Fault tolerance and Distance Reliability of Electric Imp 802.11b/g/n Wi-Fi transceiver

Charan Jagaboranahalli Puttaswamy Gowda
MSCS, Department of CECS
California State University, Long Beach
Long Beach, CA, United States
Charanjagaboranahal.Puttaswamygowda@student.csulb.edu

Varun Krishna
MSCS, Department of CECS
California State University, Long Beach
Long Beach, CA, United States
Varun.Krishna@student.csulb.edu

***Abstract*—The Internet of Things (IoT) [1] [9] are connection of uniquely identifiable embedded devices to the Internet that proved services beyond simple communication. It has evolved to allow automation in nearly all fields which pave way to smart devices and infrastructure like pacemaker and smart grid respectively[4][8]. Critical systems like pacemaker or weather monitoring systems need highly reliable data communication to transmit data. In this project we establish a wireless connection using the Electric IMP Wi-Fi module between the embedded system and internet cloud server to test and measure the distance reliability of the connection by transmitting data over several distances with implementing fault tolerant methods. The number of packets transmitted and dropped between sender and receiver are tracked before and after implementation of Double Precision checksum, Berger Code and Byzantine Generals algorithm [3]. The correctness of data transmitted and the transmission range from sender to receiver is increased using data redundancy techniques while decreasing the faults in the system.**

***Index Terms*—Byzantine, reliability, error detection, Electric IMP, wireless communication, distance reliability, Arduino microcontroller, double precision, checksum, plot.ly.**

I. Introduction

Ensuring the high reliability and performance with low fault is the major criteria to guarantee successful data transfer through a wireless network. Wireless communication allows Internet of Things to be more mobile, dynamic, and efficiently [10]. Using Wi-Fi allows data to be transmitted at higher data rates with low energy consumption as well as low installation cost [2]. This project demonstrates the creation of black box data pipeline which transfer data reliably between Arduino Uno Microcontroller embedded system and cloud server plot.ly that record received sensor readings and plots a graph on a web user interface that was transmitted through Electric IMP Wi-Fi module. The project also demonstrates how reliable the data transferred when the distance between the Wi-Fi module and the router is varied. The user interface generates a graph that display the loss of data when there is no fault tolerance method implemented versus the graph obtained after the implementation of different error detection and fault tolerance methods like double precision checksum[5], Berger code[6], Byzantine fault tolerance [3][7] and combination of these methods between the sender and receiver.

II. Purpose

The project is developed with error detection and fault tolerant techniques, specifically information redundancy to improve wireless data transmission. We will demonstrate the distance reliability of Electric Imp Wi-Fi module by implementing Double-Precision Checksum, Berger Code error detection methods, Byzantine Generals Algorithm and cascaded system which is the combination of first two error detection methods with Byzantine Generals Algorithm and compare the results to see

which methods decrease arbitrary faults while increasing data consistency in wireless data transfer.

III. Claims

We increase the data consistency by decreasing arbitrary faults with the help of double precision checksum and Berge Code error detection algorithm. Also, we implement byzantine general's algorithm [7] to increase the distance reliability with the help of data redundancy. Then go on to show that cascading the error detection algorithms with the fault tolerance algorithm will result in better fault tolerant system by comparing the results.

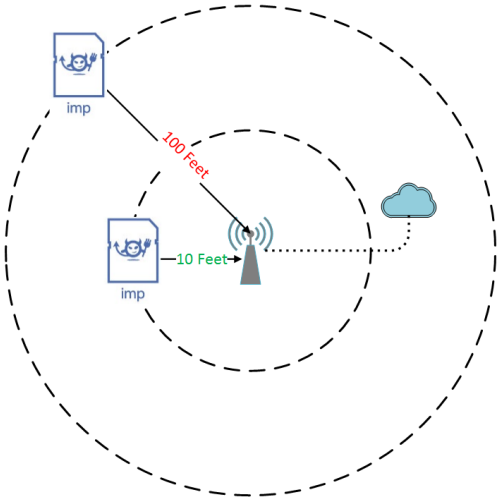

Figure. 1. Electric IMP placed at varying distance

IV. Hardware Setup

- Arduino Uno Microcontroller Board.
- April Electric IMP breakout board.
- Arduino SD card Shield.
- 2GB micro-SD card.
- Electric Imp 802.11b/g/n Wi-Fi transceiver.
  - Power Output: 16.75 dBm
  - Communication Protocol: IEEE 802.11b/g/n
  - Operating Frequency: 2.4GHz
  - Serial Communication
  - TCP Buffering
- NETGEAR Wireless Router - N600 Dual Band Gigabit (WNDR3700)
- USB cable
- Micro USB cable
- Personal Computer:
- OS: Windows 8 64 bit
- Chrome/Firefox Internet Browser

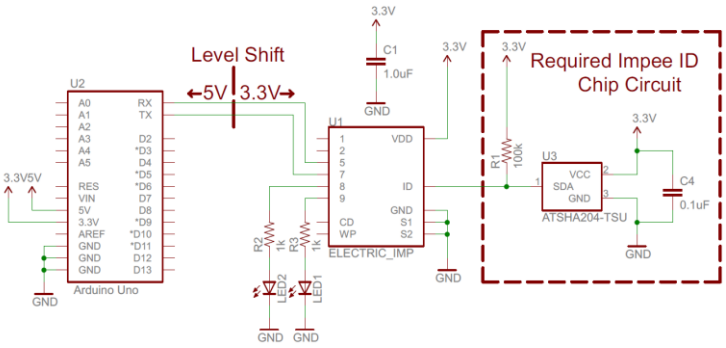

Figure. 2. Electric Imp – Adriano Schematic Diagram

The temperature sensor is connected to Arduino Uno board on an analog pin A0 with a pull down resister 1K Ohm, Analog signal on A0 Pin is converted to digital using Analog to Digital Converter (ADC). The Arduino Uno Board is connected to SD Card Shield through a digital pin D10 and programmed to poll temperature sensor data from ADC every second and log to SD card. Arduino Uno board is also connected to Electric IMP in serial through RX, TX ports. Electric IMP is programmed to receive serial data that contains Unix TimeStamp, temperature value and checksum, the data is then retransmitted to cloud agents using TCP buffering. Figure. 2 shows a diagram of the connections.

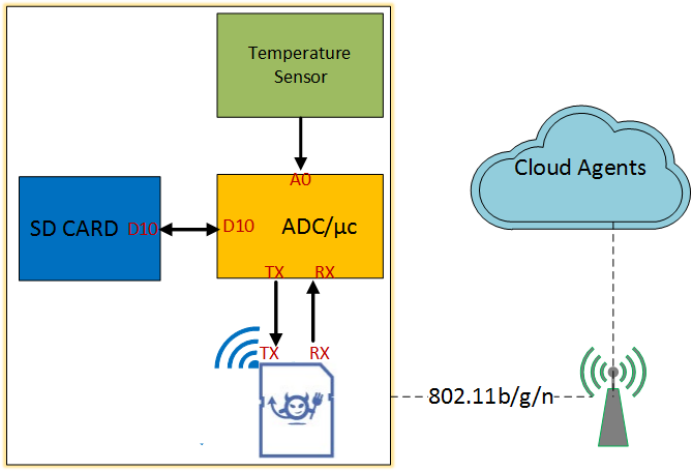

Figure. 3. Wireless communication setup

V. SOFTWARE

Arduino IDE was used to program the Arduino Uno Board. The serial communication protocol was set to: 19200 Baud, No Parity, 8 Data Bits, and 1 stop bit. Arduino serial monitor was then used to see the temperature data logged to SD card shield and Electric IMP through RX, TX ports to form serial communication. Then the electric IMP is programmed in squirrel language (Figure. 4) to pole for serial buffer full and when the complete packet arrives at the Electric IMP i.e. UNIX timestamp followed by the sensor reading and checksum, it was sent to multiple agents on the cloud using TCP buffering. After this milestone each agent receiving the data was programmed to check for data error using two error detection algorithms i.e. Double Precision Checksum and Berger Code algorithm. The error detection rate are recorded and if there is an error in the received data then a retransmission is requested. After retransmission of data second time error detection is done and error date are recorded. Then the Byzantine Generals Algorithm implemented to recover lost or dropped packets using majority voting with the help of Interactive Consistency vector [3]. The winner results are programed to be sent to cloud based plot.ly data analysis and visualization tool for visual representation. The data recovery rate is also recorded. Then finally a two cascaded system are implemented by combing and implementing Double Precession Checksum with Byzantine Generals and Berger Code with Byzantine Generals. The results are about data recovery and documented by logging the data to cloud server and Plot.ly application.

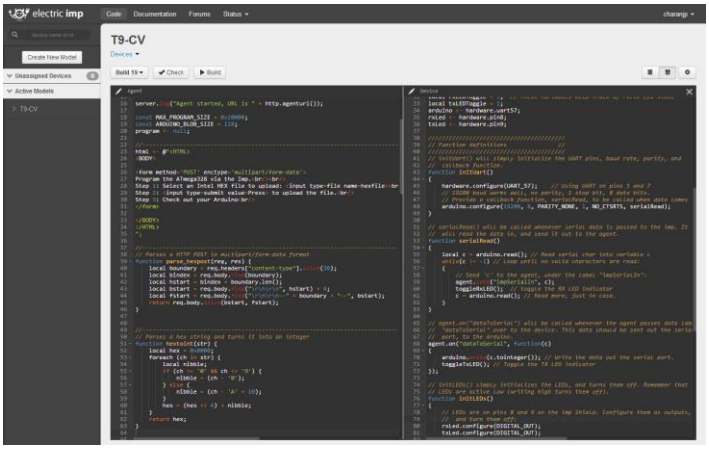

Figure. 4. Device and Agent IDE

Plot.ly provides the interface that allows the user to do the following:

- Visually represent the data transmitted and
- Download the received data as a csv file for
    - No algorithm
    - Double Precision Checksum[5]
    - Berger Code[6]
    - Byzantine Generals algorithm [3]
    - Double Precision Checksum cascaded with Byzantine Generals Algorithm
    - Berger Code cascaded with Byzantine Generals Algorithm

The User Interface (Figure. 5) shows the visual representation of the graph plotted for received data i.e. time vs temperature for the Double Precision Checksum cascaded with Byzantine Generals Algorithm.

Figure. 5. Receiver User interface

VI. DATA USED

The data that we decide to use is the temperature readings (measured in Fahrenheit) acquired with the help of a temperature sensor over a short period of time (measured in milliseconds). We observed that the temperature obtained for such a short time at a constant place would result in a linear graph and it would be difficult to identify change or loss in packets visually. We varied the temperature by changing the position of the sensor from room temperature to a colder region and increasing it with the help of an external heat source. This resulted in a graph plotted in Figure. 6. Any data that we want to

transfer would work. Also a subset of data with varying distance is obtained.

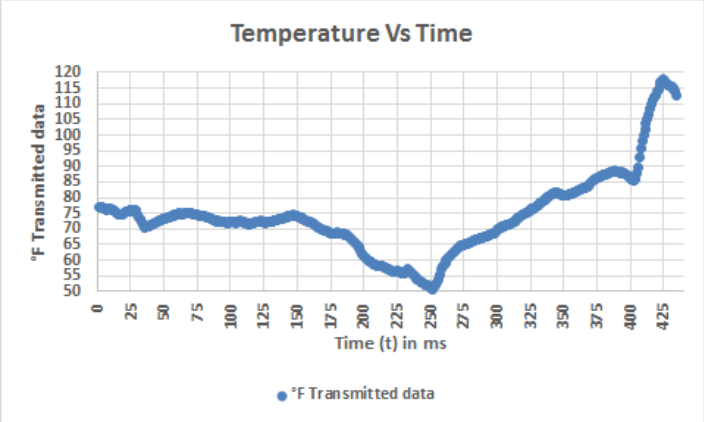

Figure. 6. Source Date transmitted: Temperature (°Fahrenheit) vs Time (millisecond)

VII. Implemented Algorithms

Squirrel is a high level imperative, object-oriented programming language and hence binary data manipulation of the implemented algorithms is difficult. To solve this problem, the data uses mainly the concepts of the algorithm. Below is the list of algorithms that we implemented:

A. Model 1 - None:

No algorithm is used. Data is being sent whether the receiver receives the data or not.

B. Model 2 - Double-Precision Checksum:

The Checksum [5] is calculated by adding up the block of data that is being transmitted and transmits this sum with the data as well. The receiver then adds up the data it received and compares this sum with the checksum it received. If the two do not match, an error is indicated. You can see an example of calculating checksum in Table.1.

Table. 1. Checksum for Source Data

Sl No.	Data Packet		
	Data X	***Data Y***	***Checksum***
	Unix TimeStamp	Temperature	Checksum
Example:	1416723273	76.9994	1416723349.9994

I. Algorithm to Calculate and Transmit Checksum:

The transmitting data contains Unix Timestamp, Temperature and the Transmission checksum as follows:
1416723273 + 76.9994 = 1416723349.9994
TransChecksum = 1416723349.9994
Table. 1: shows the tabular form of the transmitted data. The transmitted data will be sent with commas separated as follows:
1416723273, 76.9994, 1416723349.9994.

Pseudo Code (to calculate transmission checksum):
Start
Initialize TransChecksum to zero;
TransChecksum= Unix TimeStamp + Temperature;
Send Unix Timestamp, Temperature,
TransChecksum;
End;

II. Algorithm to check the correctness of the Received Data through Checksum:

The received data is identical to the transmitted data if there is no errors in the received data. To check the correctness of the data received we retrieve the transmitted checksum, then sum up the Unix Timestamp with the temperature to calculate the CalChecksum. The transmitted checksum is compared with the calculated checksum, if both the checksum are equal then there is no errors in the received data.

Table. 1: shows the received data
1416723273, 76.9994, 1416723349.9994.
TransChecksum = 1416723349.9994;
1416723273+76.9994 = 1416723349.9994
CalChecksum = 1416723349.9994

If the TransChecksum = CalChecksum (1416723349.9994 =1416723349.9994) then no error is present and if they are not equal then an error was introduced while transmission. So a request to retransmission is sent. If the retransmission also fails then a default value is set i.e. zero and this result is fed into the next stage of fault tolerance system i.e. Byzantine Generals Algorithm.

Pseudo Code (to check the correctness of the data received):
Start
Separate incoming data;
Get TransChecksum;
Initialize CalChecksum and ReqRetransmit to zero;
CalChecksum = Unix TimeStamp + Temperature;
If: CalChecksum == TransChecksum and ReqRetransmit ==0;
    Store Data;
Else If: CalChecksum! = TransChecksum and ReqRetransmit = =0;
    Request for retransmission;
    Set ReqRetransmit to 1;
Else If: CalChecksum != TransChecksum and ReqRetransmit ==1;
    Store zero as Data;

C. Model 3: Berger Code

Berger Code is a unidirectional error detection code. Unidirectional errors are errors which occurs when binary ones are changed to zeroes or zeroes changed to ones. To check if an error has occurred Berger codes is used and the bits of Berger codes are computed by summing all the zeroes in the word and express that sum in natural binary. Berger Code needs **k = [log$_2$ (n+1)]** check bits for an information word consisting of **"n"** bits [6]. Therefore the length of the Berger Code is **"k+n"** bits i.e. **"k"** check bits are required to check **n=2^k-1** information bits [6]. Berger code can detect any number of changes in the information word as long as the changed one-to-zero or zero-to-one errors are not equal and the Berger code can just detect error has occurred or not, it cannot correct any errors.

TABLE. 2. BERGER CODE FOR SOURCE DATA

Sl No	Data Packet		
	Data X	***Data Y***	***Berger Code***
	Unix TimeStamp	Temperature	Berger Code:
Ex	1416723273	76.9994	-17

I. Algorithm to Calculate and Transmit Berger code:

The transmitting data contains Unix Timestamp, Temperature and the transmitted Berger Code as follows:

1416723273 + 76.9994 = 1416723349.9994
1416723349.9994_{10} = $10101000111000101111011100101012$
Number of 1's = 17 and 1 complement of 17 = -17
TransBergerCode = -17

Table. 2: shows the tabular form of the transmitted data. The transmitted data will be sent with commas separated as follows:
1416723273, 76.9994, 1416723349.9994, -17

Pseudo Code (to calculate transmission checksum):
Start
Initialize TransBergerCode to zero;
Convert data to binary;
Count the 1's present in to binary word;
Complement the count;
TransBergerCode = Complement of the count;
Append the TransBergerCode to value;
Return TransBergerCode;
End;

II. Algorithm to check the correctness of the Received Data through Berger Code:

The received data is identical to the transmitted data if there is no errors in the received data. To check the correctness of the data received we retrieve the transmitted encoded Berger Code and compare with the new calculated Berger Code, if both the Berger Code are equal then there is no errors in the received data. If there is an error in the code then a retransmission is requested.

Received data from Table 2:
1416723273, 76.9994, 1416723313.9994,-17

Calculate Berger Code:
1416723273 + 76.9994 = 1416723313.9994
1416723313_{10} = $10101000111000101111011100101012$
Seventeen 1's = 17
Complements 17 = -17
CalcBergerCode = -17
If the TransBergerCode = CalcBergerCode (-17 = -17), then we have no error. If they are not equal then we have an error and a request for retransmission is sent.

Pseudo Code:
Start
Separate incoming data;
Retrieve TransBergerCode;
Initialize CalcBergerCode and ReqRetransmit to zero;
Convert data to binary;
Count the number of 1's;
Complement the count;
Return CalcBergerCode;
If: TransBergerCode == CalcBergerCode and ReqRetransmit == 0;
    Return true;
Else If: TransBergerCode != CalcBergerCode and ReqRetransmit == 0;
    Request for retransmission;
    Set ReqRetransmit to 1;
Else If: CalcBergerCode != TransBergerCode and ReqRetransmit ==1;
    Store zero as Data;
End;

D. Model 4 - Byzantine Generals Algorithm:

When processes suffer from arbitrary faults and produce arbitrary outputs. Such failures are known as byzantine failures. We extend this to our experiment when arbitrary packets are omitted upon or lost during transmission.

Initial Condition: Assuming 4(N) units where 4(N) >3m+1; up to m faulty. This has one source which is the Electric IMP and 3(N-1) agent receivers. Data consistency is obtained only if all the above condition holds good.

The algorithm Byzantine (N,m) has three steps:
Step 1: Source sends information to each of the 3 (N-1) agent receivers.
Step 2: All the 3(N-1) agent receivers exchange the information received from Source i.e. other 2 (N-2) receivers. If there is no message from another unit then add default value (Zero) into its records.
Step 3: Each agent receiver now has an Interactive Consistency Vector (ICV) of m values. It uses majority after voting or the default value if no majority exists [7].

Once the data packet is set to transmit, it is sent to three agents serves in the cloud. Each agent then interacts with the other two in order to check for the correct data. If the consistency vector matches with even one of the other two agents then we verify that it is the correct set of data being transmitted and store it, else default value is stored.

E. Model 5 – Double-Precision Checksum Cascaded with Byzantine Generals Algorithm

In this model two model are cascaded that is Double-Precision Checksum algorithm along with Byzantine Generals Algorithm. Here three agents are setup as described in model 4. But the difference is before a value is store using a voter winning system into interactive consistency vector, Double Precision Checksum algorithm is implemented to check for errors in the data.

Here in Model 5 on all the received values at three agents, Model 2 i.e. double precision checksum algorithm is implemented to check the correctness. If the received data is correct then it is stored and if the received packet contains error then a retransmission is requested. If the retransmitted packet is lost then the value is stored as zero. But if the retransmitted data contains error then the error value is stored and passed to the next stage so that Byzantine Generals algorithm can handle it as show in Model 4. Cascading of double precision checksum with Byzantine Generals Algorithm helps us achieve increased error correction and fault tolerance through redundancy giving greater results.

F. Model 6 – Berger Code Cascaded with Byzantine Generals Algorithm

In this model two model are cascaded that is Berger Code algorithm along with Byzantine Generals Algorithm. Here three agents are setup as described in model 4. But the difference is before a value is store using a voter winning system into interactive consistency vector, Berger Code algorithm is implemented to check for errors in the data.

Here in Model 6 on all the received values at all three agents, Model 3 i.e. Berger Code algorithm is implemented to check the correctness. If the received data is correct then it is stored and if the received packet contains error then a retransmission is requested. If the retransmitted packet is lost then

the value is stored as zero. But if the retransmitted data contains error then the error value is stored and passed to the next stage so that Byzantine Generals algorithm can handle it as show in Model 4. Cascading of Berger Code with Byzantine Generals Algorithm helps us achieve increased error correction and fault tolerance through redundancy giving greater results than that of Model 5 with less computation and overhead.

VIII. Results and Analysis of Data

All the results obtained by the experiment were performed in a controlled environment using a NETGEAR Wireless Router - N600 Dual Band Gigabit (WNDR3700). We have varied the position of the Electric Imp with the router placed at the center and increasing the distance from 10 feet to a 100 feet as shown in Figure. 1 in the increments of 10 feet. During the testing the router used was a medium range router and experiment was conducted up to 100 feet from the router after which the Wi-Fi signal was lost and all packets were dropped during the transmission. Table 3 and 4 shows the data transmission achieved by implementing different algorithms models. Figures. 7 and 8 shows the test results graphically obtained at 60 feet.

Table. 3. Data Transmission Percentages using Model 1, 2, 3 and 4

Percentage of data Transferred				
Distance (feet)	After Implementing			
	No Algorithm (Model 1)	Double-Precision Checksum (Model 2)	Berger Code (Model 3)	Byzantine Generals (Model 4)
10	42.29	52.61	59.65	100
20	43.67	54.41	58.32	100
30	37.70	46.55	53.21	100
40	17.24	19.69	30.39	100
50	25.74	30.66	41.05	100
60	16.32	18.54	29.11	96.90
80	24.13	28.54	39.18	97.54
100	0	0	0	94.28

Table. 4. Data Transmission Percentages using Model 1, 5 and 6

Percentage of data Transferred			
Distance (feet)	No Algorithm	After Implementing Model 5	After Implementing Model 6
10	42.29	100	100
20	43.67	100	100
30	37.70	100	100
40	17.24	100	100
50	25.74	100	100
60	16.32	97.70	97.751
80	24.13	99.54	99.542
100	0	100	100

At 60 feet, we can observe as show in Figure. 7 that the data transmitted without the implementation of the algorithm is not complete and there is a tremendous loss of packets during transmission. Whereas in Figure. 8 we can see the graph is complete which is identical to the source data shown in Figure. 6 wherein all the packets are transmitted, this is achieved after implementing of Berger Code cascaded with Byzantine Generals Algorithms [3].

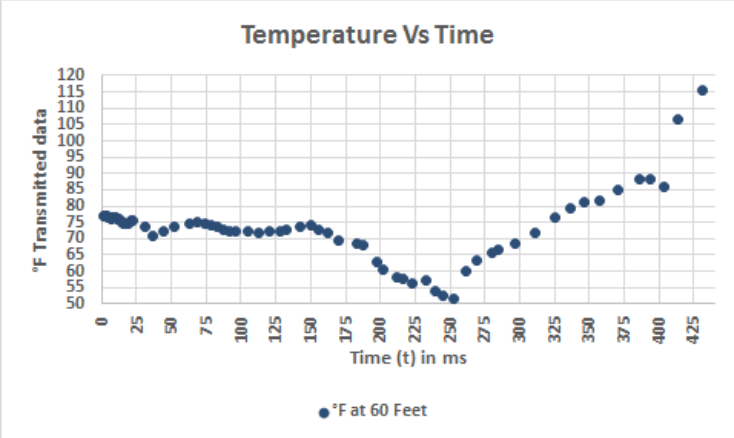

Figure. 7. Graph of Data Transfer with no Algorithm at 60 feet

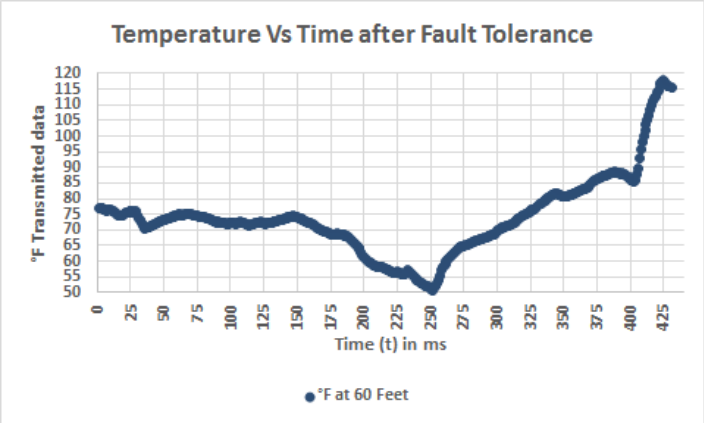

Figure. 8. Graph of Data Transfer with Fault Tolerant methods at 60 feet

TABLE. 5. RATE OF DATA TRANSMISSION ACHIEVED USING DIFFERENT MODELS

Model 1: No Algorithm	25.88
Model 2: Double Precision Checksum	31.37
Model 3: Berger Code	38.86
Model 4: Byzantine Generals Algorithm	87.63
Model 5: Double Precision Checksum cascaded with Byzantine Generals algorithm	99.65
Model 6: Berger Code cascaded with Byzantine Generals algorithm	99.66

Table 5 shows the rate of data transmission achieved by implementing different algorithms models. When we used Model 1 we achieved a data transmission of about 25.88%. This was improved when we implemented the Model 2 by 5.49% to 31.37%. But Model 3 gave a better result i.e. the data received increased by 12.98%. We can see a drastic improvement after implementing Model 4 using Byzantine Generals Algorithm where the rate of data transmission reached 87.63%. But initially we claimed that cascading the models will yield a better rate of data transmission. So, we tested out this theory and found out it is indeed true, by cascading Double-Precision Checksum and Byzantine Generals algorithms we were able to get 100% of data transmission rate (in Table. 4) up to 50 feet. After 50 feet we were able to achieve on an average 99.65% and 99.66% data transmission with Model 5 and 6 respectively. If no algorithm is implemented there is no guarantee that 100% of the sent data is being received throughout the various distances. But with the implementation of Model 5 and 6 we can on an average 99.65% and 99.66% data transmission rate respectively with no error and are fault tolerant.

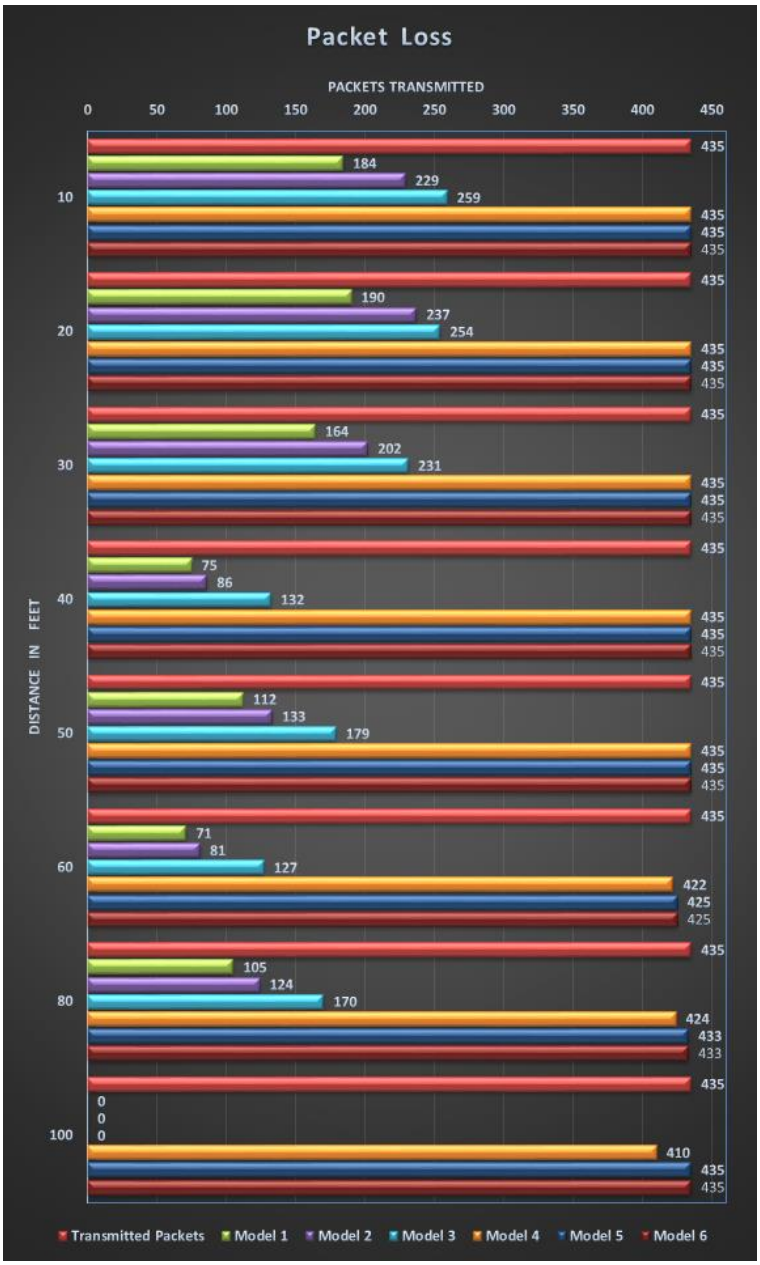

Figure. 9. Graph of Packets Transfer with different Fault Tolerant methods at varying distance

In terms of packet loss, the results in Figure. 9 depicted by bar graphs provide an overview of the efficiency of the algorithm we have used. We can observe that the average percentage of data transmitted of all the distances before implementing the algorithm is 25.88% and the percentage of data

transmitted after implementing the algorithms is a staggering 99.65% and 99.66% i.e.We were able to transfer nearly all of the data. Thus, with this experiment we were able to test and verify the distance reliability and fault tolerance of the Electric Imp Wi-Fi transceiver.

IX. Challenges

The biggest challenges we faced was the use of new technology, we had to learn the working and coding of Electric Imp and Arduino microcontroller. We use squirrel language for programming both on the device and the agent. For the purpose of establishing a perceivable output we had to learn to write subroutines in Microsoft excel. The physical challenge was conducting the experiment in an outdoor environment; we had to wait for a few 7minutes at every attempt of transmitting data at distances from 10 feet to 100 feet.

X. Future Improvements

We would like to extend the project for future improvements by implementing a fully functional real time device which works with Electric IMP. Some of the prototypes we are considering are pacemakers and weather systems and we are also considering to improve the error correction algorithms to improve the reliability of data transfer, which would be an effective way to decrease faulty data transfer in addition to observing the reliability of the current technology available.

XI. Conclusion

In conclusion, this experiment demonstrated that as distance increases between the Electric IMP and the router, the consistency in the data being transferred decrease by lost of packets or by induction of errors during transmission. With the help of double precision checksum, Berger Code and Byzantine fault tolerant algorithms we are able to substantially increase data consistency over the same distances. Where the double precision checksums and Berger Code algorithm makes sure the data being transmitted is correct and decease the data loss be requiting for a retransmission of faulty data. Byzantine generals algorithm decreases packet loss by introducing data redundancy and decreasing error rate by voter winning system. And finally the experiment validates that by cascading two system as shown in Model 5 and 6 will yield greater results. Even though the Double-Precision Checksum with Byzantine generals Algorithm yields a good result, we can conclude that Model 6 i.e. Berger Code cascaded with Byzantine general's algorithm is the better fault tolerant model and has more reliable error detection technique which is much needed for implementing fault tolerance methods on a wireless network.

References

[1] Shafee, Ahmed El, and Karim Alaa Hamed. "Towards a Wi-Fi Based Home Automation System." International Journal of Computer and Electrical Engineering, 4.6 (2012): 803.

[2] Zhang, Xinyu, and Kang Shin. "Enabling Coexistence of Heterogeneous Wireless Systems: Case for ZigBee and Wi-Fi." Proceedings of the Twelfth ACM International Symposium on Mobile Ad Hoc Networking and Computing, (2011): 1-11.

[3] Veronese, G., Correia, M., Bessani, A., Lung, L., & Verissimo, P. (2013). "Efficient byzantine fault-tolerance. IEEE Transactions on Computers", 62(1), 16-30.

[4] Wang, W., Xu, Y., & Khanna, M. (2011). "A survey on the communication architectures in smart grid. Computer Networks", 55(15), 3604-3629.

[5] Saxena, N. , & McCluskey, E. (1990). Analysis of checksums, extended-precision checksums, and cyclic redundancy checks. IEEE Transactions on Computers, 39(7), 969-975.

[6] Xu, B., Zhao, C. , Hu, E. , & Hu, B. (2011). Job scheduling algorithm based on berger model in cloud environment. Advances in Engineering Software, 42(7), 419-425.

[7] Veronese, G. , Correia, M. , Bessani, A. , Lung, L. , & Verissimo, P. (2013). Efficient byzantine fault-tolerance. IEEE Transactions on Computers, 62(1), 16-30.

[8] Suryadevara, N. , Mukhopadhyay, S. , Kelly, S. , & Gill, S. (2015). Wsn-based smart sensors and actuator for power management in intelligent buildings. IEEE/ASME Transactions on Mechatronics, 20(2), 564-571.

[9] Su, P. , Shih, C. , Hsu, J. , Lin, K. , & Wang, Y. (2014). Decentralized fault tolerance mechanism for intelligent iot/m2m middleware. 2014 IEEE World Forum on Internet of Things (WF-IoT), 45-50.

[10] Wang, C. , Jiang, C. , Liu, Y. , Li, X. , & Tang, S. (2014). Aggregation capacity of wireless sensor networks: Extended network case. IEEE Transactions on Computers, 63(6), 1351-1364.