VIDEO SENSOR OVER ULTRA-LOW POWER RELAY WIRELESS NETWORKS

ONG MAU DUNG
Faculty of Electronics Technology, Industrial University of Ho Chi Minh City
ongmaudung@iuh.edu.vn
DOIs: https://doi.org/10.46242/jstiuh.v64i04.4892

Abstract. Ultra-low-power wireless networks have a vast array of potential applications, which are becoming increasingly achievable as technology gets smaller. Before these systems can be used to their full potential, however, efficient methods of data transfer are imperative in order to keep power consumption low while allowing the devices to cooperate as one unit or group. The transfer of large files, such as image data, presents a particular challenge because of the size limits imposed by the 802.15.4 MAC protocol. Operating sensory nodes at larger distances than normal also results in significant data loss. This paper presents a solution to this issue using a modified sensor network that includes nodes dedicated to listening for packet recovery. A set of algorithms is devised in order to allow nodes to detect missing data from a source as well as offer the ability to request that missing data from one of the dedicated relay nodes. The methods described were tested with the iMote2 device and found to almost double the distance at which two nodes can be placed from each other while still maintaining reliable data transfer. It was also concluded that the Linux driver in charge of passing values from the 802.15.4 chip to the MAC layer does not properly pass the Cyclical Redundancy Check (CRC) result into the MAC frame. As a result, it is not possible to filter out corrupt data at the application layer.
Keywords. Ultra-low power device, video sensor, relay wireless network, tinyOS.

1 INTRODUCTION

Wireless Sensor Networks (WSNs) are a collection of very low-power radio communication devices called network nodes that are cheaply distributed over the surface of the space to be tracked so that real-time signal perception, processing, and communication can be provided, and it is important to have coordinated action with the network nodes [1, 2, 3].

Collections of these devices use advanced networking technology in order to communicate with one another. These devices have unique requirements compared to other microcomputers, as they must operate for extended periods of time, usually on batteries, and communicate through wireless radio chips that must be regulated to conserve energy. New methods for minimizing energy consumption while maximizing reliability are very important for improving the viability of these devices for future use [4, 5, 6].

In this paper, we investigate the use of WSNs to encode and transmit images while minimizing data loss in order to maximize fidelity. Fidelity is defined as the accuracy with which the image transferred matches the original. First, we propose a method that can help recover data lost during image transfer between two ultra-low-power wireless terminals. Next, we will put this algorithm to the test to see if it is suitable for a real-world, low-power system.

A solution to this problem of reliable data transmission is very important to enhance transmission quality and widen the application spectrum in which WSN can be effectively deployed in the future. The proposed solution is tested in three parts. We first transmitted an image from one node to another, which was taken by a camera mounted on the sending node. Second, we step-by-step extend the transmission distance between two nodes and count packet loss. Finally, we add a third network node between the two transmitting nodes to coordinate transmission support and continue to test the transmission distance between the two nodes.

The algorithm was to be written in *C* language for an ARM-based Linux distribution. The general structure of the WSN was strictly defined prior to this research work, and its base source code, as well as the code for interfacing with the camera, were provided to alleviate the learning curve for this research work and give us a solid starting point.

The remaining paper is divided into the following primary sections: Section 2 highlights the WSN concepts and structures. Section 3 presents our proposed algorithm design and implementation. Section 4 describes system testing and results. Section 5 mentions our future development. Finally, the paper concludes in Section 6.

2 WSN CONCEPTS AND STRUCTURES

In this session, we present the equipment used for this paper as well as the basic network structure that was chosen for us. The following is an overview of the hardware chosen for this research as well as a description of how wireless sensor networks were to be structured for the testing of this paper.

2.1 Hardware

The Intel iMote2 is the sensor board used to create the WSNs for this paper. The iMote2 architecture is based on a low-power PXA271 XScale processor augmented by a wireless 802.15.4-compliant radio chip. This allows the iMote device, or mote as it is sometimes called, to perform on-chip calculations as well as easily communicate with other mote devices using a well-defined wireless standard. The device also offers a USB interface for interacting directly with a computer. The device is normally powered directly off of the USB connection, but battery packs for the iMote2 are available that take standard or rechargeable batteries [7, 8].

Since the subject of the project was to have a mote transmit a picture, the C328R camera was used along with an intermediary board and mounted directly to the mote. This camera uses a UART or serial interface for communicating with applications up to speeds of 115.2 kbps. The camera is capable of taking images and outputting them to the application using a well-defined protocol described in the product's user manual. For the purposes of this paper, the camera was used to take images in a 320 by 240-pixel format and compress them to jpeg format before passing them to the application [9].

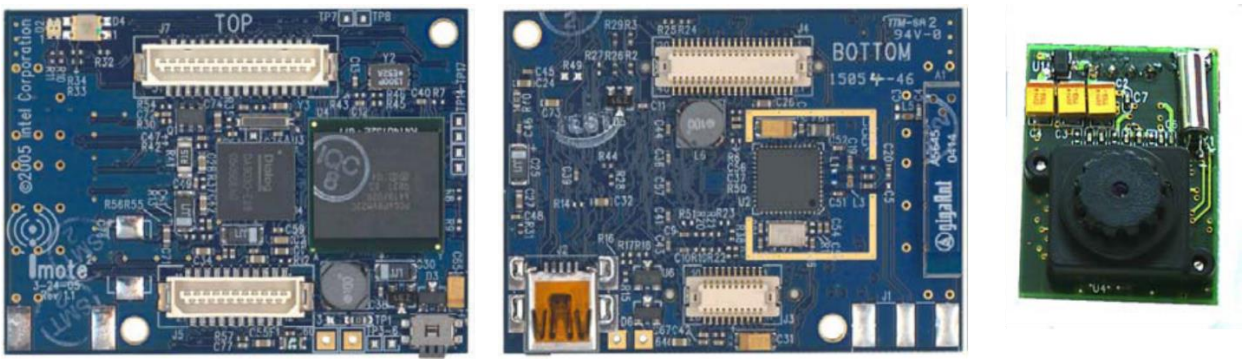

Figure 1: Top and Bottom views of iMote2 hardware, and C328R Camera Module.

2.2 Operating System Environment

The iMote2 is capable of running several operating systems, including TinyOS, Linux, and SOS, but Linux was chosen as the operating system to base this paper on [10, 11, 12]. This operating system offers an interface that most programmers are accustomed to, as well as many APIs that allow the programmer to work more easily and effectively, for example, with devices. The driver included in the Linux distribution for wireless transfer with the 802.15.4 radio chip was based on the TinyOS *MAC* Protocol (TOSMAC). This protocol defines the lower-level data packet structure or frame used to transfer data from one device to another. The frame structure shown in Figure 2 below.

LENGTH	FCFHI	FCFIO	DSN	DESTPAN	ADDR	TYPE	GROUP	DATA	STRENGTH	LQI	CRC	ACk	TIME

Figure 2: TOSMAC frame structure.

Figure 2 shows how the TOSMAC frame is broken up into 14 parts. Each of these parts has a specific purpose to be used by either the radio driver or the application programmer.

The *LENGTH* field is an integer value for the number of bytes included in the *DATA* segment. The

TOSMAC protocol limits the data portion of any given packet to a maximum of 116 bytes, so this value is valid from 0 to 116. The data segment contains the data inserted by the application to be transferred with the packet. The *ADDR* field contains the address of the intended destination node. The *GROUP* field defines the network group within which the particular node is situated. There could potentially be multiple groups of nodes working in any given area.
The *DSN* field contains the sequence number of the packet being sent or received. This value ranges from 0 to 255. The 802.15.4 chip on the iMote automatically sets the value of this sequence number for any given packet. Sequence numbers start at a randomly selected value and increment by 1 for every consecutive packet sent, wrapping around from 255 to 0. For example, a set of three packets is to be sent by an iMote. In this case, the sequence numbers for these packets could be 3, 4, 5, or 255, 0, 1, and so on. These sequence numbers offer an easy way to track certain packets, which will be described in more detail later when determining if packets are missing from a transmission.
The *CRC* field is set by the Linux driver for the 802.15.4 chip when a packet is sent or received. This *CRC* value uses a special bit-check algorithm to determine data integrity. This value is assumed to be automatically set to 0 when a received packet is correct and 1 when the incoming packet is corrupt.
Lastly, the *TYPE* value is a user-specified variable that identifies the type of data that can be found within that particular packet. In the context of this paper, the types that are relevant to this paper include: image header, image data, help request, help reply, and camera node. Each of these types has a numeric value saved to the type of field when sending or checked when a packet is received. The image header type is a packet that contains the size of the image to be received in the data portion of the TOSMAC frame. An image data packet contains image data in its data field.
A help request packet contains the sequence number of the packet it is requesting. A help reply packet contains the data that was requested in its data field. Lastly, the camera node on the packet only requires the type of field to be set in order to function. The data field is an array of unsigned char variables, which are one byte in size each. As a result, if the data being passed into the packet is bigger than one byte, for example, an integer, the data must be split up and placed in the data array byte by byte before being concatenated again on the receiving end.

2.3 Network Structure

Wireless sensor networks, as defined earlier, are simply a grouping of sensor nodes, or in our case, iMote2 devices (also called mote). Each mote can be configured to perform a particular task for the group.
The network structure used for this paper is classified as multi-hop infrastructure-based. This signifies that there is a designated base station node that is connected to a larger network, to which all data is sent. In our case, the base station, or gateway node, as it will be referred to throughout this report, is connected to the Internet and communicates through an open port with a controlling server, from which commands are issued to the WSN [13, 14]. An example of how this type of network is physically structured is shown in Figure 3. There are two camera nodes and one gateway node. Gateway nodes have the ability to communicate with any number of camera nodes, but usually a line of communication is only established with one node at a time.

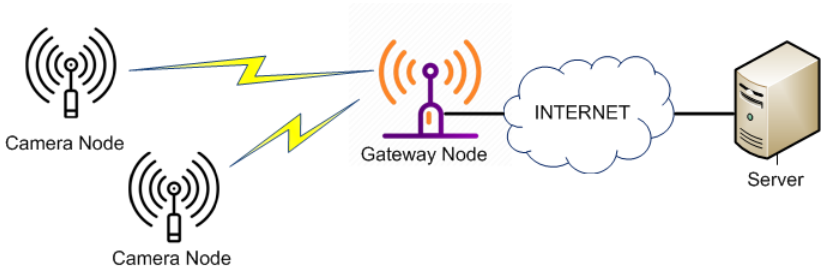

Figure 3: Basic WSN Structure.

2.4 Network Operation

In the context of our research, the basic network interaction normally progresses as follows:

- The server sends the gateway node a command to turn on a specific camera node over the Internet.
- The gateway node sends a specific command to the specific camera node.

- The camera node receives the command and turns on.
- The camera node sends back an image immediately after being turned on.
- The gateway node receives the image and sends it to the server.
- The gateway sends a poll message to all active camera nodes, one at a time.
- The camera node determines if there is motion.
- If motion is detected, the camera node sends an image.
- After a timeout, the gateway polls an active camera node.
- If motion is detected, the camera node sends an image.
- After a timeout, the gateway polls an active camera node.

This polling and retrieval mechanism using a timeout timer continues in an infinite loop. The timeout mechanism prevents blocking the program in the event of a camera node that does not respond. There are many more commands that the server is capable of sending to the gateway node, but the one to turn on a camera node is the only one relevant to this paper.

Images taken by the camera can be anywhere from 4Kbytes to 15Kbytes, with an average size of 7Kbytes. This poses a problem because, as mentioned earlier, a data packet can only contain a maximum of 116 bytes. This means a picture must be broken up or fragmented into, on average, 60 packets if the packet size is 116 bytes. It is this very fragmentation that allows for the possibility of data loss in WSN systems such as the one used here. For all project work and testing, note that the maximum packet size was set to 100 bytes for efficiency.

When a packet is transferred, there are many factors that can prevent the data from being properly received by another node. Packet loss can be caused by general signal degradation, network saturation, and wireless interference. In these cases, some packets simply do not make it to their destination. Although most of the data likely makes it to its destination, building an image with missing data results in pixilation, image shifting, and lost color data. It is evident that in order to create a reliable system that maintains image fidelity, a mechanism for data recovery is important.

2.5 Reliable Network Structure

In order to provide more stable interaction between iMote devices over longer distances, one general idea is to introduce a third node in between two communicating nodes. This third node, called the "watcher node" or "helper node", listens for passing traffic and saves it to memory. In this way, if packets are found to be missing after a transfer, the missing packets can be requested and retrieved from this dedicated node instead of interrupting and interacting with the already energy-intensive processes of the camera node itself. A simple depiction of how this interaction might occur is in Figure 4.

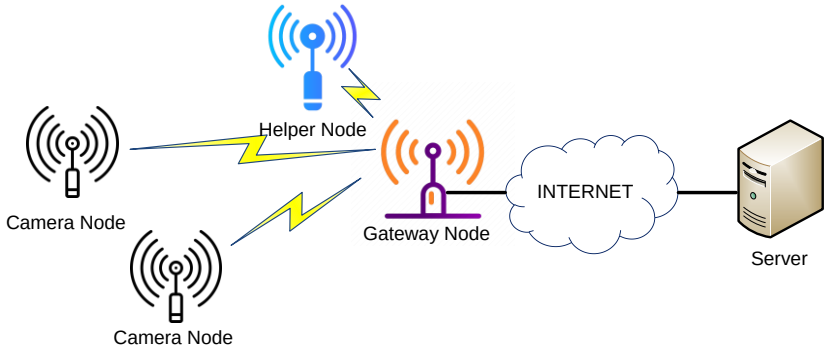

Figure 4: Reliable WSN Structure.

As you can see in Figure 4, a node is introduced between the camera and gateway nodes. Because the watcher node is closer to both the gateway and camera nodes than they are to each other, the probability is significantly higher that all data will be caught and saved by the watcher node, even if the data does not manage to reach its intended destination. This vantage point also ensures that help requests from the gateway reach the watcher node without being lost themselves, although this is still a possibility in practice. Because missing data can be retrieved, this type of model should actually increase the distance that two communicating nodes can be placed from one another. It should also significantly increase the number of successful transmissions at a reasonable distance. Another benefit is that this method decouples the camera

nodes' power consumption from bulky reliable data transfer protocols such as those that use handshaking and acknowledgment packets and allows the average power consumption of the camera node to be much more accurately predicted.

3 ALGORITHM DESIGN AND IMPLEMENTATION

Stemming from the basic idea mentioned above that a helping node could be placed between the gateway node and the camera node in order to facilitate transfer reliability, two portions of code had to be written. The following is a description of these algorithms as well as a description of how they were created and work together to accomplish the requirements of the project. It is worth noting that the camera node code supplied as a starting point for the project was not modified except to change the maximum packet length to 100 bytes. All of the changes made are in the gateway and watcher node codes described in this session.

3.1 Development Environment

The development environment used to program and interact with the iMote device was an Ubuntu distribution of Linux. The particular CPU used on the iMote2 works in conjunction with a class of drivers called USB Gadgets in the Linux kernel to support the creation of a networking device over a USB connection. This allows a Linux machine to connect to an iMote device through the USB port as if it were an Ethernet port. This connection mechanism, referred to as USBNet, is used to communicate with the device and perform some basic configuration. The iMote must first have a static local IP address setup in order to connect to it using USBNet. This initial setup is done using a development board to access the on-board operating system as a root user. Once this was done, the *ifconfig* command was used on the connecting computer to set up a connection over the specific USB port.

Once this setup was complete, the connecting computer could access the iMote using the secure shell protocol with valid user credentials. With this type of connection, it is possible to read and write files to the device as well as run executables. The *SCP* command was used to transfer files back and forth from the device. A cross-compiler referred to as *arm-linux-gcc* was used to produce executables that could run on the iMote system.

3.2 Gateway Node Modifications

The source code for the gateway node that was provided contained a very basic data collection scheme. The algorithm simply counted incoming bytes and placed them immediately and sequentially into a buffer. When all of the bytes were received, an image was built. This only worked if all packets were received in perfect order. This is a very unlikely scenario in practice.

(a) Data Buffer

In order to facilitate reliable data transfer, the first thing to do is create a buffer to hold incoming data, which could also be easily used to identify missing packets after the transmission is complete. This was implemented with an array of *C* structs, defined in *image_utils.h* as *p_data*. The structures we devised contained a length variable as well as a data array that was set to the maximum data size of the packets being sent or received. The length variables were set to 0 on initialization. As mentioned earlier, the sequence number of a packet can be from 0 to 255, and the packets are always sent with sequential sequence numbers. Setting the buffer size at 256 and matching the data segment size within each *C* struct to the maximum packet size limits the amount of data that can be received. Since our project work used a maximum packet size of 100 bytes, the maximum image size that can be transferred is 25,600 bytes. Since images taken by the mounted camera are generally less than 10,000 bytes, this buffer size is more than enough. Because of this, we chose to have the buffer have a size of 256 in order to cover every sequence number once. A function defined in *image_utils.h* called *add_image_data* is useful for quickly and cleanly pushing data into one of the *p_data* structures in the buffer. In order to aid in finding and determining missing data, the image buffer is always indexed by the sequence number of the packet containing the image data. So, if an image data packet is received with a sequence number of 20, the packet data is placed into the buffer at index 20. The reason for this will be further explained in the next subsection.

This data buffer is accompanied by three variables for keeping track of incoming data packets. The first variable contains the number of expected packets; another contains the number of packets already received;

and the third holds the starting index value of the stream of data. All of these values are set to zero on startup. When an image header packet is received, the buffer data for that camera node is flushed and prepared for new image data. It also causes the number of received packets to be set to zero and the number of expected packets to be set to the correct value calculated from the image size specified in the image header packet and the maximum packet size. With the starting sequence number and number of expected packets, it is straightforward to determine what the predicted last packet will be. When image data is received, the number of received packets is incremented and checked against the number of expected packets. If the number of packets received equals the number of expected packets, all the data was received. If this is not the case, but the most recently received packet has the same sequence number as the predicted last packet sequence number, data is known to have been lost, and the program is passed over to the help request algorithm explained in the following section. At any point, if a packet is received that has a *CRC* value of one, the data is discarded. This data can then be recovered using the method described in the next section.

An important point about the method just described is that the reliability of the algorithm is strictly dependent on the fact that the image header packet is correctly received (to determine the number of packets to wait for) and the final packet is correctly received (to determine that the transfer is complete). If this condition is not met, the algorithm breaks down. However, it was assumed that the chances of this condition not being met were low enough for the algorithm to be effective.

(b) Help Request Algorithm

The help request algorithm is only called in the event that a packet is determined to be missing. The algorithm first takes the image data buffer along with the start and ending sequence numbers and determines the packets that are missing. This is done by iterating through the relevant indexes in the buffer and simply checking if the length value is 0. If so, the data is missing. A check list structure, defined in *image_utils.h* as *check_list*, is built, which is made up of an array of integers that represent missing sequence numbers and a length variable initialized to 0. The algorithm first sends a request packet with the sequence number of the last value in this check list to the watcher node and waits for a response in the form of a help reply, also known as a help response packet. This algorithm performs as depicted in Figure 5.

On a reply from the watcher node, the data is put into the image data buffer at the index specified in the last value in the check list. The check list length is decremented. If the check list length value is 0, all data is accounted for and the image is built; otherwise, the next request packet is readied and sent. This repeats, collecting the missing data one packet at a time.

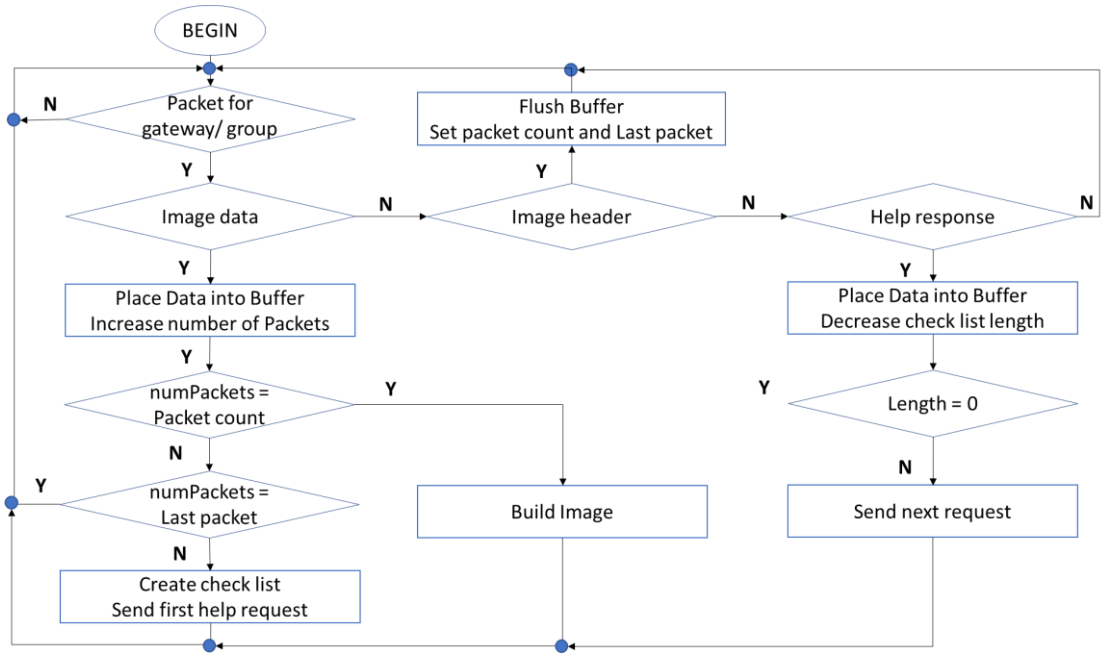

Figure 5: Gateway Node Flow Chart.

3.3 Watcher Node Design

The watcher node is modeled after the gateway node. The program is designed as a loop that is constantly checking for incoming data. Just like the modified gateway node specified above, the watcher node maintains an image data buffer. This buffer is used to save incoming data as well as perform help request operations. The flow of the code is depicted in the following diagram, Figure 6. The program waits for three types of packets: help request packets from the gateway, image data packets from a camera node, and image header packets from a camera node.

If the node receives an image header packet, the buffer is flushed, and all its *p_data* structs reset their length member variables to 0. No image information is actually saved from the image header packet; it simply serves as an indicator to flush the buffer for new incoming image data. If the watcher node receives image data packets, their data is copied into the buffer using the same *image_utils.h* function, *add_image_data*, used in the code for the gateway node. In this way, the data is saved at the index of the corresponding sequence number of the packet. This mimics the way in which the gateway node saves the image data. The receipt of a help request packet from the gateway node indicates that the gateway has determined a packet was missing from a recent transmission. The watcher node first checks if that data is, in fact, in its buffer. This is done by indexing into the buffer at the index specified by the sequence number passed in the data field of the received help request packet. A non-zero length value of the *p_data* object indicates that the data is present. If so, the data is packaged and sent.

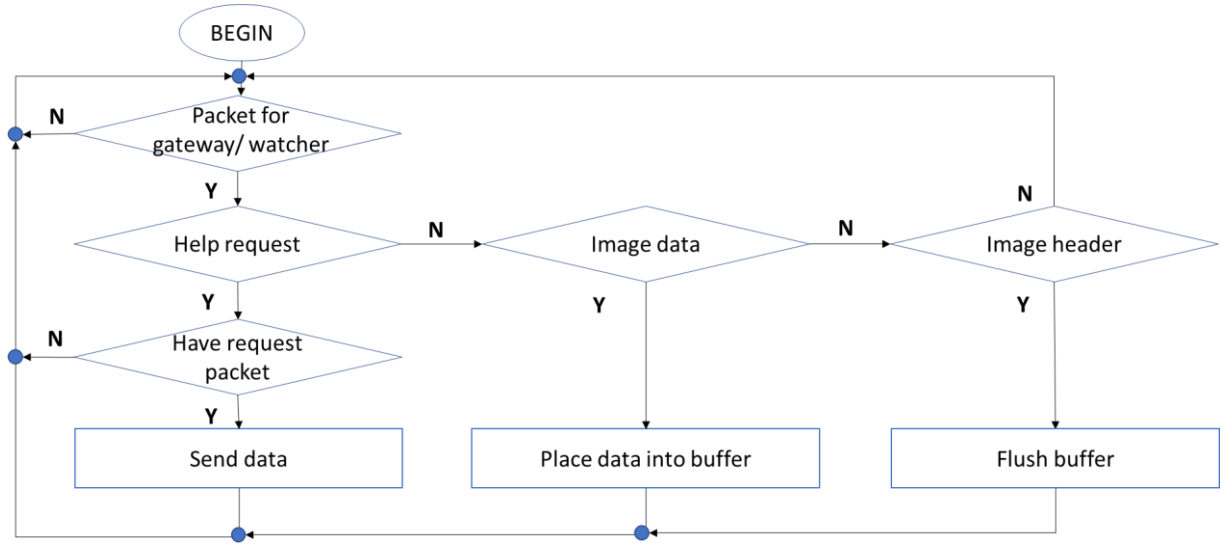

Figure 6: Watcher Node Flow Chart.

In the event that a request is made for a packet that the watcher node never received, the transmission is unrecoverable. This was a design decision. The watcher node could request that the camera node resend certain packets in this case but doing so would negate some of the original benefits of using this method in the first place, namely reduced power consumption. Additionally, this would increase the transfer time, likely past the timeout period of the gateway node, in which case the gateway node would simply request a new image anyway. In this way, the gateway already has a method for circumventing unrecoverable image transfers.

4 TESTING AND RESULTS

4.1 Functional Testing

Functional tests of the programs were performed to verify the correct operation of the code under conditions of no packet loss. This was done to ensure that code not directly related to the modifications made above was not broken by the changes made. For these reasons, the watcher node was not used during these tests.

The above-mentioned gateway and watcher node code was unit tested for each function during development, but the final system was tested by modifying the gateway node to turn on one camera node

and request an image. The nodes were placed within centimeters of each other to ensure little to no packet loss, and the programs were run. Because the gateway automatically and almost immediately turns on the camera node, these tests were accomplished by ensuring that the camera node program was executed before the gateway program. Print statements were strategically placed in order to verify the proper operation of the code. Test images were also copied from the device to verify their quality.

Two main functional tests were performed under these conditions. The first functional test was run by having the camera node transfer an image already saved to memory to the gateway. The second test used the mounted camera to take a picture and transfer it to the gateway node.

4.2 Distance Testing

Tests in which the iMote nodes were placed farther apart were performed in order to verify the correct operation of the system with minimal to high packet loss. These tests were also performed to determine if the system did in fact increase the distance at which iMote nodes can be placed from each other while still maintaining reliable data transfer. For these tests, the watcher node was always placed midway between the two nodes. The tests were performed just as the functional tests above, but the iMotes were pushed farther apart in increments of one meter each time. Several tests were run at each distance to verify the result and collect an average sampling. The data recorded for each test was the number of packets lost and the number of packets retrieved from the watcher node, if applicable. Images of each of these tests were also saved for verification and future reference.

4.3 Saved Image Transfer Results

Functional testing of image transfers using a saved image showed that no packet loss transmitted a perfect replica of the original image in most cases. There were occasional cases where the image was corrupted by a small amount even after receiving all the data. This resulted in slight fragmentation or discoloration. These instances were very rare but worth noting, as this implies a small bug in the code without any modification.

4.4 Distance Testing Results

As the distance tests were performed, it became clear that there was a sharp contrast forming between the numerical results, such as the number of packets lost, and the actual image received, or visual result. For example, during some tests, there was actually no data missing, but the image received was still distorted. The following subsections describe the numerical and visual results separately and attempt to explain their disparity.

(a) Numerical Results

As shown in Figure 7, with practical implementation, it was found that attempting a transfer between two nodes under 7 meters apart without the use of a watcher node resulted in very good reliability, almost always resulting in a perfect transfer and only rare failures. Once the two nodes were pushed past 7 meters apart, however, packet loss increased significantly with distance. After 22 meters, it was discovered that nearly all the packets were lost.

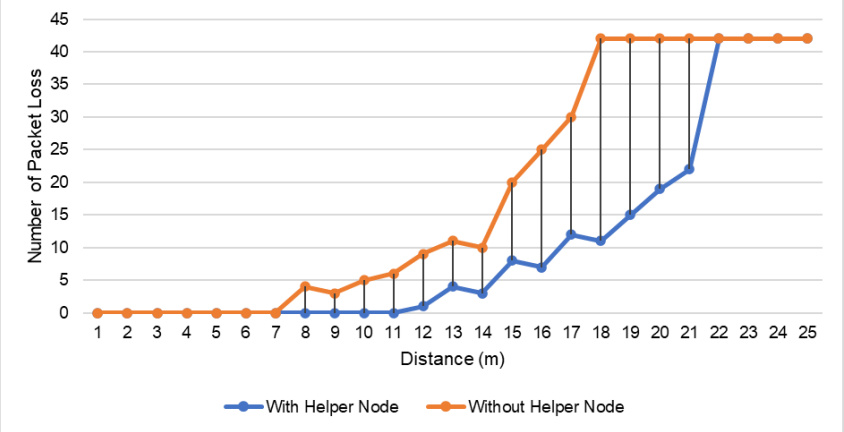

Figure 7: Packet loss v/s Distance transmission results.

Using the watcher node, it was found that reliability up to 14 meters resulted in 0 to 10 packets being lost, but all were almost always successfully recovered. Once the nodes were pushed past 14 meters, it was found that some packets not received by the gateway were also not received by the watcher node. These resulted in a transfer failure. It was also found that after 14 meters, the number of instances in which either the first or last packet, which are required for the correct operation of our algorithm, were not received increased. These tests also resulted in a transfer failure. It can be seen that the designed algorithm succeeds in transferring lost data to the gateway without interrupting the camera node. It can also be shown that the reliable distance between two iMote nodes can be approximately doubled from 7 to 14 meters with the use of our method.

(b) Visual Results

As the distance between the nodes increased, it was found that the fidelity of the image continued to degrade intermittently, even though the watcher node was successfully passing all of the missing packet data to the gateway. Some tests with significant packet loss provided an image that looked perfect, while others produced a very distorted image with much less packet loss.

A byte-wise ASCII printout of an image that recovered from significant packet loss was compared with the same printout of a perfect copy of the original image. This ASCII printout was generated by the *dump_image_data* function in *image_utils.c*. The resulting data was cross-referenced with the actual test output from the Linux terminal, which revealed the specific packets that were lost and recovered during that particular test. It was found that the packets that had been lost and recovered during the transfer were in fact correct when compared to the original image. It was also found that some of the packets that had been received directly from the camera node were actually corrupt. This corrupt data was found to be closely situated around the packets that had been lost and recovered. It was determined that the *CRC* check in the TOSMAC frame, which was assumed to reveal corruption errors in incoming packets, was in fact not being set by the Linux driver.

As was mentioned above, a design decision was made that if corrupt packets were received, the packet data should be treated like a lost packet and discarded. However, because the *CRC* check value was not set, corrupt packets were being interpreted as correct packets. This is most likely the cause of the rare instances in the functional tests where the images were slightly distorted. This issue dramatically affects the fidelity of the image as the distance between the nodes increases. Also, the fact that corrupt packets occur in close proximity to lost packets simply implies that the same disturbances, obstacles, or electronic noise that caused some packets to be lost are also the cause of the corruption in packets sent or received close to that same point in time.

5 FUTURE DEVELOPEMENT

Although our research did reveal that the use of a watcher node does in fact provide all the expected benefits, the use of the algorithms proposed and implemented cannot be fully utilized until the driver used for retrieving data from the 802.15.4 chip is updated. The driver must be reworked to properly pass the correct value from the *CRC* check to the *CRC* field in the TOSMAC frame.

The CC2420 is the actual 802.15.4 chip used on the iMote 2. After reviewing the data sheet for this device, it is possible that a control bit called *MODEMCTRL0.AUTOCRC* is not being set, which tells the chip to perform the *CRC* check in the first place. Alternatively, the driver could simply be neglecting to set the *CRC* field even though a *CRC* calculation is performed. In this case, the reference document indicates that "the most significant bit in the last byte of each frame is set high if the *CRC* of the received frame is correct and low otherwise". Using this information, one could have a simple conditional statement to set the *CRC* field.

Other efforts to improve upon the algorithms presented here include removing or reducing the dependence of a successful transfer on the proper receipt of both the image header and the last data packet of the image. As it stands, these only affect the operation of the algorithm occasionally, but future versions of these programs could implement a timer that is started when an image header is received and times out after a given period if the last data packet is never received. This method removes the algorithm's dependence on the last packet. Another option is to send the first and last packets multiple times to reduce the probability that they are lost.

6 CONCLUSIONS

We have investigated the design and implementation of an algorithm for retrieving data lost during the wireless transfer of an image in a WSN, as well as the testing of this algorithm for its effect on reliable data transfer.

As this paper progressed, an algorithm involving a specialized image buffer and checking mechanism was devised. The provided gateway node source code was modified to utilize this algorithm and communicate with a watcher node, which was coded based on the gateway node model. The watcher node was designed to transmit missing data if it was lost during a transfer between a camera node and the gateway node.

After testing the device configuration and algorithms, it was determined that the watcher approximately doubled the distance, from 7 to 14 meters, at which two communicating nodes could be placed from each other while still maintaining reliable data transfer. However, a bug in the Linux driver for the TOSMAC protocol caused corrupt data to be interpreted as correct data.

Overall, the use of a watcher node and the algorithms presented in this paper show promise for providing a reliable method for transferring images from one node to another. With small modifications to the driver, this method would become a viable method of data transfer for future wireless sensory networks.

ACKNOWLEDGMENT

This study was supported by funding from the Industrial University of Ho Chi Minh City, which grants the scientific research activities of lecturers with PhD degrees.

REFERENCES

[1] Kane, Michael B., Courtney Peckens, and Jerome P. Lynch. “Introduction to wireless sensor networks for monitoring applications: principles, design, and selection”. Sensor Technologies for Civil Infrastructures. Woodhead Publishing, 2022. 335-368.

[2] S. Randula, S. Balasooriya and L. Yasakethu, “Ultra-Low Power Wireless Sensor Network for Air Quality Monitoring System”, 2022 International Conference on Green Energy, Computing and Sustainable Technology (GECOST), Miri Sarawak, Malaysia, 2022, pp. 264-268.

[3] S. Frey, S. Vostrikov, L. Benini and A. Cossettini, “WULPUS: a Wearable Ultra Low-Power Ultrasound probe for multi-day monitoring of carotid artery and muscle activity”, 2022 IEEE International Ultrasonics Symposium (IUS), Venice, Italy, 2022, pp. 1-4.

[4] X. Pang et al., “An Ultra-Low Power Time-Domain Temperature Sensor for IoT Applications”, 2022 IEEE 4th International Conference on Circuits and Systems (ICCS), Chengdu, China, 2022, pp. 221-224.

[5] Tayal, Shubham, et al., eds. “Emerging Low-Power Semiconductor Devices: Applications for Future Technology Nodes”. CRC Press, 2022.

[6] Yin, Jun. “Ultra-Low Power Zigbee/BLE Transmitter for IoT Applications”. Selected Topics in Power, RF, and Mixed-Signal ICs. River Publishers, 2022. 315-337.

[7] Crossbow Technology, Inc. “Imote2 Hardware Bundle (WSN-IMOTE2)”, San Jose, California, Document Part Number: 6020-0148-01.

[8] Crossbow Technology, Inc. “Imote2.Builder Kit Manual”, Revision A, September 2007.

[9] COMedia ltd., “C328R User Manual”, June 2007.

[10] Levis, Philip, et al. “TinyOS: An operating system for sensor networks”. Ambient intelligence (2005): 115-148.

[11] Amjad, Muhammad, et al. “TinyOS-new trends, comparative views, and supported sensing applications: A review”. IEEE Sensors Journal 16.9 (2016): 2865-2889.

[12] Kasteleiner, Jorg. “Principles of applying embedded linux on imote2”. Diss. Diploma Thesis, Faculty of Computer Science and Engineering, University of Applied Sciences Frankfurt am Main, 2010.

[13] Chen, Min, et al. “Enabling low bit-rate and reliable video surveillance over practical wireless sensor network”. The Journal of Supercomputing 65 (2013): 287-300.

[14] Vo, Nguyen- Son, et al. “Green two- tiered wireless multimedia sensor systems: an energy, bandwidth, and quality optimisation framework”. IET Communications 10.18 (2016): 2543-2550.

CẢM BIẾN VIDEO THÔNG QUA CHUYỂN TIẾP MẠNG KHÔNG DÂY NĂNG LƯỢNG CỰC THẤP

ONG MẪU DŨNG
Khoa Công nghệ Điện tử, Trường Đại học Công nghiệp Thành phố Hồ Chí Minh
ongmaudung@iuh.edu.vn

Tóm tắt. Các mạng không dây công suất cực thấp có vô số ứng dụng tiềm năng, ngày càng trở nên khả thi khi công nghệ ngày càng nhỏ hơn. Tuy nhiên, trước khi các hệ thống này có thể được sử dụng hết tiềm năng, các phương pháp truyền dữ liệu hiệu quả là bắt buộc để giữ mức tiêu thụ điện năng thấp đồng thời cho phép các thiết bị hoạt động đơn lẻ hoặc theo nhóm. Việc truyền các tệp lớn, chẳng hạn như dữ liệu hình ảnh, là một thách thức đặc biệt do giới hạn kích thước do giao thức MAC 802.15.4. Vận hành các nút cảm biến ở khoảng cách lớn hơn bình thường cũng dẫn đến mất dữ liệu đáng kể. Bài báo này trình bày giải pháp cho vấn đề này bằng cách sử dụng một mạng cảm biến đã được sửa đổi bao gồm các nút dành riêng cho việc lắng nghe để phục hồi gói tin. Thuật toán đề xuất cho phép các nút phát hiện dữ liệu bị thiếu từ một nguồn cũng như cung cấp khả năng yêu cầu dữ liệu bị thiếu đó từ một trong các nút chuyển tiếp chuyên dụng. Thuật toán đề xuất được thử nghiệm với thiết bị iMote2 và cho thấy khoảng cách gần gấp đôi, khoảng cách mà hai nút có thể được đặt với nhau trong khi vẫn duy trì truyền dữ liệu đáng tin cậy. Thông qua thử nghiệm, kết luận rằng trình điều khiển Linux chịu trách nhiệm chuyển các CRC từ lớp 802.15.4 sang lớp MAC là không đúng. Do đó, không thể lọc dữ liệu bị hỏng ở lớp ứng dụng.
Từ khóa. Thiết bị không dây công suất cực thấp, cảm biến video, chuyển tiếp mạng không dây, tinyOS.

Received on: 29/01/2023

Accepted on: 25/05/2023