**About us**

Eliab Woldeyes is in Computer Systems Engineering which gives a technical background on system design from a hardware and software point of view. Software design techniques will be used in building the device firmware while skills learned in real-time development of microcontrollers will be used in building the device itself.

Jonathan Chan is in Computer Systems engineering, throughout this degree the focus has been on software design with a sprinkling of computer hardware.

Denis Chupin is a 4th year student in Computer System Engineering with an experience of Software application tester. Testing experience will be implemented to make sure that system is working as it is intended to be.

**The Team Project**



LED’s are one of the key components in almost any electronic device. They are used in numerous different ways but the common baseline of all these ways to use an LED is either to display information to a user or to simply function as a light. Our project aims to combine these two conventional uses of an LED to create an innovative way to light a room with data provided by a user.

The project that is being proposed is the “RGB LED globe,” also called a POV globe (Persistence of Vision Globe). The POV optical illusion is when a visual representation of an object is still present after the light source projecting that object has stopped. In this case this effect will reply on the timing of the LEDs and the rotation speed to have the spinning circle lined with LEDs to present a 3D image.

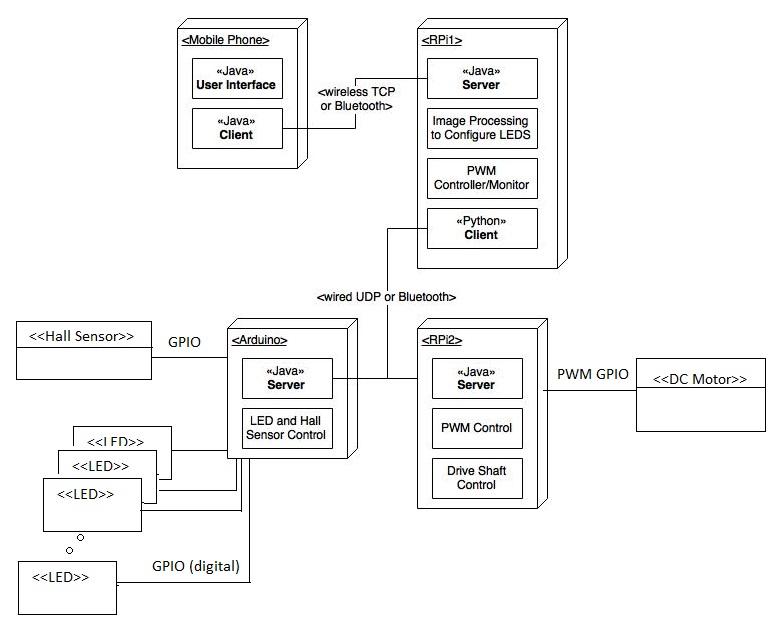
There is no real world problem that this project is taking on, this project is just to demonstrate that this effect can be created with the LED globe.

The functional requirements are:

* To construct appropriate housing for the motor driving the shaft supporting and turning the LED ring.
* Control the motor speed to be able to simulate the optical illusion, reach an appropriate number of frames per second so the image looks complete. A raspberry pi PWM output could be used to drive the motor or use an external PWM controller.
* A separate Raspberry pi can display the interface to select an image to send to the globe. If an Ethernet cable is required to send the image, the procedure would be to connect, send it, and then disconnect it.
* The second Raspberry pi could also be used to send the take a user’s desired speed and send it to the raspberry pi controlling the motor, then poll for new user input speed.
* Set the correct timing of the LEDs to change colour or blink when spinning to present the correct image on the globe. An Arduino can be used to control the LEDs.
* Using a Hall sensor to detect when one revolution is complete, this helps with synchronization for the pattern to be sent to the LEDs. This may be controlled by the same Arduino.
* To convert images to be able to be displayed on the globe, the image has to be converted into a pixelated version then determine the (r,g,b) array for each pixel. The image processing can be automated with Python then the arrays can be directly copied to Arduino Sketch.
* The mobile application should provide a user-friendly interface to control the globe, allowing seamless connection to the system. If time permits it, the application should also be able to send a photo to display on the globe.

**The Team Project Design**

**Initial Systems Architecture**



**Design – Communication Protocols**

The protocol that will be used in this project is a modified Trivial File Transfer Protocol, TFTP, with multi-threaded Client and Server. The most important component of this protocol is the procedure to track packets and handling of errors.

In this project the write and read job request can be used to write a file that contains the bit pattern of an image to the Arduino. Then send an action job request to initialize the program that controls the LED ring with a specified bit pattern file.

Communication with the DC motor starts with a handshake, then a user entered value (may be RPM) is sent as a job request to operate the motor at that value. The thread remains polling if the user decides to alter the value.

Any transfer begins with Read, Write, or Job request, essentially a request to establish a connection. Data packet is a fixed length is 512 bytes, this allows the receiver to realize that a packet is the final packet when it is less than 512 bytes. Every Data packet received must be replied by an Acknowledgement and the order of packets is kept track with a block numbers, where the ACK echos the block number of the Data packet being acknowledged.

**Error codes**

**Error code 1 - File Not Found**

WRQ: The Client does not make a request packet if the ﬁle does not exist. It displays the error and asks the user for the ﬁle-name again.

RRQ: The Client sends a read request packet to Server and if the Server cannot ﬁnd the ﬁle, it sends an error packet (error code 1) and the transfer stops.

**Error code 2 - Access Violation**

WRQ: The Client sends a write request and if the Server cannot access the ﬁle, it will send an error packet (error code 2) back and the transfer will stop.

RRQ: The Client sends a read request and then if it cannot access the ﬁle, it displays ”ACCESS VIOLATION” and quits the transfer. The Server times out since it does not receive any packets.

One way to simulate this is when the client asks for the ﬁle to write to ( the second ﬁlename), enter a path to a read-only folder. When this is done the Server(for WRQ) or the Client(for RRQ) won’t be able to write to it, causing an ACCESS VIOLATION error. For a visual representation of this error see ﬁgure 1.

**Error code 3 - Disk Full**

WRQ: The transfer goes on as normal until the Server cannot write because the disk got full. It sends an error packet (error code 3) to the Client and the transfer stops.

RRQ: The transfer goes on as normal until the Client cannot write because the disk got full. The Client prints an error message saying the disk is full and quits the transfer. The Server times out and quits on its end too. One way to simulate this is - to write to a USB that has less space than the ﬁle being written. This will cause an error when the disk gets full mid-transfer. For a visual representation of this error see ﬁgure 2.

**Error code 4 - Illegal TFTP Operation**

**Invalid TFTP Operation**

WRQ, RRQ, and JRQ: When the server receives a packet, either a request packet, data packet or ACK (Acknowledge) in which the op-code does not correspond to the current TFTP operation, the server will send an error packet containing details about the operation and the transfer will end.

**Incorrect Block Number**

WRQ, RRQ, and JRQ: When the client receives an ACK or DATA packet with a block number less than what is expected the packet is ignored and the previous packet is re-transmitted. When the server receives a DATA packet with a block **number less than** what is expected the server sends a error packet to the client and the transfer is shut-down. When the server receives an ACK packet with a block number less that what is expected the packet is ignored and the server re-transmits the previous packet. For a visual representation of this error see ﬁgure 4.

If the client or server receives an ACK or DATA packet with a block **number greater than** what is expected the server sends an error packet containing details about the operation and the client then shuts down the transfer.

**Missing File Name**

WRQ, RRQ: When the server receives a request packet with no ﬁle-name it sends and error packet to the client and the transfer stops. Below is a Timing diagram representing what happens in the event the ﬁlename is missing. For a visual representation of this error see ﬁgure 5.

**Incorrect Request Packet Format**

WRQ, RRQ, and JRQ: When the server receives a request packet with missing zero’s, invalid packet size, or incorrect mode the server will send an error packet to the client and the transfer will stop. For a visual representation of this error see ﬁgure 6.

**Invalid Mode**

Figure 6.1

**Error code 5 - Unknown Transfer ID**

WRQ, RRQ, and JRQ: When an unknown TID is received the recipient will send an error packet and the transfer will stop. The sender’s socket will then close. See ﬁgure 7.

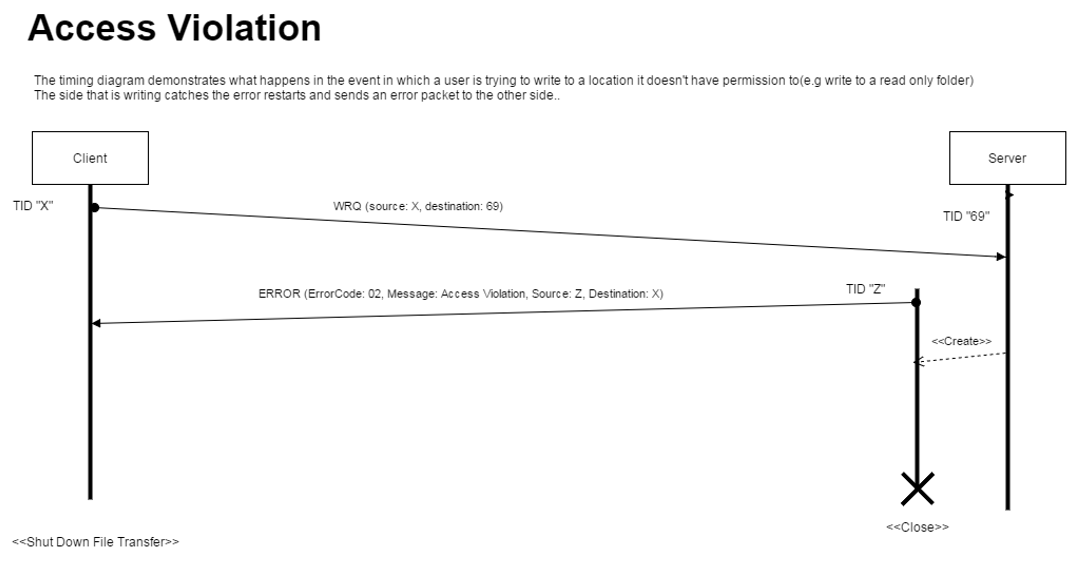
**Error code 6 - File Already Exist**

WRQ: The Client sends a write request and once the Server sees that the ﬁle it is supposed to write to already exists, it sends an error packet (error code 6) and the the transfer stops. See ﬁgure 8.

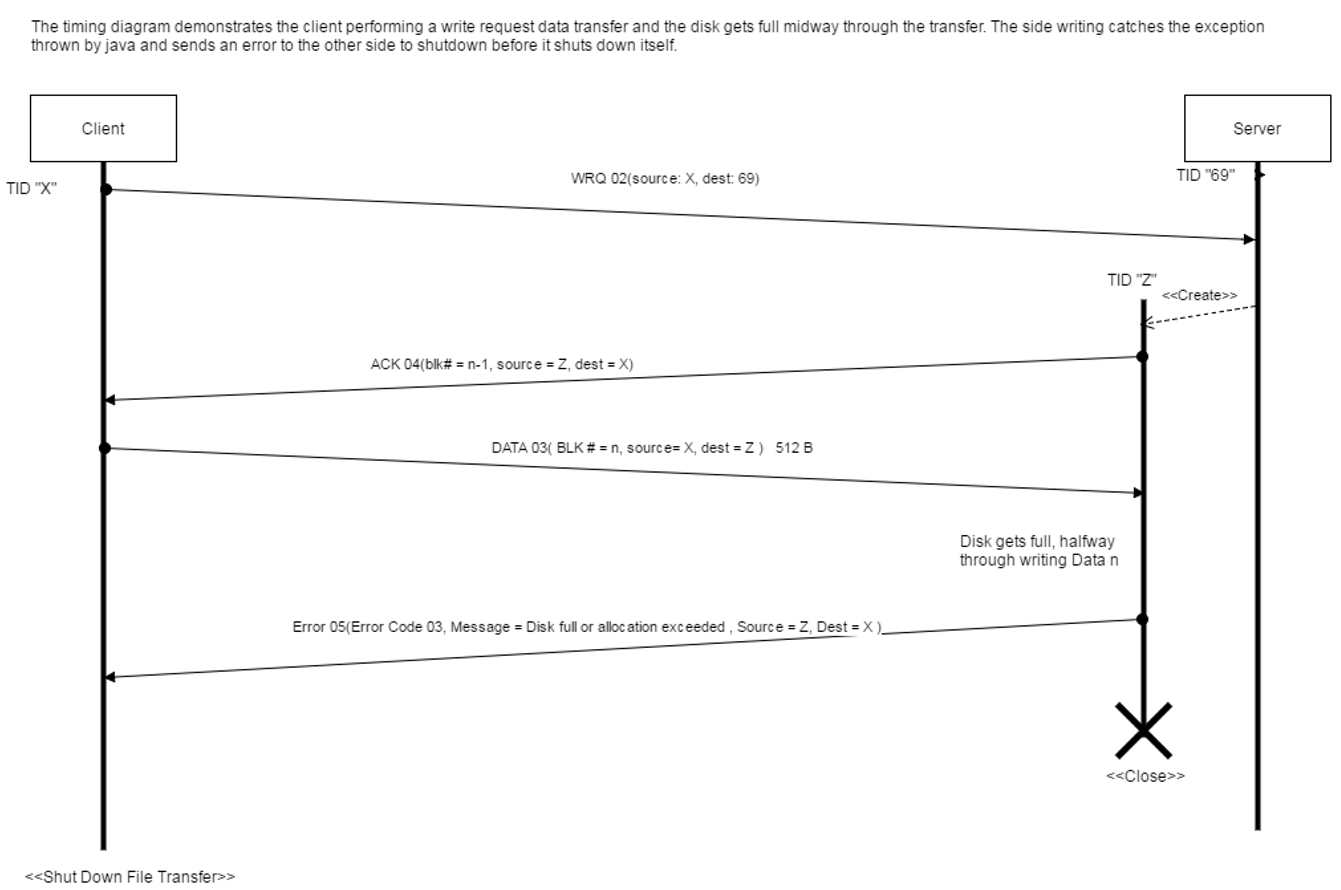
RRQ: If the ﬁle the Client is supposed to write to exists, it simply overwrites it. So the transfer continues at normal.

**Timing Diagrams for error codes**

**Figure 1: Access Violation**

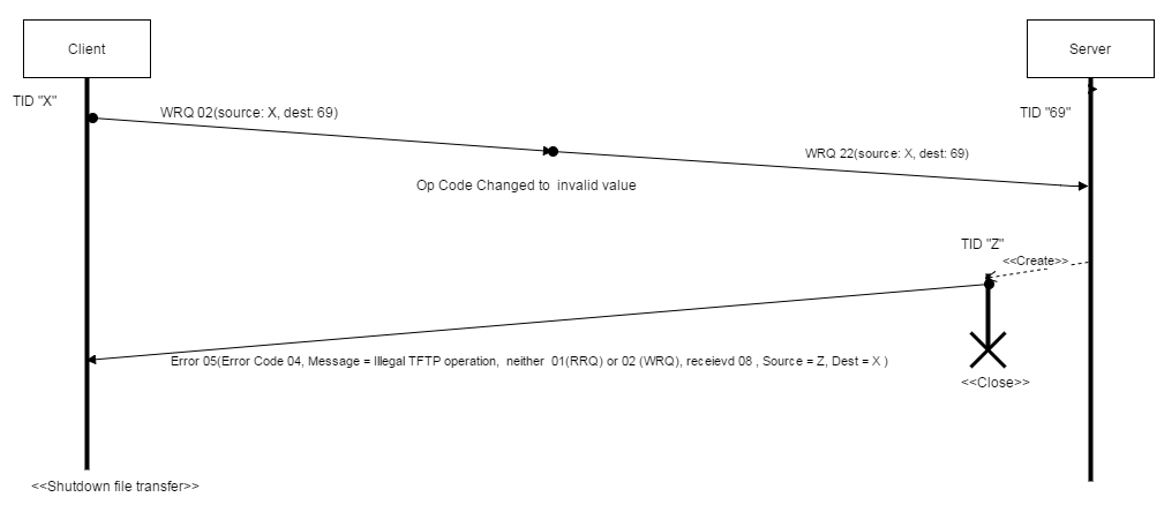


**Figure 2: Disk Full**

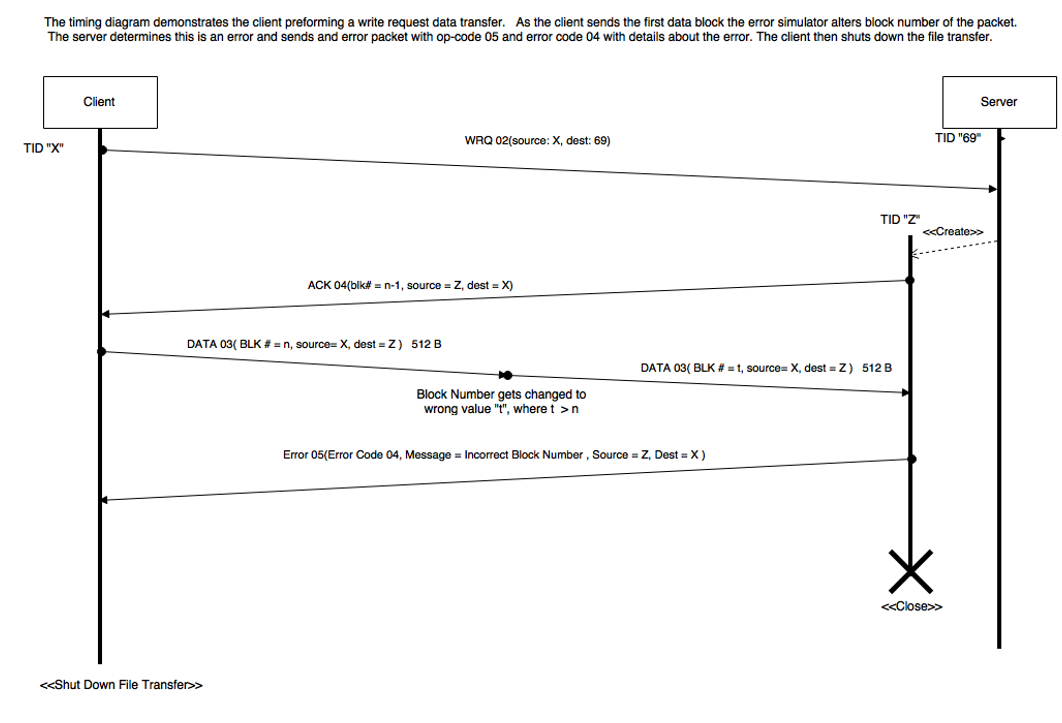
****

**Figure 3: Incorrect Op code**

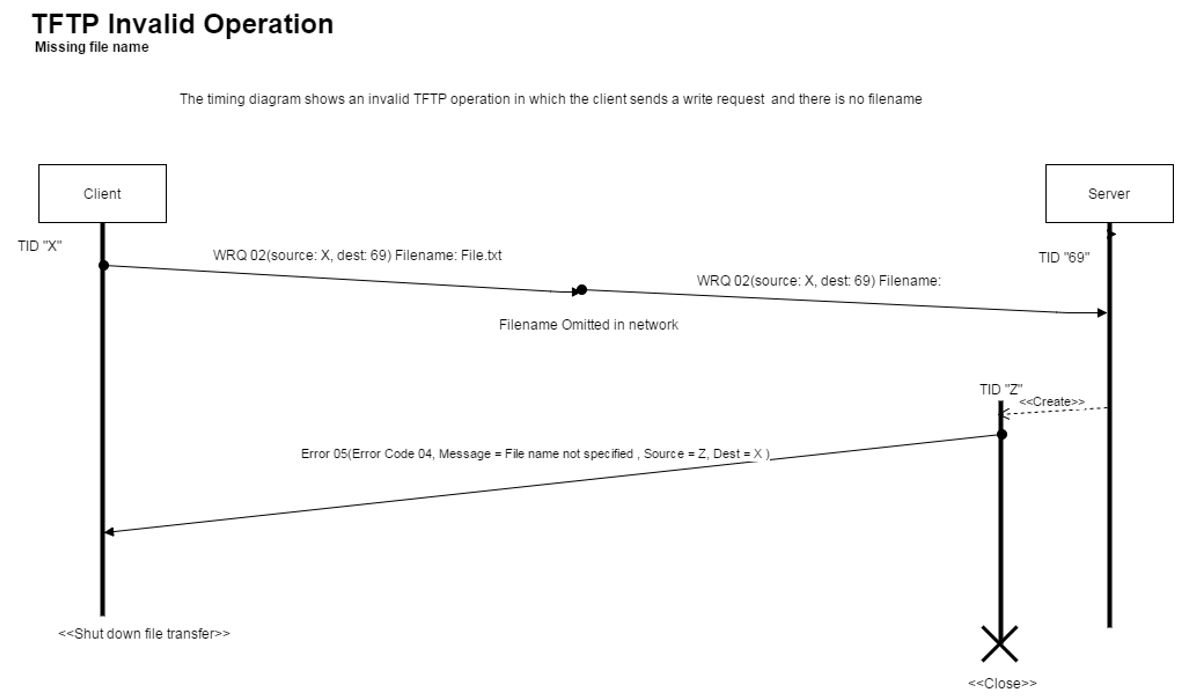
This timing diagram demonstrates the client sending a write request to an error simulation in the middle, where simulator alters the op-code of the packet to something invalid, then sends it to the server. The server determines this is an error, so it creates a thread and responds with an error packet with op-code 05 containing error code 04 and a description about the error. Same error handling occurs with other packets with invalid or inconsistent (when transfer already in progress) op codes.



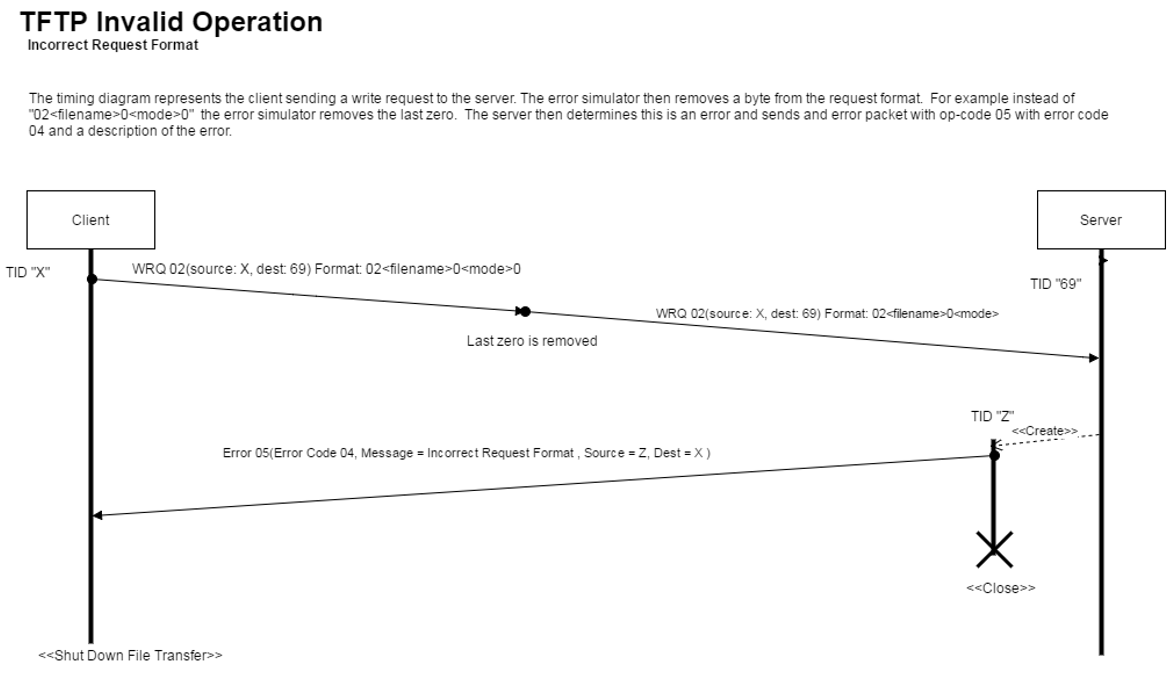
**Figure 4: Incorrect Block Number**



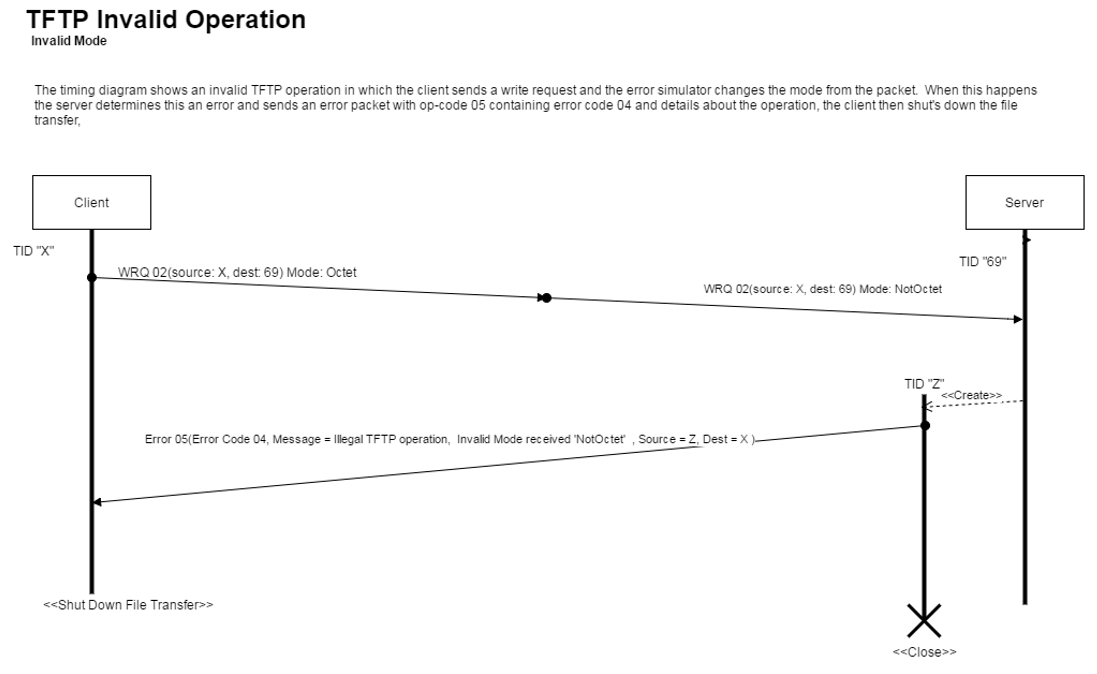
**Figure 5: Missing File Name**

****

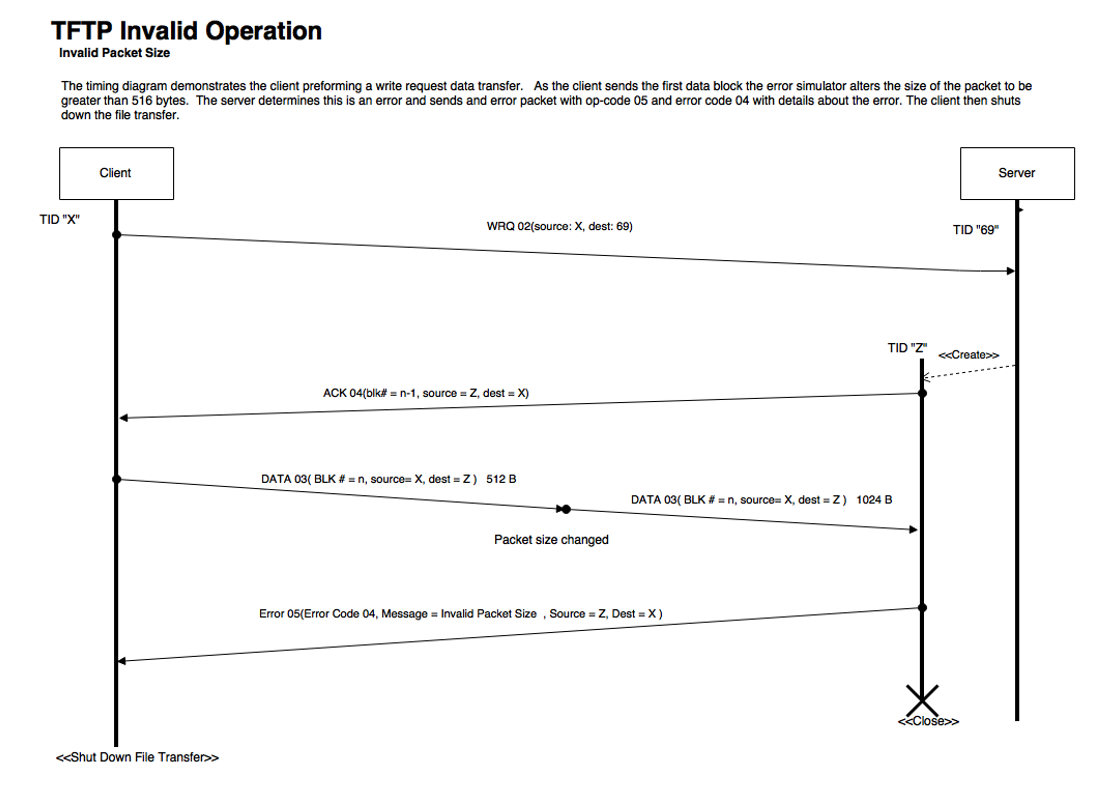
**Figure 6: Incorrect Request Format**

****

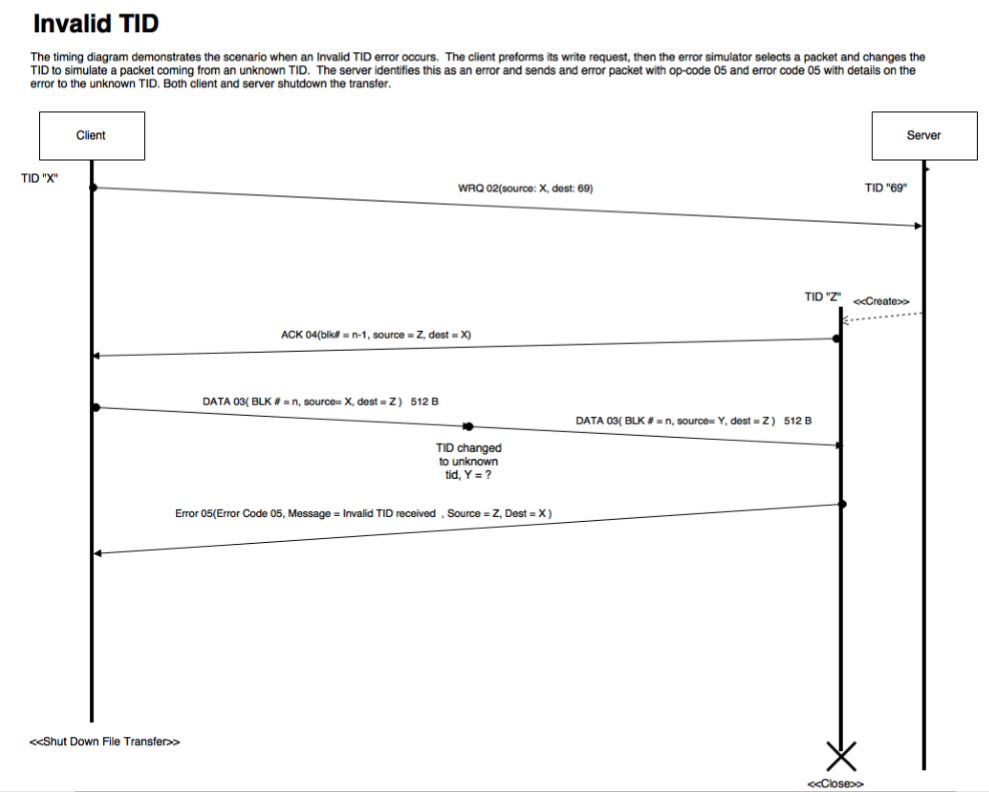
**Figure 6.1: Invalid Mode**



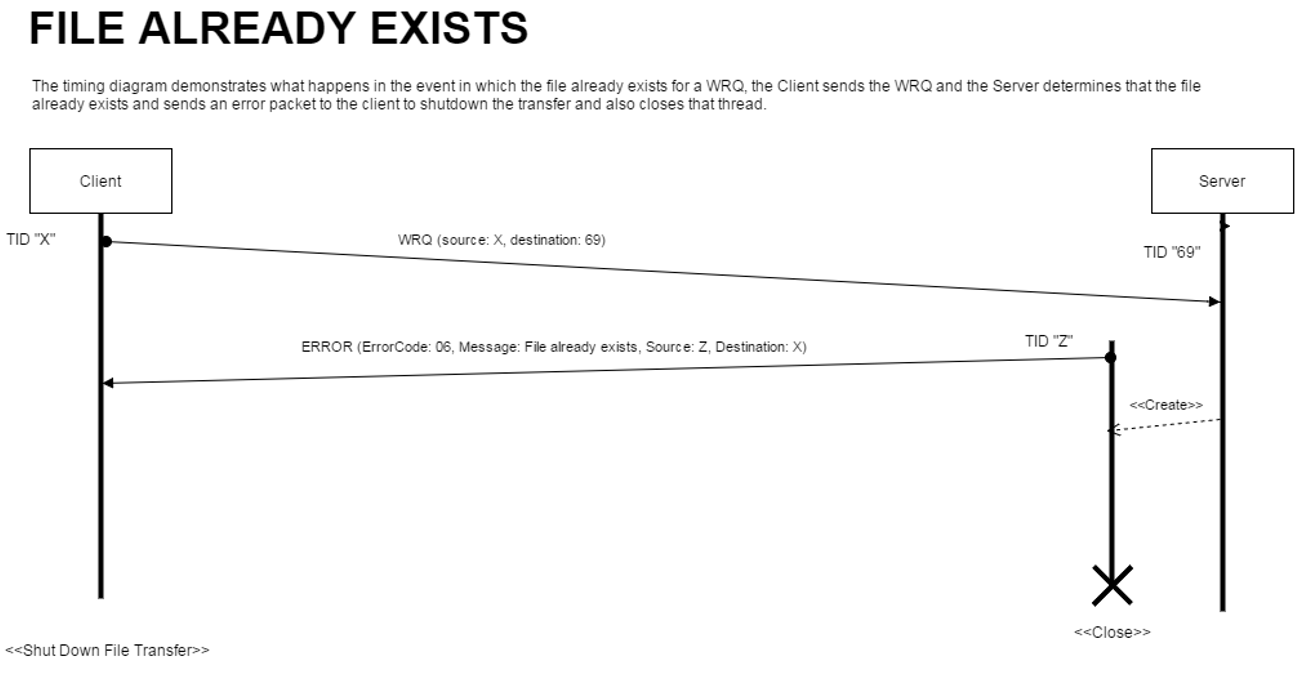
**Figure 6.2: Invalid Packet Size**



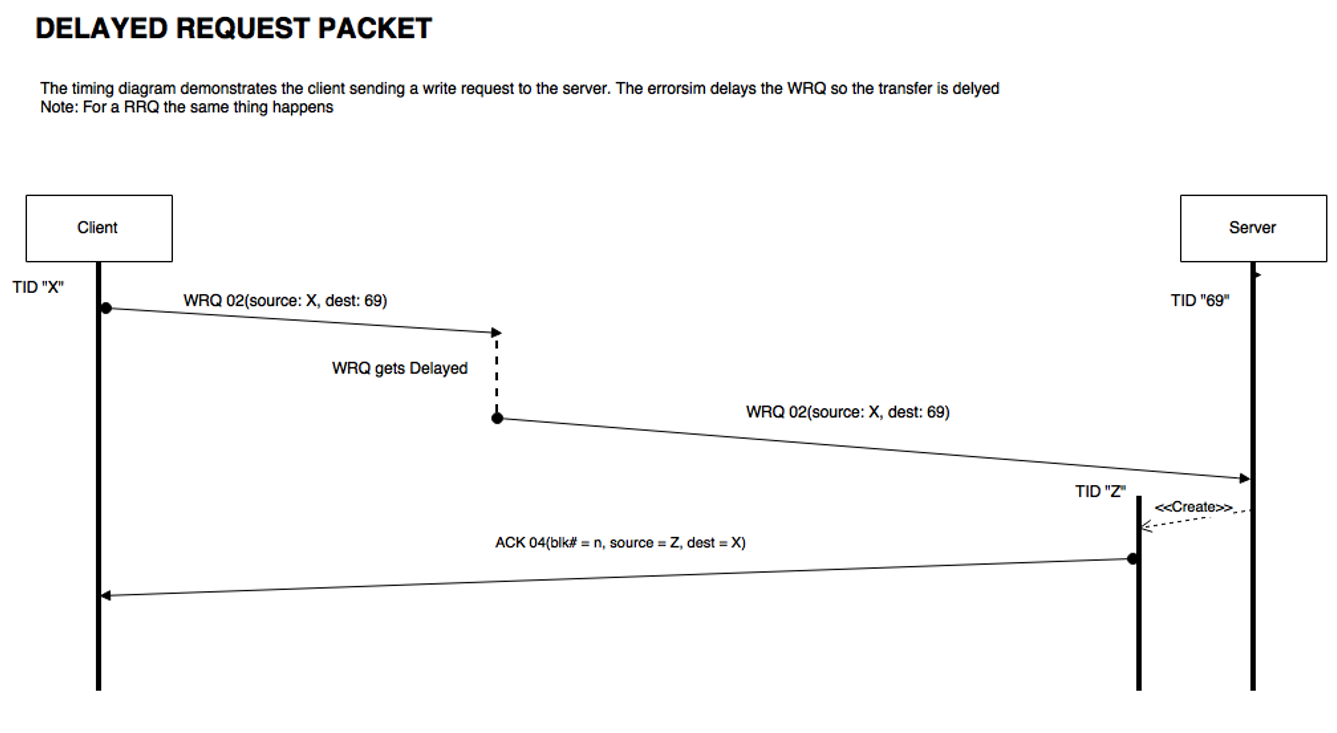
**Figure 7: Unknown Transfer ID**

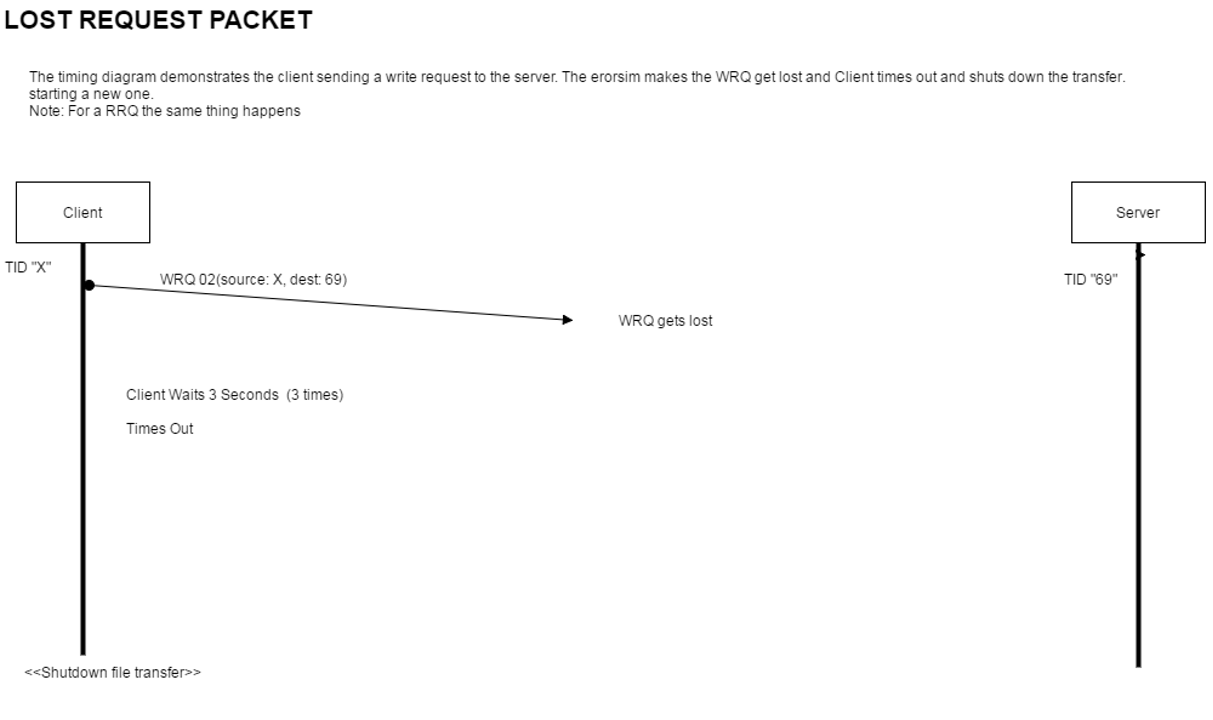


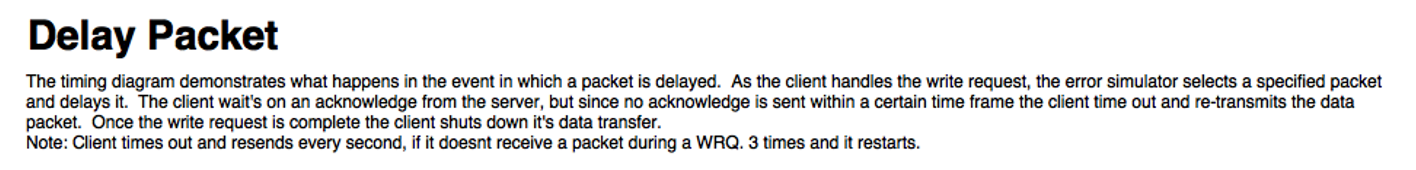
**Figure 8: File Already Exists**

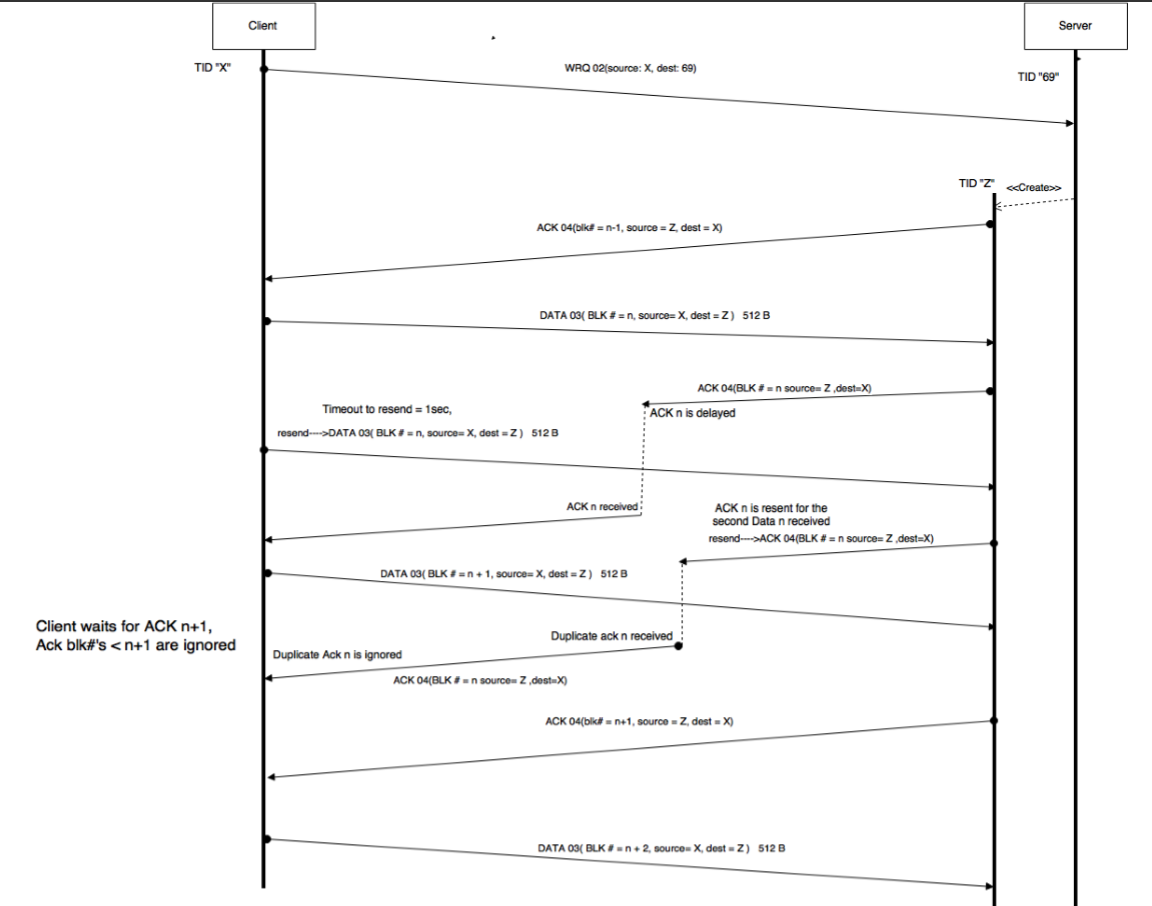


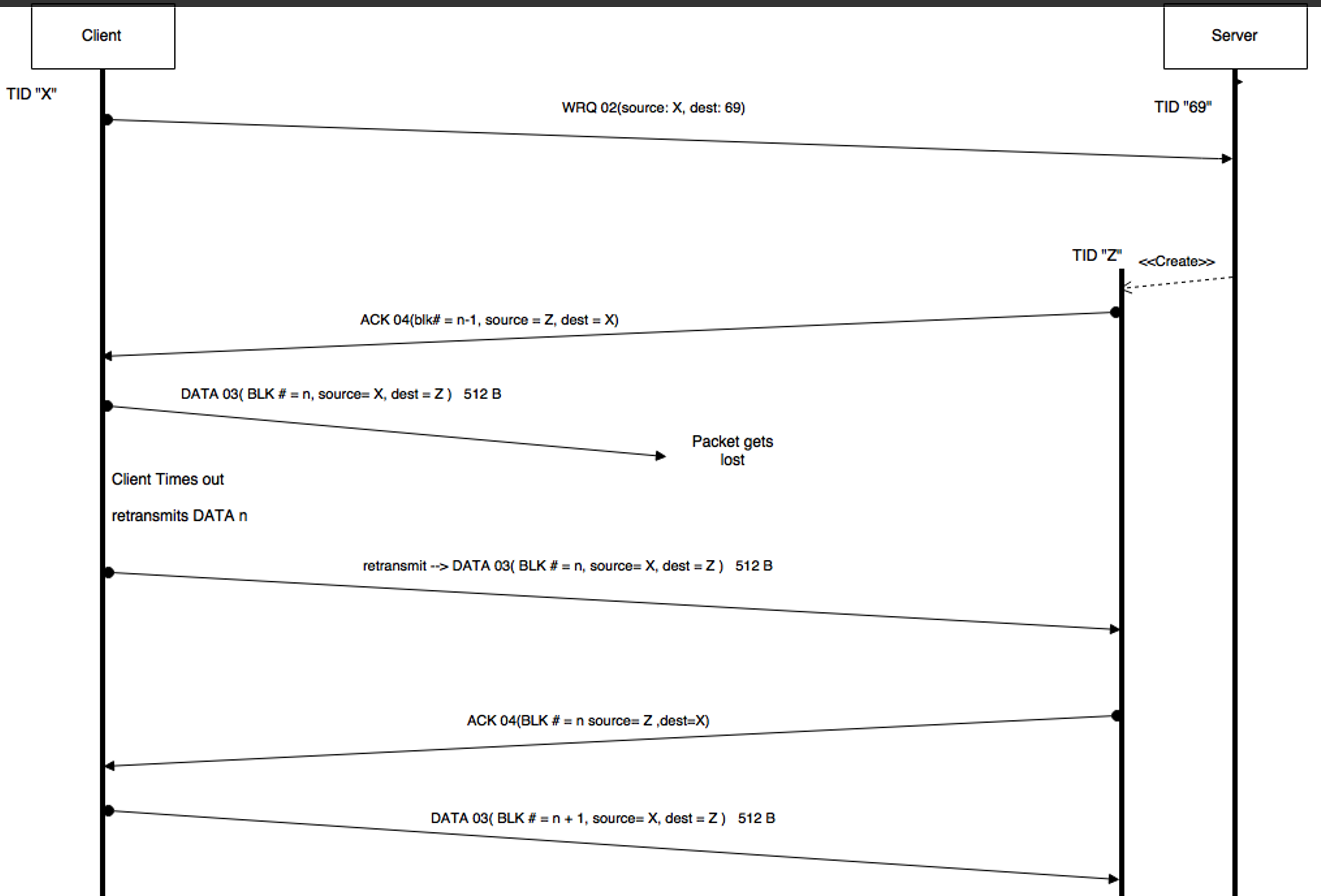
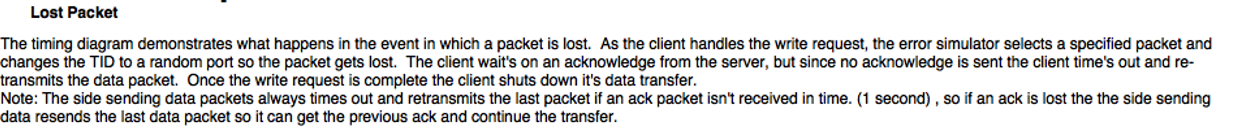
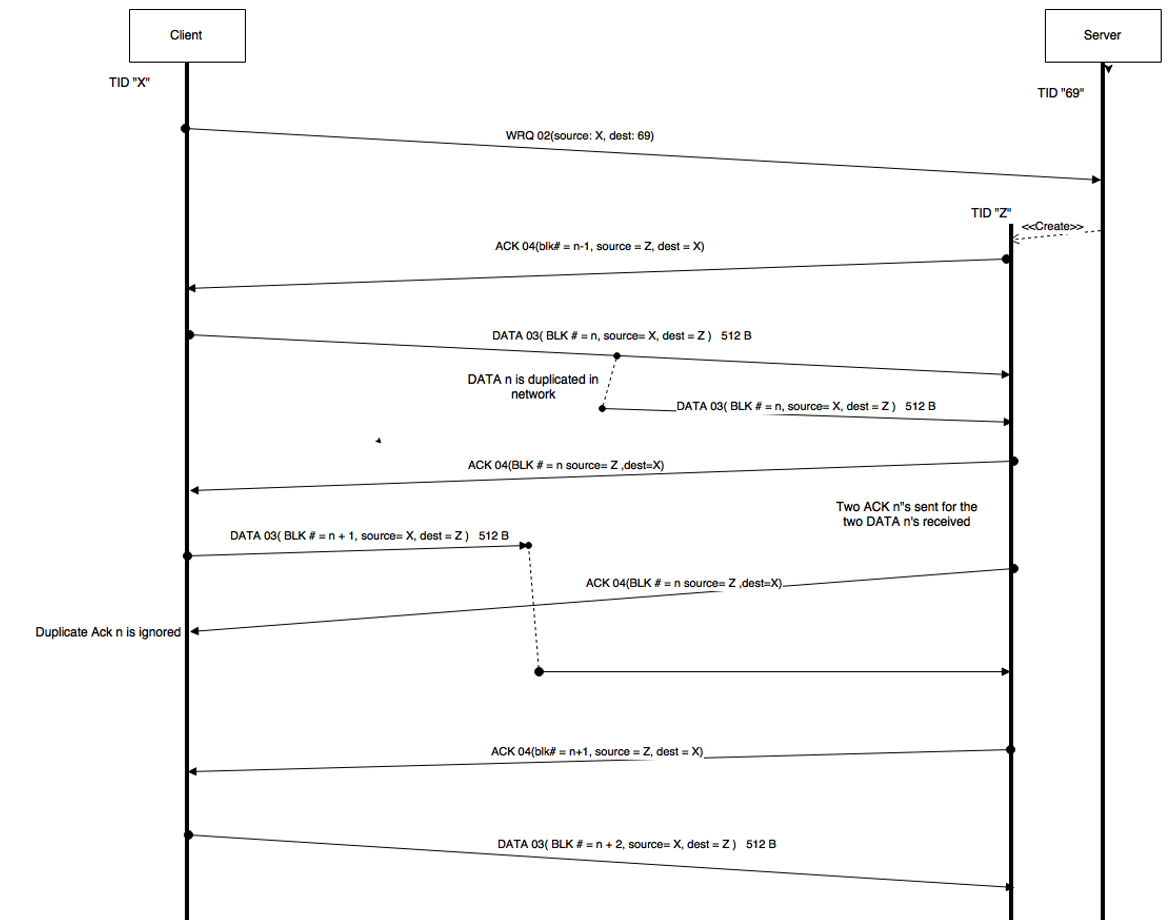
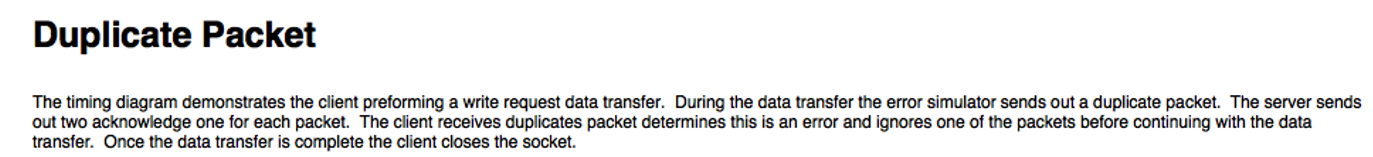
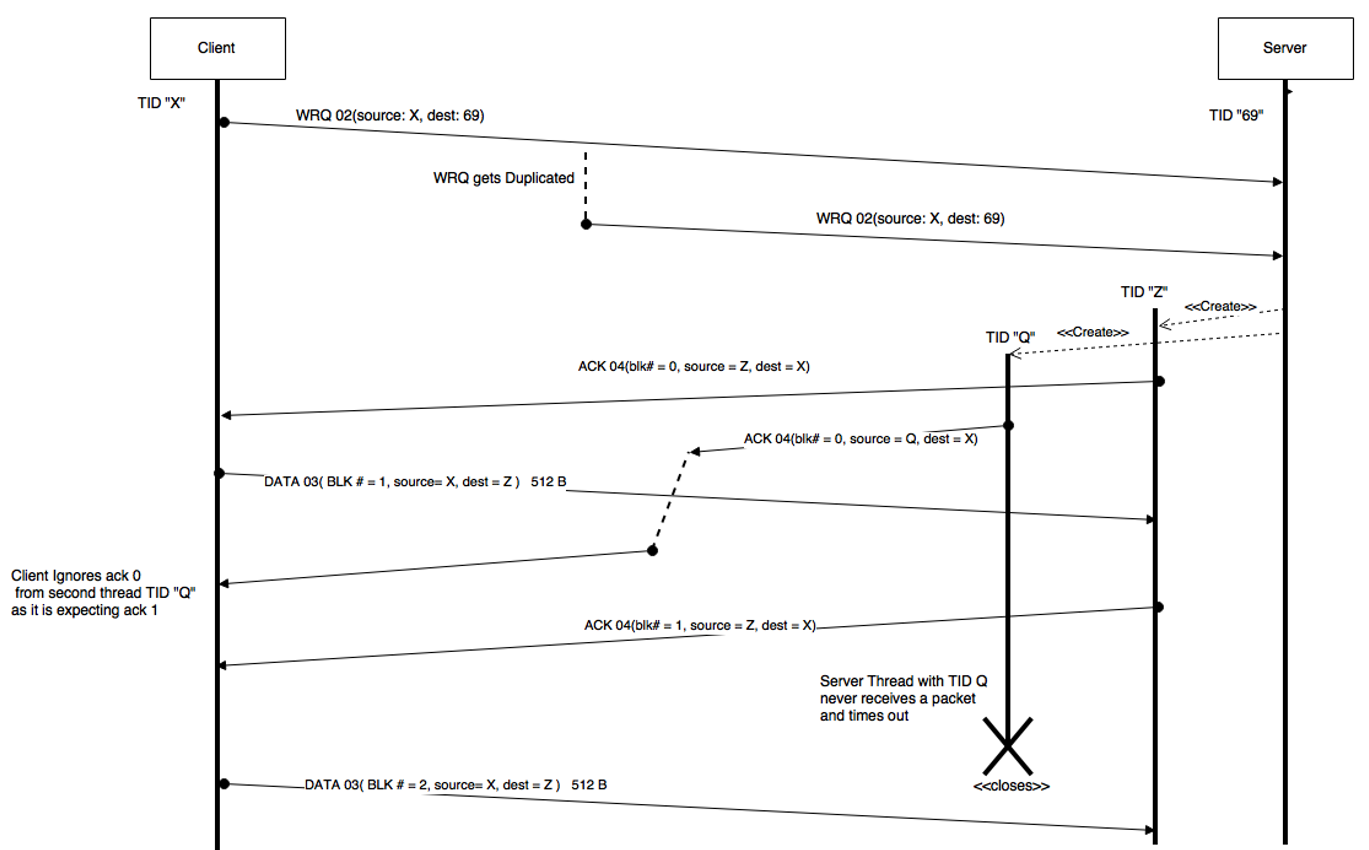
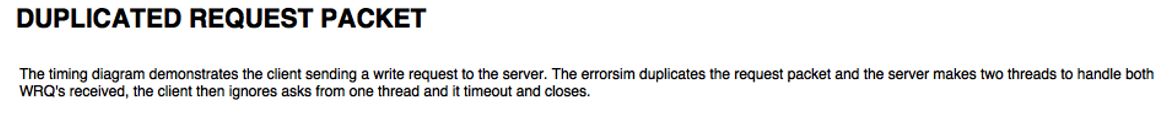
**Timing Diagrams for transfer errors**









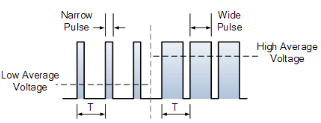


**Design – Hardware**

**DC Motor**

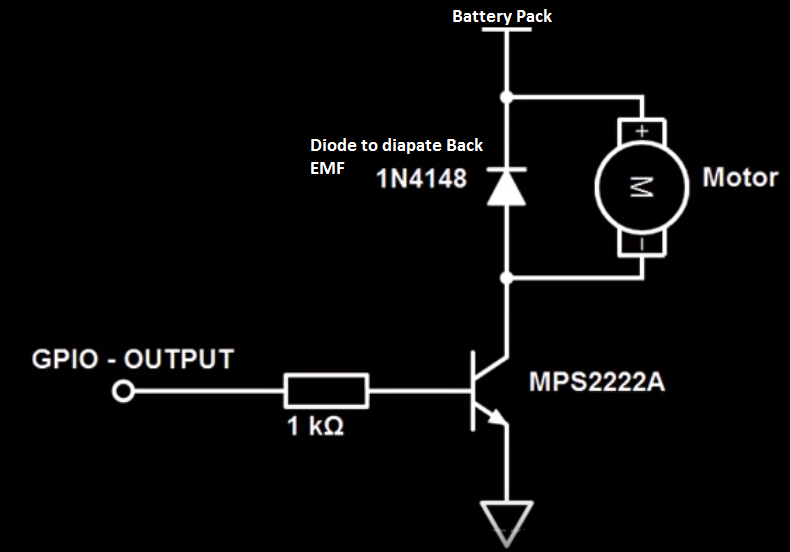
DC motor not purchased yet, trying to determine a good fit. There are some important specifications.

* No Load RPM: Looking for a decent RPM motor so when the Ring is attached on top it can still achieve a RPM to simulate the persistence of vision effect. At around 30 frames/ per = 30 RPM under load. The weight of the load is unknown since the part has not been 3d printed yet.
* Nominal Voltage: This will determine the main battery pack that should be used to match the nominal voltage. Matching the nominal voltage will maximize its efficiency and will not shorten it's lifespan.
* Stall Torque: This is the maximum torque the motor can apply before the shaft halts spinning and starts to be damaged. The torque required by the LED ring can be calculated and it should be less than 1/3 of the stall torque. What may happen is that the 3d LED ring is printed without the socket for a motor shaft so the weight and required torque can be determined, then purchase a suitable motor, and then 3d model an adapter to put it all together.

The DC motor will be controlled by a Raspberry pi where it is connected at the GPIO that controls the Pulse Width Modulation. Essentially the more voltage applied to the motor the faster it will rotate and a PWM waveform allows us to control the voltage.

 The proportion that the HIGH pulse is in a cycle is the duty cycle, which determines the speed of the motor. For example, if the HIGH pulse is 50% of a cycle, then the duty cycle is 50% and the speed of 50% of maximum.

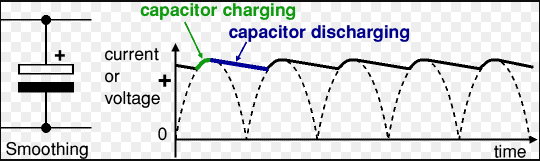
**DC motor – Sample Schematic**

****

* The GPIO output from the Raspberry Pi to control the pulse width modulation.
* The diode is used to suppress and dissipate the back EMF from the motor when turned off.  When the output is set low and the MPS2222A switch is opened, the supply to the motor is switched off. This causes the motor's magnetic field to collapse which generates a back EMF that contains a high amount of current and also flips the polarity of the motor. With the polarity flipped, the current goes through the diode to be dissipated.

**Design considerations that may be implemented.**

* Since PWM is a series of ON and OFF pulses this means the switch is closed and opened to control current, resulting in motor speed, being somewhat choppy at lower duty cycles. A capacitor could be implemented to store some energy to smooth the PWM output.



* Instead of PWM to control speed, a potentiometer on the resistor could be used to control the current draw, resulting in speed control of the motor. This will allow the current to be constant once a resistance has been adjusted to with this variable resistor. 

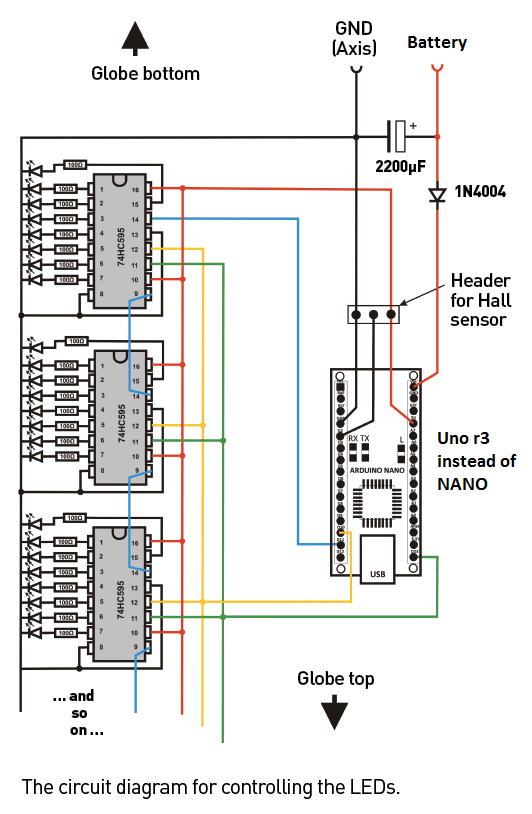
**DC Motor Test procedures**

* Need tests to ensure the Raspberry pi PWM output or PWM motor controller is regulating the DC motor correctly. Three options, using PWM to control the DC motor (if using a potentiometer tests are similar), tests 1 and 2 are with no load:

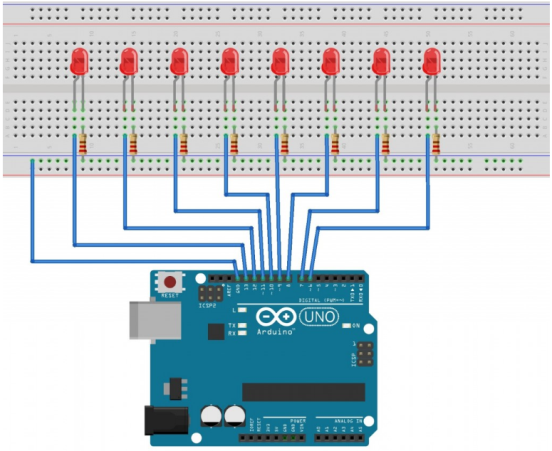
1. If the DC motor purchased has a graph for voltage vs rpm, we can first determine, from the PWM, the average voltage for a set of PWM waveforms then compare the the RPMs to the graph to check consistency with manufacturer. Then we could physically measure the RPM with some sensor.
2. The results of the test could be purely based on observation, where there are a set number of PWM waveforms to output and use our judgement.
3. If using the hall sensor and tracking the number of interrupts in a chosen time frame, a load would have to be attached. This introduces the load and torque to the calculations when we want to do comparisons to the measured values.

**LED and Hall sensor Sample Schematic**

* Using shift registers is to allow more LEDs to be controlled with the Arduino. Images are converted to bit patterns, so each pin of the Shift register, 74HC595 from the kuman kit, we can fill all the registers to be able to control many LEDs.



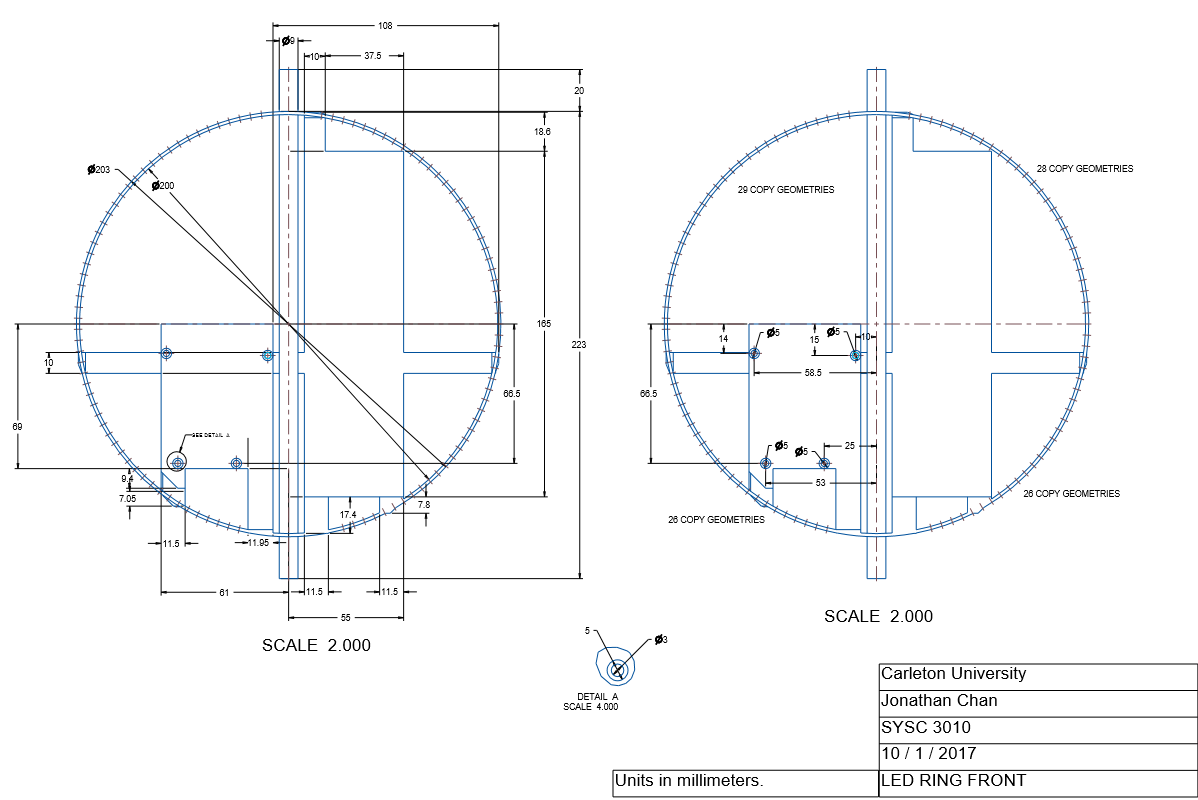
* The minimum requirement for this project is use the digital outputs to control a lower number of LEDs (sample schematic below)



**LED and Hall sensor Test Procedures**

* Before placing the LEDs on the circle, an automated test to test the blink rate of the LEDs. After soldering or glued the LEDs the test would be run again to ensure all they work.
* To test the Hall sensor interrupt, enable a LED bit pattern where the LEDs turn on only when the hall sensor is moved over the magnet near the ring. If the LEDs do not turn on at this point, since we test the LEDs beforehand, it means the interrupt is incorrectly implemented.

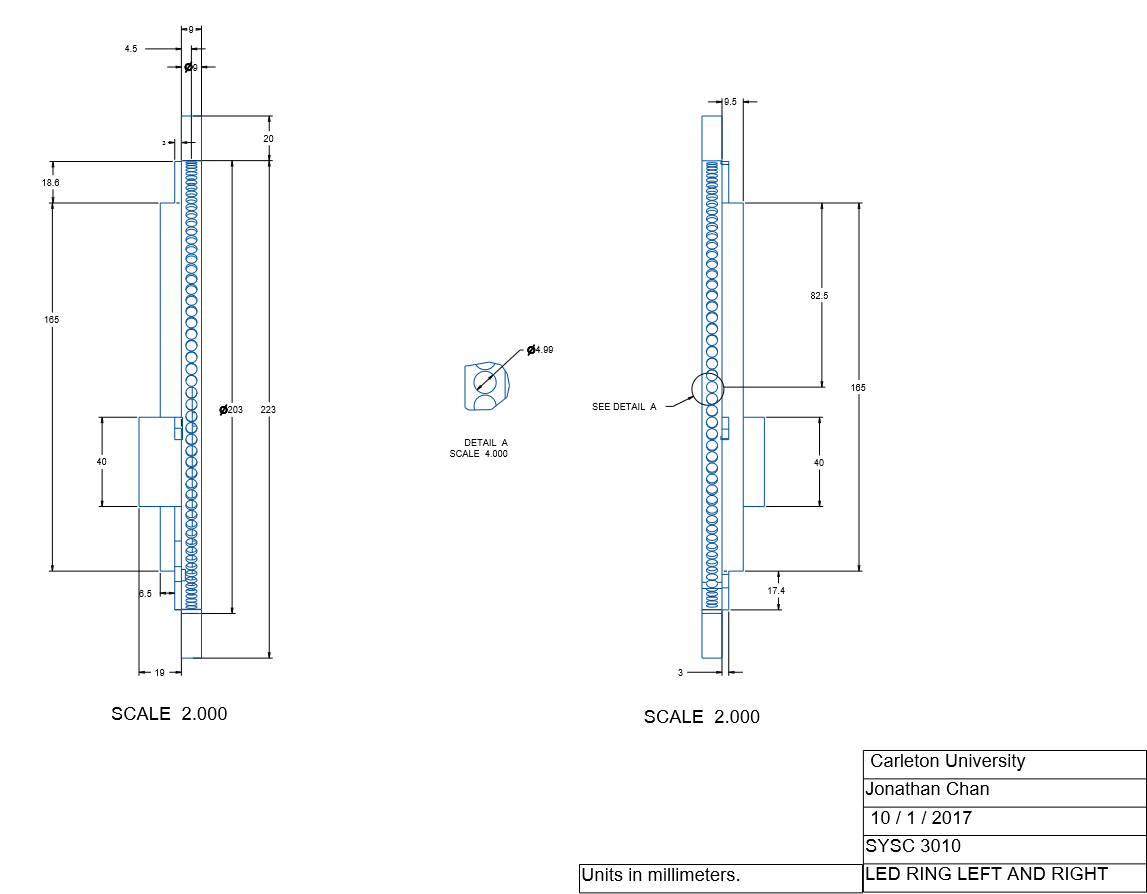
**Design – LED Ring Drawings Front**



**Design – LED Ring Drawing Back**

Changes to be made

**Design – LED Ring Drawing Left and Right**



**Design – LED Ring Drawing Top and Bottom**

Changes to be made

**Design – Software**

Incomplete

**The Team Project Implementation**

Under construction

**The Team Project Test Plan and Results**

**Hardware**

* Before placing the LEDs on the circle, an automated test to test the blink rate of the LEDs. After soldering or glued the LEDs the test would be run again to ensure all they work.
* Need tests to ensure the Raspberry pi PWM output or PWM motor controller is regulating the DC motor correctly. Three options, using PWM to control the DC motor (if using a potentiometer tests are similar), tests 1 and 2 are with no load:

1. If the DC motor purchased has a graph for voltage vs rpm, we can first determine, from the PWM, the average voltage for a set of PWM waveforms then compare the the RPMs to the graph to check consistency with manufacturer. Then we could physically measure the RPM with some sensor.
2. The results of the test could be purely based on observation, where there are a set number of PWM waveforms to output and use our judgement.
3. If using the hall sensor and tracking the number of interrupts in a chosen time frame, a load would have to be attached. This introduces the load and torque to the calculations when we want to do comparisons.

* To test the Hall sensor interrupt, enable a LED bit pattern where the LEDs turn on when the hall sensor is moved over the magnet near the ring. If the LEDs do not turn on at this point, since we test the LEDs beforehand, it means the interrupt is incorrectly implemented.

**Software**

* GUI buttons and overall functionality
* RPi1 image processing could have a post-processing test (maybe a unit test) to ensure proper image handling
* Connections between all components must be robust. For every connection type considered, all corner cases will be handled to ensure no data packets are dropped or lost. If they are, proper error handling must be implemented.
* Arduino must properly map processed image data to LEDs.

**The Team Project Process**

|  |  |  |
| --- | --- | --- |
| **Role** | **Assigned To** | **Date (Week of)** |
| Compile component list | All | Oct 1. |
| Component acquisition | Jon | Oct 1. |
| 3d Model of LED ring created.  DC motor pwm control program in python | Jon | Oct 1 |
| Hardware and communication Design | Jon | Oct 15 |
| GUI outline with button placeholders | Denis | Oct 23 |
| Image processing into bit array and calculating signal interval for LEDs. In python. | Jon | Oct 30 |
| Test plans for a general utility class and embedded unit test. | Jon | Nov 6 |
| RPi1 image processor | Eliab | Nov 13 |
| By now, 3d model printed and components attached | Jon | Nov 13 |
| Confirm data transfer between components fill in GUI functionality | Denis | Nov 20 |
| LED image pattern with mono colour | Eliab | Nov 20 |
| Image with r,g,b combinations | Eliab | Dec 4 |
| Automated conversion of an image into a pixel array of r,g,b values. | Denis | Dec 4, concurrent development |
| Fully functioning globe | All | Dec 18 |