Appendix

Perl Script

The Perl script that was written to encode, edit, and insert the information into the database a various files.

```
#!/usr/bin/perl -w
use DBI;

# Flow.pl
# This program will discard the sif and dif information
# from the newflow.txt file and mask the source and destination
# ip addresses then add this information to a new file

# *****OUTLINE*****s
# Get rid of Sif and Dif
# use the new file to pull out IP addresses, and encode them
# Take each part of the IP address into variables
# Multiply each part by 265 (combinations) to respective power
# Add each vaiable after computation for IP's integer
# Run cipher to encode the vaiables by shifting the number order
# Replace in file
# Insert all of the data into the database

# *****INSTRUCTIONS*****
# Before running make sure the outfile of olft program matches the infile of this program
# Change any outfile names as needed to prevent overwriting

# *****Database*****
# host - $hostname
# port number - $port
# name of database - $database
# database username - $username
# database password - $password
# connection string to a specific database - $dsn
# actual connection - $dbn

# *****Open flowdata file*****
$FlowFile = "flowdata.txt";
open(FIN, $FlowFile) or die "The file $FlowFile could not be found. \n";

# *****Open newflow file*****
$TheFile = "newflow.txt";
```

```
open(IN, $TheFile) or die "The file $TheFile could not be found. \n";

# *****Open outFlow file*****
$OutFile = ">outFlow.txt";
open(OUT, $OutFile) or die "The file $OutFile could not be found. \n";
#$OutIPs = ">IPs.txt";
#open(OUTIPs, $OutIPs) or die "The file $OutFile could not be found. \n";

# *****Header variables*****
$Sif = 0;              # Take in but ignore
$SrcIPaddress = 0;     # Source IP address
$Dif = 0;              # Take in but ignore
$DstIPaddress = 0;     # Destination IP address
$Pr = 0;               # Protocol (hex)
$SrcP = 0;             # Source Port (hex)
$DstP = 0;             # Destination Port (hex)
$Pkts = 0;             # Packets
$Octets = 0;           # Octets (bytes)
$StartTime = 0;        # Start time of transfer
$EndTime = 0;          # End time of transfer
$Active = 0;           # Time in ms the flow was active
$BPk = 0;              # Bytes per packet
$Ts = 0;               # Type of service
$Fl = 0;               # Flags

$SipInt = 0;           # Source IP address as integer
$DipInt = 0;           # Destination IP address as integer

# *****Encoding variables*****
$pos = 0;                          # position of the numbers
$pos2 = 0;
$shift = 4;                        # how to shift the number ordering
$sHold = $shift - 1;

$track = 0;                        # counters
$i = 0;
$lineCount = 1;
$message = 10000;

$NewFlow = "newflow.txt";
```

```
open(IN, $NewFlow) or die "The file $NewFlow could not be opened. \n";

print "Reformating complete.\n";

# *****IP to int*****
sub IPmath {
	($first, $second, $third, $fourth) = split(/\./, $_[0]);
	$ipInt = 0;
	$first = $first * 256**3;
	$second = $second * 256**2;
	$third = $third * 256**1;
	$fourth = $fourth * 256**0;
	$ipInt = $first + $second + $third + $fourth;
	return $ipInt;
}

# *****Encryption*****
sub ENCODE {
	$Int = $_[0];
	@SOURCE = split(/ */, $Int);
	$count = 0;
	$epos = 0;
	for ($i = 0; $i <= $#SOURCE; $i += 1) {
		for ($j = 0; $j < 10; $j +=1) {
			if ($SOURCE[$i] == $nums[$j]) {
				$eSOURCE[$epos] = $newNums[$j];
				$count += 1;
				$epos += 1;
			}
		}
	}
	return join('',@eSOURCE);
}

# *****Time Formatting (2008-06-03 xx:xx:xx miliseconds)*****
sub TIME {
	($Date, $Time, $MS) = split(/\./, $_[0]);
	($Day1, $Day2, $Month1, $Month2) = split(/ */, $Date);
	$DateTime = "2008-$Day1$Day2-$Month1$Month2 $Time $MS";
	return $DateTime;
```

```
}

# *****Number ordering for encoding the ip addresses*****
# orginal number ordering
#print STDOUT "Conversion chart:\n";

@nums = ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9');
@newNums = ();

# create a new number ordering
for ($pos = 0; $pos < 11 - $shift; $pos +=1) {
        $track = $pos + $sHold;
        $newNums[$pos] = $nums[$track];
        #print STDOUT "$pos $newNums[$pos] \n";
}
for ($pos2 = 11 - $shift; $pos2 < 10; $pos2 +=1) {
        $newNums[$pos2] = $nums[$i];
        #print STDOUT "$pos2 $newNums[$pos2] \n";
        $i += 1;
}

# *****Print header*****
$head = <IN>;
($Sif, $SrcIPaddress, $Dif, $DstIPaddress, $Pr, $SrcP, $DstP, $Pkts, $Octets, $StartTime,
$EndTime, $Active, $BPk, $Ts, $Fl) =$
print OUT "$SrcIPaddress $DstIPaddress $Pr $SrcP $DstP $Pkts $Octets StartDate $StartTime
StartMiliseconds EndDate $EndTime E$
#print OUTIPs "Encoded Original\n";

print STDOUT "may take a few minutes to complete \n";
print STDOUT "running... \n";

# *****Get rid of Sif and Dif, apply IP to int math, encode the ints, and insert information into
the database*****
while (<IN>) {
        $line = $_;
        ($Sif, $SrcIPaddress, $Dif, $DstIPaddress, $Pr, $SrcP, $DstP, $Pkts, $Octets, $StartTime,
$EndTime, $Active, $BPk, $Ts$

        # $SrcIPaddress
        $SipInt = &IPmath($SrcIPaddress);
        # $DstIPaddress
        $DipInt = &IPmath($DstIPaddress);

        # *****Encrypt the int*****
        $esource = &ENCODE($SipInt);
        $edestination = &ENCODE($DipInt);

        # *****New date format*****
        $StartDateTime = &TIME($StartTime);
        ($StartDate, $StartTime, $StartMS) = split(/\s+/, $StartDateTime);
        $StartDate = "$StartDate $StartTime";
        $EndDateTime = &TIME($EndTime);
        ($EndDate, $EndTime, $EndMS) = split(/\s+/, $EndDateTime);
        $EndDate = "$EndDate $EndTime";

        # *****Hex conversion*****
        $SrcPnum = hex($SrcP);
        $DstPnum = hex($DstP);
        $Tsnum = hex($Ts);
        $Flnum = hex($Fl);
        $Prnum = hex($Pr);

        print OUT "$esource $edestination $Pr $SrcPnum $DstPnum $Pkts $Octets $StartTime
$StartMS $EndTime $EndMS $Active $BPk$
        #print OUTIPs "$esource $SrcIPaddress\n";

        # *****Insert into database*****
        #$dbh->do("INSERT INTO flowdata VALUES ($SipInt, $DipInt, $Prnum, $SrcPnum,
$DstPnum, $Pkts, $Octets, '$StartDateTime'$

        # *****Line counter*****
        if ($lineCount == $message)
        {
              print STDOUT "$message lines added...\n";
              $message +=10000;
        }
        $lineCount +=1;
}
```

```
close(IN);
close(OUT);
#close(OUTIPs);
print STDOUT "Complete. $lineCount lines.\n";
#$dbh->disconnect;
```

Processing Code

The code written to read network information from a file and create a graph based on the information.

```
// netdata.pde
// X = Src host
// Y = Dst host
// Z = Duration
// Color = Octets

color a, b, c, d, e, f, col;
PFont Gfont;
int value;
String[] lines;
String[] parts;
int count = 2;
int k = 0;

float cameraY;
float fov;
float cameraZ;
float aspect;
int rotnum = 1;
float zoom = 0.0;

// Adjust to configure to your monitor
int mwidth = 1280;
int mheight = 900;

void setup() {
  size(mwidth,mheight, P3D);
  frameRate(30);
  background(0);
  Gfont = loadFont("Gfont.vlw");
  textFont(Gfont, 20);
}

void draw() {
  int x, y, s, e, o, z;
  a = color(255, 0, 0);     //red
  b = color(255, 132, 0);   //orange
  c = color(255, 252, 0);   //yellow
  d = color(0, 255, 0);     //green
  e = color(38, 79, 255);   //blue
```

```
f = color(155, 38, 255);  //purple

lines = loadStrings("test.txt");

for(int i = 0; i < lines.length; i++) {
 parts = split(lines[i], '\t');

  for(int j = 0; j < parts.length; j += 5){
    x = int(parts[j]);
    y = int(parts[j + 1]);
    s = int(parts[j + 2]);
    e = int(parts[j + 3]);
    o = int(parts[j + 4]);

    int[] xpoints = new int[count];
    int[] ypoints = new int[count];
    xpoints[k] = x;
    ypoints[k] = y;
    count++;
    k++;

    z = e - s;

    if (mousePressed){
      // print coordinates
      // search through and compare points for more accurate click
      for(int p = 0; p < xpoints.length; p++) {
        int xClickRange_max = xpoints[p] + 5;
        int xClickRange_min = xpoints[p] - 5;
        int yClickRange_max = ypoints[p] + 5;
        int yClickRange_min = ypoints[p] - 5;

        float xLogClickRange_max = log(float(xpoints[p])) + 5;
        float xLogClickRange_min = log(float(xpoints[p])) - 5;
        float yLogClickRange_max = log(float(ypoints[p])) + 5;
        float yLogClickRange_min = log(float(ypoints[p])) - 5;

        if((mouseX <= xClickRange_max) && (mouseX >= xClickRange_min) &&
          (mouseY <= yClickRange_max) && (mouseY >= yClickRange_min) ||
          (mouseX <= xLogClickRange_max) && (mouseX >= xLogClickRange_min) ||
          (mouseY <= yLogClickRange_max) && (mouseY >= yLogClickRange_min)) {
           fill(value);
           text("Src host: " + mouseX, 1100,880);
           text("Dst host: " + mouseY, 1100,898);
        }
      }
    }

    int acol = o;
    if (acol <= 1024){
      col = f;
    } else if ((acol > 1024) && (acol <= 10000)) {
      col = e;
    } else if ((acol > 10000) && (acol <= 100000)) {
      col = d;
    } else if ((acol < 100000) && (acol <= 1000000)) {
      col = c;
    } else if ((acol > 1000000) && (acol <= 10000000)) {
      col = b;
    } else if (acol < 10000000) {
      col = a;
    }

    float logX = log(float(x));
    float logY = log(float(y));

    if (x > mwidth) {
      stroke(col);
      fill(col);
      ellipse(logX, y, 4, 4);
      line(logX, y, 0, logX, y, z);
    }else if (y > mheight) {
      stroke(col);
      fill(col);
      ellipse(x, logY, 6, 6);
      line(x, logY, 0, x, logY, z);
    }else if ((x > mwidth) && (y > mheight)){
      stroke(col);
      fill(col);
      ellipse(logX, logY, 8, 8);
```

```
        line(logX, logY, 0, logX, logY, z);
      }else{
        stroke(col);
        fill(col);
        ellipse(x, y, 2, 2);
        line(x, y, 0, x, y, z);
      }
    }
  }
}

void mousePressed() {
  noStroke();
  //loop();
  if(value == 0) {
    value = 255;
  } else {
    value = 0;
    fill(0);
    rect(1090,860,150,150);
  }
}
```