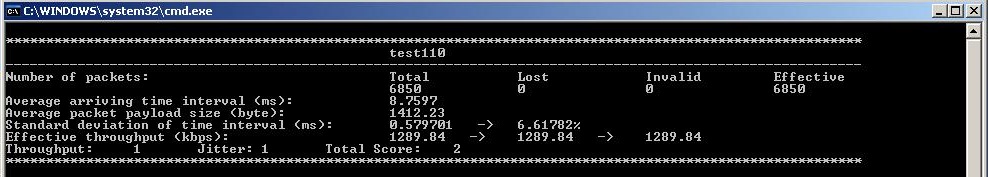
1. OPTIMISATION

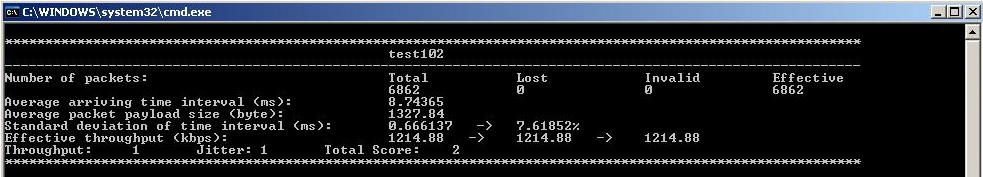
According to 90/10 rule of thumb, 90% of execution time is spent on loop while remaining part of codes take just 10% of time. So, for any real time operation, it is very important to perform loop optimisation to ensure we have as low loop overhead as possible. Particularly when there are such strict requirements for throughput, it is necessary to effectively manage loop overheads.

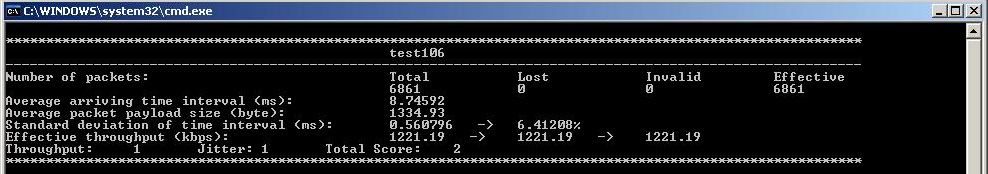
We perform various loop-based optimisation to ensure we have proper trade-off between throughput and jitter. We remove unnecessary operations consuming lot machine cycles (such as print operation and LED turn on/off inside loops). We also perform instruction in lining to save repetitive usage of constant expressions. Computations such as ACK header initiation in receiver and defining frames for random packet generation in transmitter are done in main loop to ensure that such constant operations are done only once. This saves lot of time, since these operations are otherwise performed inside loops.

1. RESULTS

The results obtained for this project was as expected. Fig 4 illustrates evaluation done for the packets as seen by Wireshark.







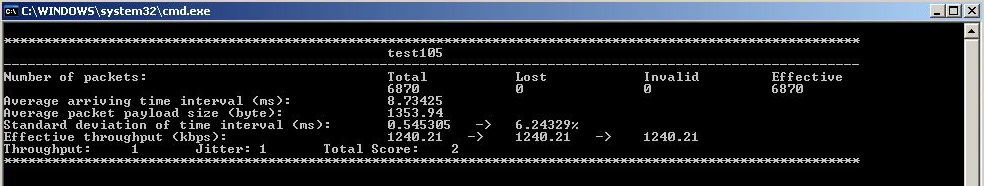


Fig 4: Results provided by Evaluation Program

Fig 4 illustrates 4 test cases we executed in this project. As you can see, the jitter is well below the set target and the throughput is as well, well above. However, we have noticed on several occasions, in other test cases, that due to presence of multiple persons or objects inside the testing area, there exists additional multipath effect, adding interference, due to which the jitter rate increases. So, I had to make sure, in all the above cases, minimum number of reflecting objects are present inside the testing area to ensure maximum achievable results. The results also vary based on environment (temperature, humidity etc.).

6. REFERENCES

[1] <http://warpproject.org/trac/wiki/SoftwareFramework>

[2]http://warpproject.org/trac/attachment/wiki/Workshops/Rice\_2008July/Files/ WARP\_WorkshopExercise\_4\_noMAC.pdf?format=r

[3] <https://en.wikipedia.org/wiki/Carrier-sense_multiple_access>