Capstone Project Proposal

Introduction to flow Data Science by Springboard

Network Data Flow analysis

By Gabriel Fontenot

Introduction

The problem statement

Network engineers have been collecting flow data for decades regarding performance and behavior. The flow data sets collected include but not limited to device configurations, syslog (device messaging), and device snmptraps (fault management). Engineers and operations have for the most part used the collected flow data to assist with various troubleshooting and design exercises.

Using the statistical, proablitity and machine learning modules, the idea is to enhance the overall flow data analization experience and provide a conceise and targeted approach to understanding network and application behavior.

The host OS (distribution) is a constant and necesary part of the overall application structure. The intent of the analysis below is to understand how the OS is utilized by various components of the network to influence application behavior. This will include understanding the volument of traffic per OS and how each OS uses the network resourses.

The results inferred can further network engineers and architects understanding of applications and be considered when looking at network designs.

About the flow/host data set

The flow data sets subject to flow data analysis include a packet capture describing the overall network behavior (who is talking to who) and a description of the host communicating on the network during the specified time interval. The network flow data includes source and destination ip addresses along with host descriptions to assist in identifying traffic patterns and anomolies.

The intended flow data set has the following structures and descriptions.

Flows

1) host_name : Factor : the host name of the guest OS
2) last_software_update_at : int : the last time the host agent was updated
3) data_plane_disabled : logic : the status of the data plane forwarding
4) platform : Factor : the guest OS host distribution

5) agent_type : Factor : the current agent deployed on the guest os
6) current_sw_version : Factor : the current agent version as installed on the host
7) enable_pid_lookup : logic : the current status of the PID lookup functionality
8) last_config_fetch_at : int : the timeframe the guest OS last checked in

Sensors

1) start_timestamp : num : the timeframe the flow started
2) src_port : int : the source port for the selected flow
3) rev_pkts : int : the packet count for the reverse flows
4) rev_bytes : int : the byte count for the reverse flows
5) proto : Factor : the IP protocol used for the current flow
6) src_address : Factor : source IP ADDRESS of the host that initiated the flow
7) timestamp : Factor : the timestamp the flow was collected
8) fwd_bytes : int : the byte count for the forward direction of the flow
9) src_hostname : Factor : the host name of the source host in the flow
10) dst_address : Factor : destination IP ADDRESS of the host of which the source is communicating with
11) src_is_internal : Factor : the current state of the host relative to is location
12) dst_port : int : the destination port for teh selected flow
13) srtt_usec : int : the srtt latency associated with the current flow
14) vrf_id : int : the vrf id for the current flow
15) vrf_name : Factor : the vrf name for the current flow
16) fwd_pkts : int : the packet count for the forward direction of the flow
17) server_app_latency_usec : int : the application latency as derived from the application agent
18) total_network_latency_usec: int : the calculated network latency
19) total_pkts : int : the total fwd and rev packet count
20) platform_windowns : logic : the state of the platform, windows
21) total_bytes : int : the toal fwd and rev byte count

Preparing the data

RStudio Libraries

The libraries listed below are utilized to provide functionality required to parse, analyze and display the appropriate flow data sets.

```
library(dplyr)
library(tidyr)
library(stringr)
library(ggplot2)
library(rjson)
library(lubridate)
```

```
library(RColorBrewer)
library(knitr)
library(caTools)
```

Loading the data sets

The 'csv' files representing the flow analysis are loaded into the variables flowData and SensorData. The data loaded is comprised of a network capture of multiple host communicating on the network during the define collection period and a host description file containing host attributes describing the given hosts.

Cleaning the flow/host data sets

The prepare the data for analysis, the data must first be normalized.

The collected data sets contain IPv4 and IPv6 addresses. For the purpuse of this exercise, all IPv6 addresses will be removed as they are not part of the considered analysis. The data sets contain entries without hostnames defined, the assocaited flows will be identified and removed from the analysis as they are flows that do not contain agents.

To getter a holistic view of the flows, the flow data and the host data will be combined. This effort will allow for additional oppurtunities to corelate flow and host level data. Once the data sources have been combined, additional fields will be added to the data set to assist in providing analysis. To provide summarization for packet attributes, a sumamry field of the fwd and rev packets and bytes will be added. The additional fields will allow for total packet/byte count per flow. For evaluating time intervals, a new column will be added to translate the discovered time internal into an hour and minute format.

In an effort to focus the data analyzation, a filter for the specifc hosts to be evaluated will be applied. This filter will remove any entries for flows outside of the desired hosts.

When considering the overall problem set and the data to be analyzed, a dependent variable related to the host type will be required. This field will be a summarization of the type of platform and translated into a '1' or '0' representing the host type (1= Windows, 2 = Linux).

Exploring the data

Part of uderstanding the results of a network study is to first understand the data. The next few sections will provide a description of the data elements and foundataion for the analysis. The data exploration will describe various data points related to the overall traffic across the network and the hosts communicating.

Counting records and variables

Using R methods and functionality, determine various flow data set counts. Counting various aspects of your flow data set can prepare you for dealing with the content during analysis.

- Count the number of rows and columns in the flow data set, SensorData.csv.

There are a total of 160049 ROWs and 30 COLUMNs in the combined flow data set. The 160049 entries in the data set contain the flow attributes and the host descriptions accordingly. The overall number of flows/records can further be filtered to allow for a more granular view.

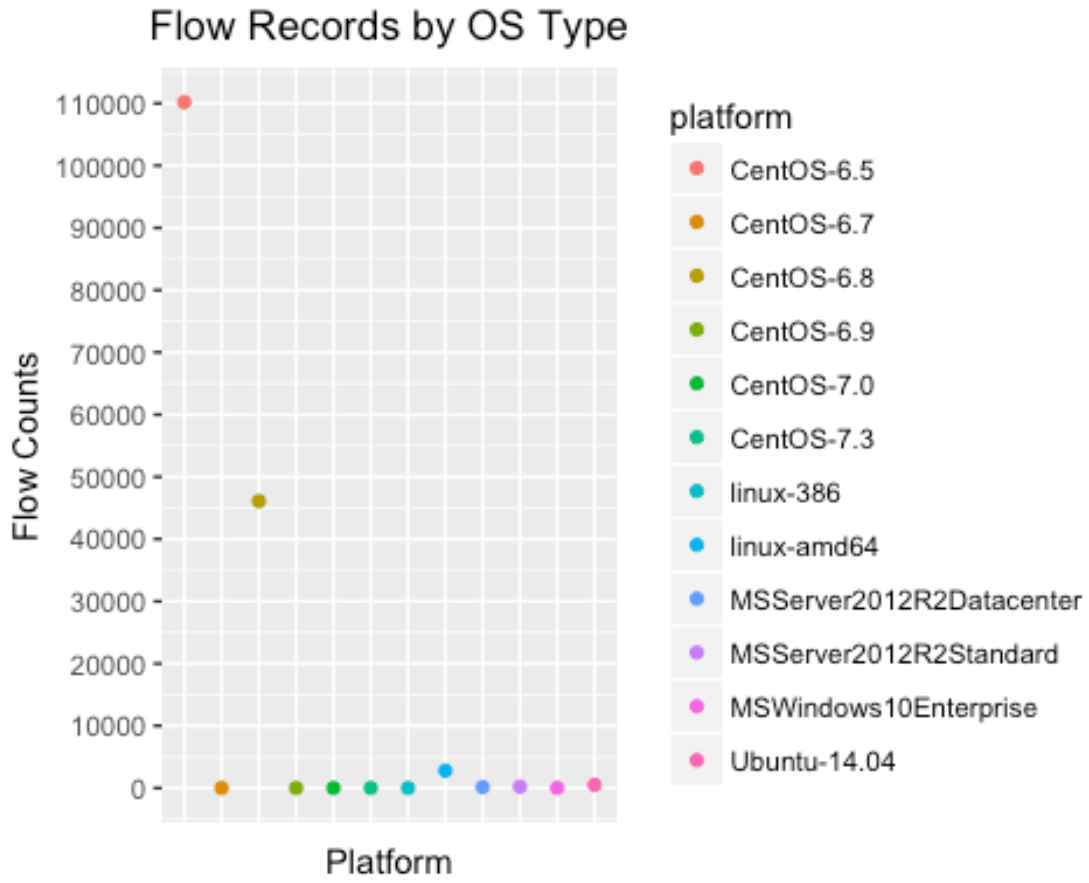
- Count the number of rows in the flow data set per platform type (the data set dependent variable).

Within the data sets analyzed, the focus of the analysis will be how host types, Windows and/or Linux, affect traffic patterns in the network. From the selected data sets there are a total of 407 flows attributed to Windows based hosts and 159642 based on Linux hosts.

Understanding the data set as related to host OS.

Using the combined data sets, we can get a better understanding of the guest OS distribution accross the collected flows. As a first step in the overall data analysis we need to gain insight into the overall distribution flows by host guest OS type, Windows and Linux distributions.

```
a <- data %>% group_by(platform) %>% tally

a %>% ggplot() +
  geom_point(aes(x=platform, y=n, color = platform)) +
  scale_y_continuous(breaks=number_ticks(10)) +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank(),
        plot.title = element_text(hjust = 0.5)) +
  xlab("Platform") +
  ylab("Flow Counts") +
  ggtitle(paste("Flow Records by OS Type"))
```




```
#data$timestamp <- factor(data$timestamp, levels=unique(data$timestamp))
```

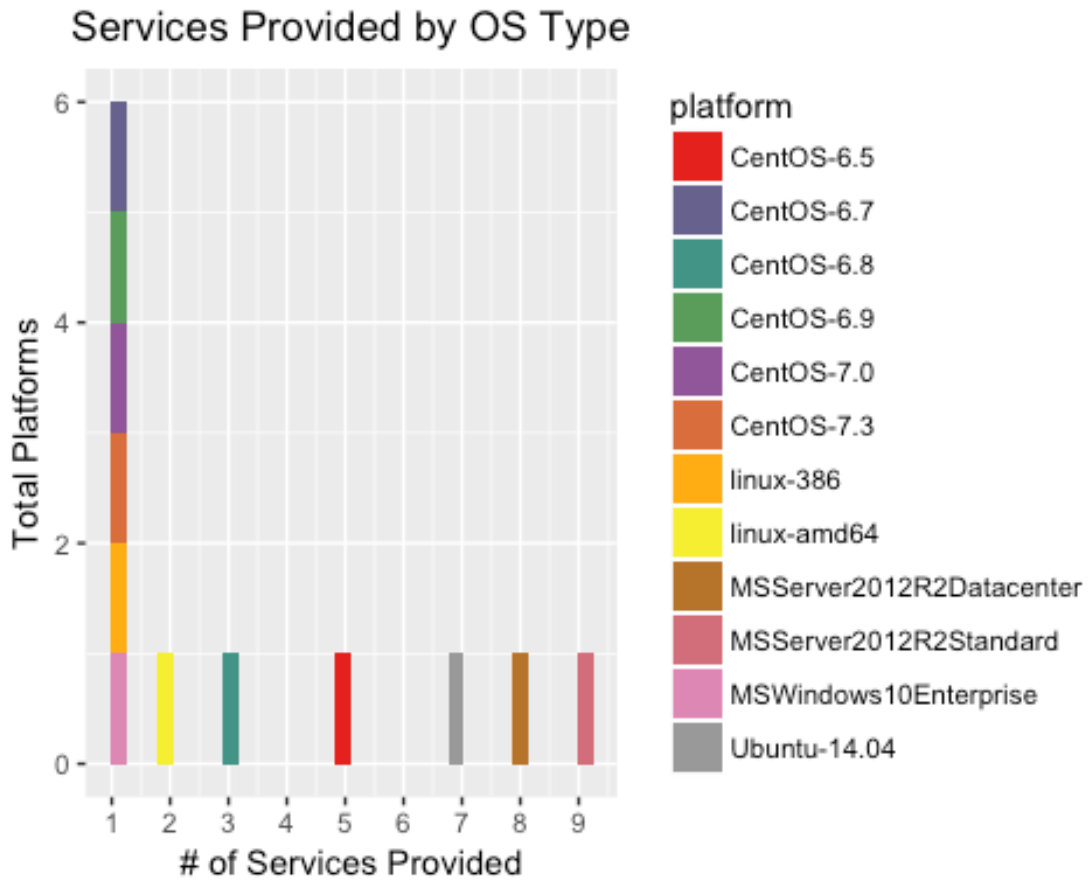
The table above shows CentOS-6.5 as the most popular OS utilized during the provided flow capture. As noted above in the table, the source of the traffic, host OS, is not evenly distributed accross OS types.

Services provided per OS

As we explore the flow data set further, we want to understand the services provided (by port number) by the guest OS. This may be an indication of the guest OS (application) that is the most utlized on the network during the flow capture timeframe. The port number (destination port) ultimatley gives an indication of the specific applitation present during the flow capture.

```
data_port <- data %>% group_by(platform) %>% summarise(n_distinct(dst_port))
names(data_port)[2]<-paste("services_provided")

data_port %>% ggplot() +
  geom_histogram(aes(x=services_provided, fill=platform)) +
  scale_fill_manual(values = getPalette(colourCount)) +
  scale_x_continuous(breaks=number_ticks(7)) +
  ylab("Total Platforms") +
  xlab("# of Services Provided") +
```

```
  ggtitle(paste("Services Provided by OS Type")) +
  theme(plot.title = element_text(hjust = 0.5))
```

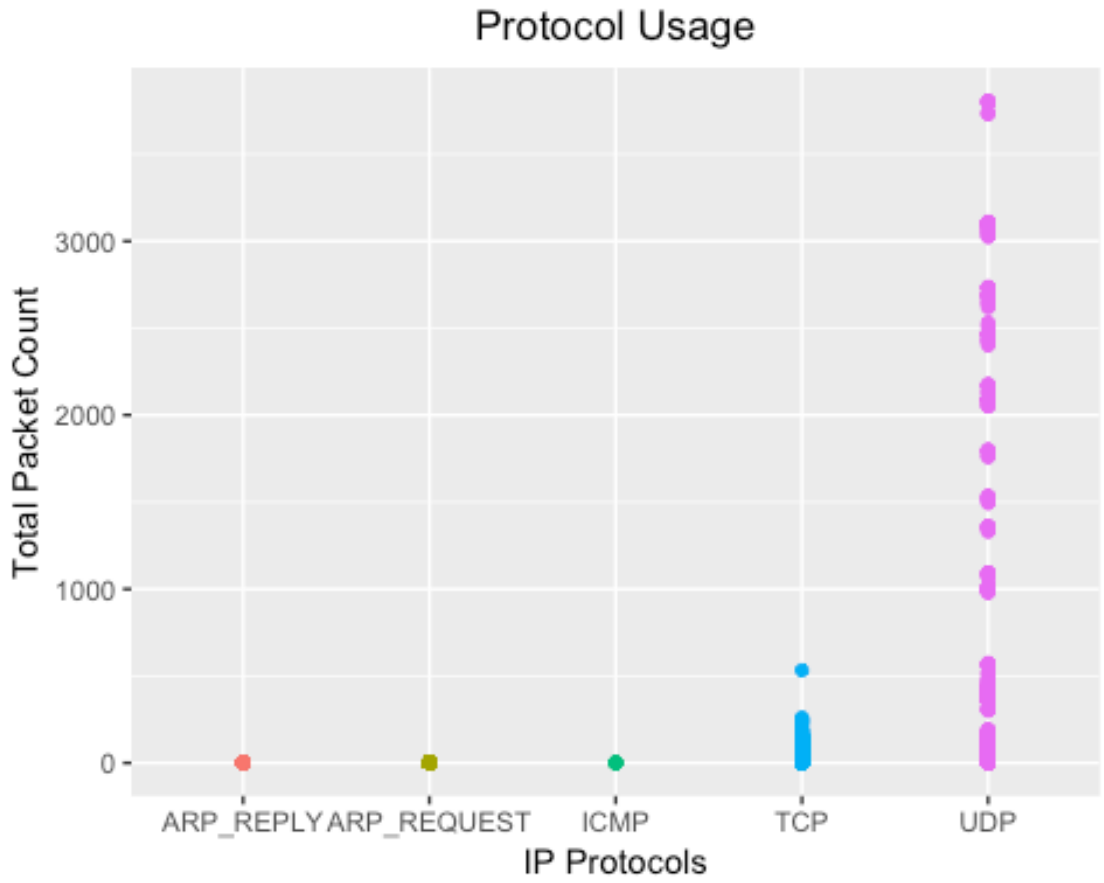


As indicated in the histogram above, we can determine that majority of the guest OS's are providing a single service. This can be interpreteded as the majority of the OSs analyzed within the flow analysis are providing serivce to a single application.

Overall Protocol Usage

As we continue to understand how applications consume overall flows within the network, it's useful to understand overall protocol distribution. The section will focus on understanding how IP protocols are implemented for the flow capture period.

```
ggplot(data, aes(proto, total_pkts)) +
  geom_point(aes(colour = proto), na.rm=TRUE) +
  theme(legend.position="none", plot.title = element_text(hjust = 0.5)) +
  ylab("Total Packet Count") +
  xlab("IP Protocols") +
  ggtitle(paste("Protocol Usage"))
```

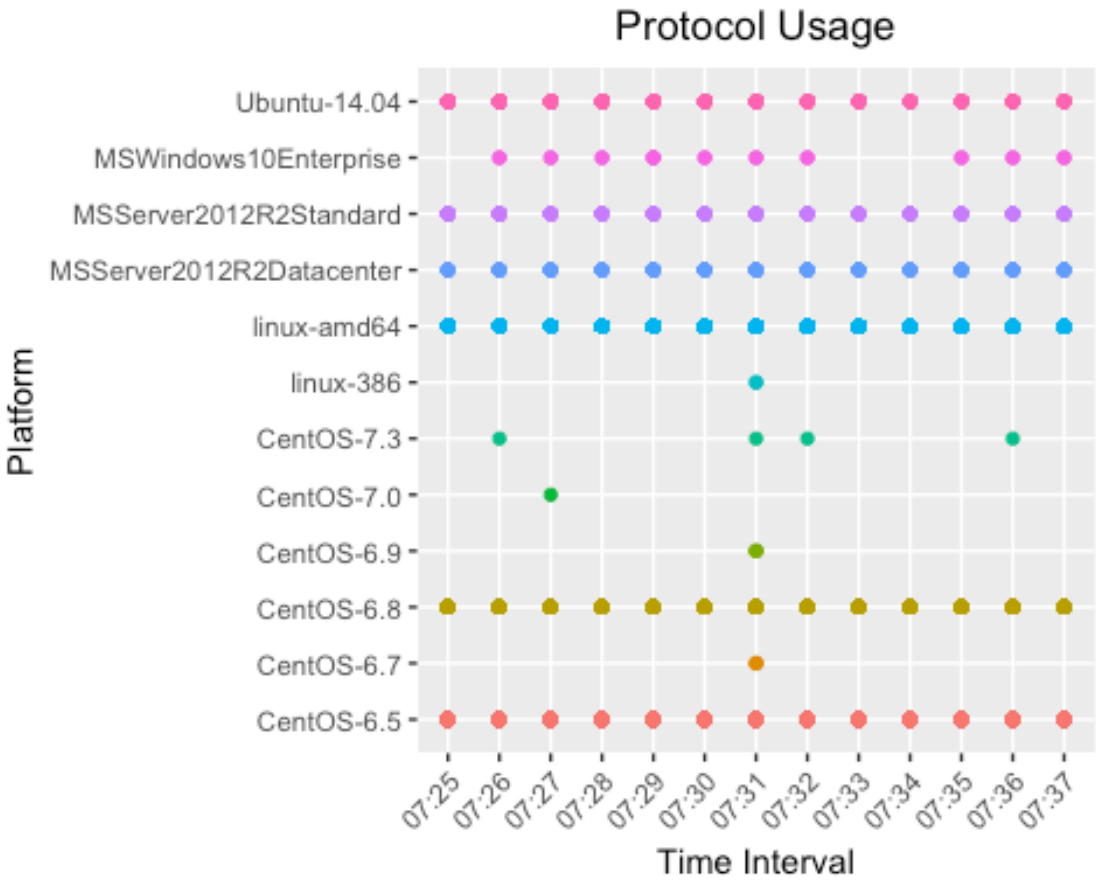


From the table above we can cleary determine that UDP is predominately utilized via the applications analysed during the capture period. The range of packets part of the UDP flows extend from 1 to 3800+ packets per flow.

Platform usage by Time

By further evaluating the OS traffic over time, we can get an idea as to when the applications communicated accross the network. The network consumption (flows consuming the netwrok) can be used to understand traffic patterns and network usage.

```
ggplot(data, aes(x=time_min, y=platform)) +
  geom_point(na.rm=TRUE, aes(colour = platform)) +
  theme(legend.position="none", axis.text.x = element_text(angle = 45, hjust
= 1), plot.title = element_text(hjust = 0.5)) +
  ylab("Platform") +
  xlab("Time Interval") +
  ggtitle(paste("Protocol Usage"))
```

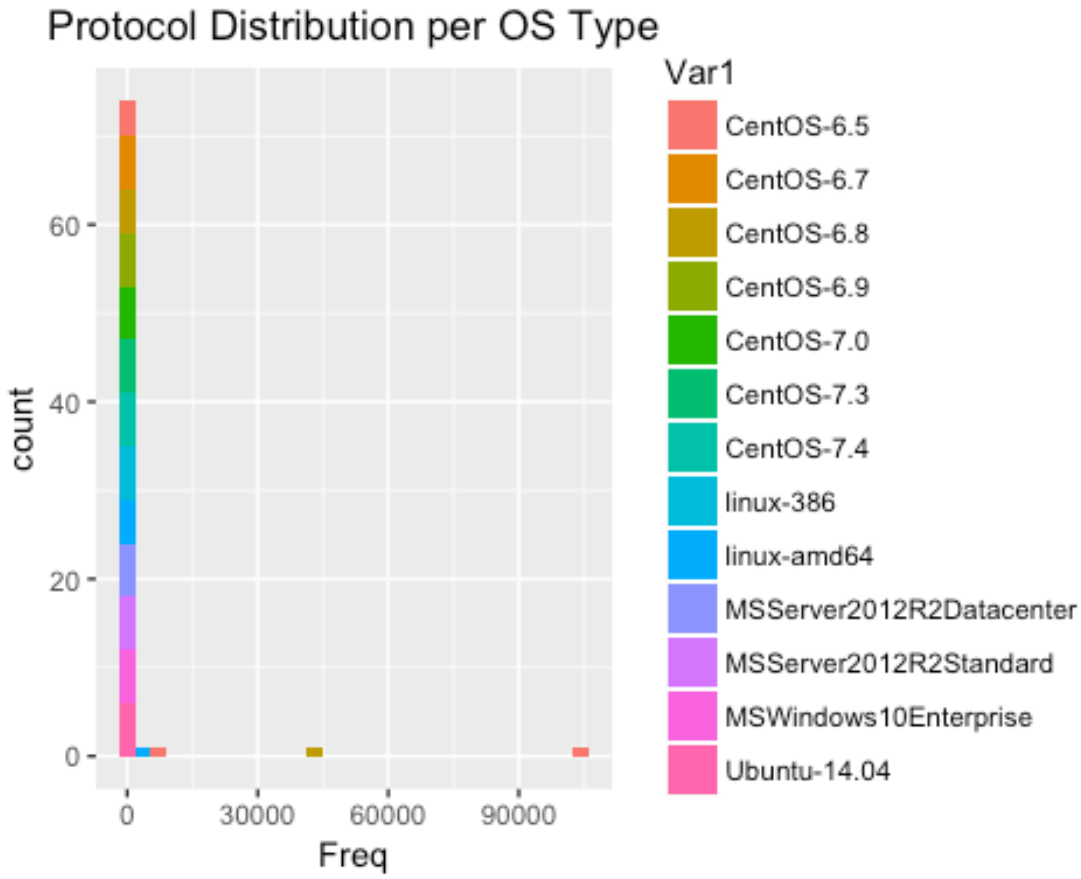


As indicated in the table above, majority of the OS distributions commincate throught the interval of time with a couple of distriburtions only utilizing only network resources on accasion.

Protcoal usage by Platform

We have noted above how specifc guest OS's utilize the network when considering the overall data collection. Now with R table functionality we can quickly evlatuate the overall frequecny of communications protocols per guest OS.

```
myTable <- table(data$platform, data$proto)

#kable(myTable, caption = "Protocol Distribution per OS Type")

myFrame <- as.data.frame((myTable))
myFrame %>% ggplot() +
  geom_histogram(aes(x=Freq, fill=Var1)) +
  ggtitle(paste("Protocol Distribution per OS Type")) +
  theme(plot.title = element_text(hjust = 0.5))
```



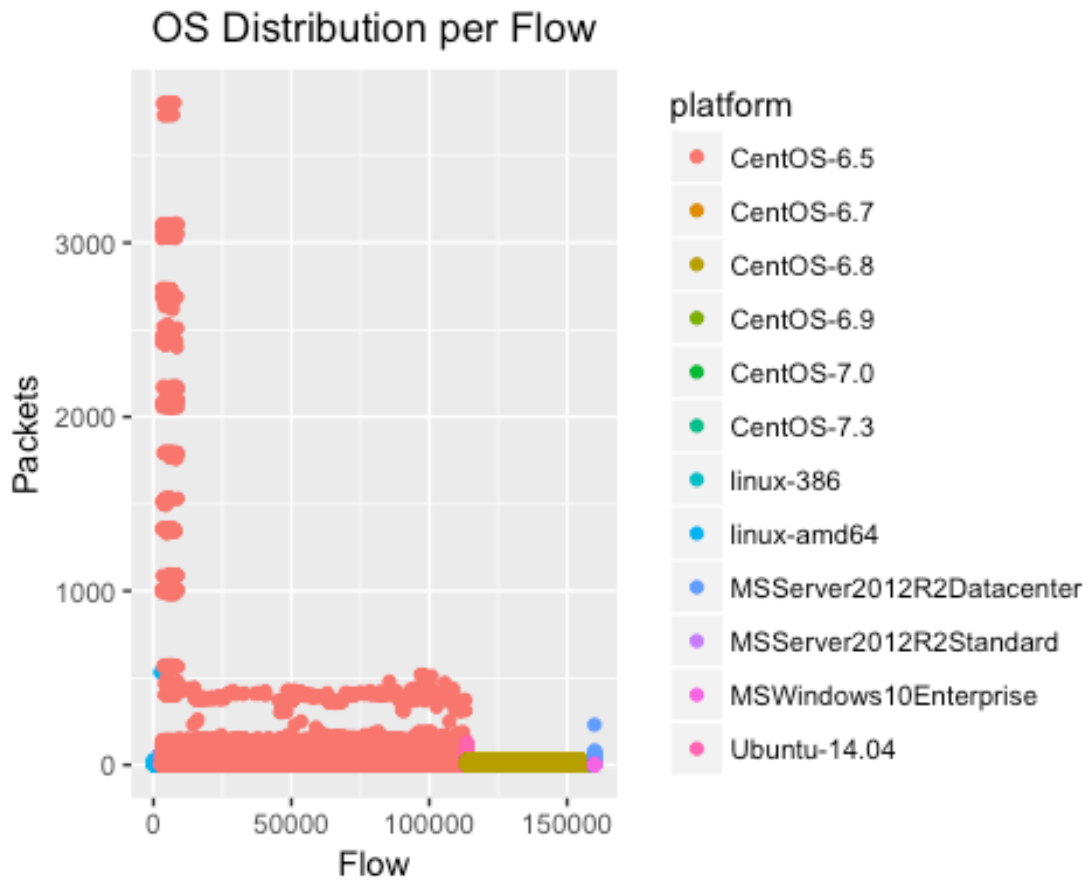
As indicated in the chart above, there is a high frequency of flows across majority of the distributions. There appears to be a few anomolies that indicate high network usage for OS distributions, CentOS-6.5 and CentOS-6.8.

Packect distribution over flow period

As noted in the previous section, we can see that CentOS makes up a majority of the traffic. Now that we have identified the top users of the network by OS distribution, let's now understand during the capture interval where the traffic was consumsed.

To accomplish this task, we will first evaluate the traffic distirbution accross the entire interval, all data flows.

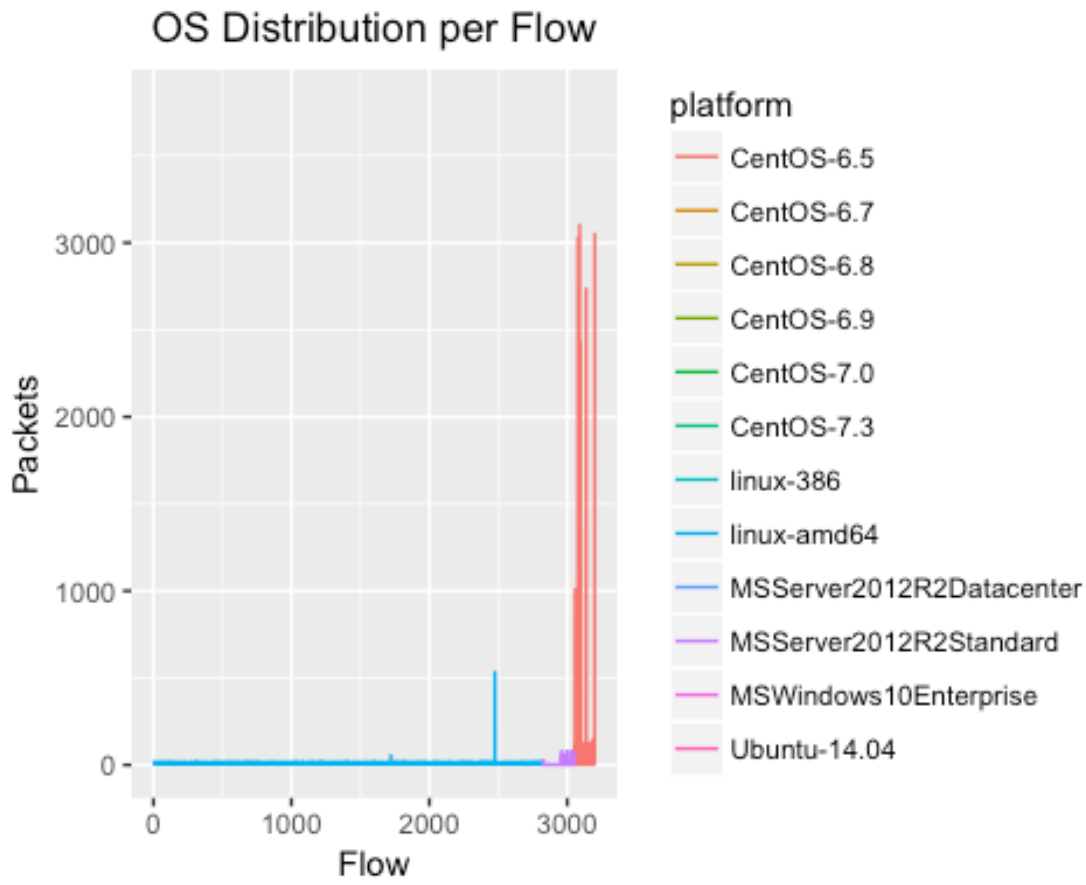
```
# describes the traffic usage per OS type based on total_pkts per capture
ggplot(data, aes(x = as.numeric(row.names(data)), total_pkts, color =
platform)) +
  geom_point() +
  ggtitle(paste("OS Distribution per Flow")) +
  theme(plot.title = element_text(hjust = 0.5)) +
  ylab("Packets") +
  xlab("Flow")
```



The output shows that during the initial part of the data colletion there is a spike in usage by a set of distributions and over time the activity level switches from one platfrom to another.

To understand the uptic in usage as indicated above, lets 'zoom' in.

```
# describes the traffic usage per OS type based on total_pkts per capture
ZOOMed in
ggplot(data, aes(x = as.numeric(row.names(data)), total_pkts, color =
platform)) +
  geom_line() +
  xlim(0,3200) +
  ggtitle(paste("OS Distribution per Flow")) +
  theme(plot.title = element_text(hjust = 0.5)) +
  ylab("Packets") +
  xlab("Flow")
```



As we 'zoom' in, we can clearly see the sequence and number of packets associated with the uptic in traffic. Based on the output, we can begin to understand the level of chattiness of the appplication.

Analysing the data

Up until this point, we have used the data sets to gleen data and begin to understand usage across the network. Now we will use data analysis techniques to help understand and predict flow behavior based on guest OS type, platform.

Data Analysis with Regression

Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. Logistic regression is a special case of linear regression when the outcome variable is categorical.

This method is used to predict the probability of occurrence of an event by fitting data to a logit function. in this case the event is based on platform type.

```
#Logistic Regrsssion
```

```r
# Generate a TEST AND TRAINING SET
set.seed(100)
split = sample.split(data$platform_type, SplitRatio = 0.55)
dataTrain = subset(data, split == TRUE)
#nrow(dataTrain)
dataTest = subset(data, split == FALSE)
#nrow(dataTest)

# Determine the total for each field of the dependent variable
#table(dataTrain$platform_type)

# Update the dependent variable field to convert it to a vector
data$platform_type <- as.factor(data$platform_type)

# Logistic Regression
platformLog = glm(platform_type ~ proto + total_pkts, data = dataTrain,
family = binomial)
summary(platformLog)
```

```
##
## Call:
## glm(formula = platform_type ~ proto + total_pkts, family = binomial,
##     data = dataTrain)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.6802   0.0195   0.0378   0.0418   1.8930
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)       2.227004   0.148657  14.981  < 2e-16 ***
## protoARP_REQUEST  0.609847   0.236678   2.577 0.009975 **
## protoICMP        -3.887210   1.105447  -3.516 0.000437 ***
## protoTCP          4.663631   0.213033  21.892  < 2e-16 ***
## protoUDP          0.148689   0.194340   0.765 0.444213
## total_pkts        0.050768   0.006653   7.631 2.33e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3123.7  on 88026  degrees of freedom
## Residual deviance: 2063.0  on 88021  degrees of freedom
## AIC: 2075
##
## Number of Fisher Scoring iterations: 12
```

```r
# Confidence Interval
confint(platformLog)
```

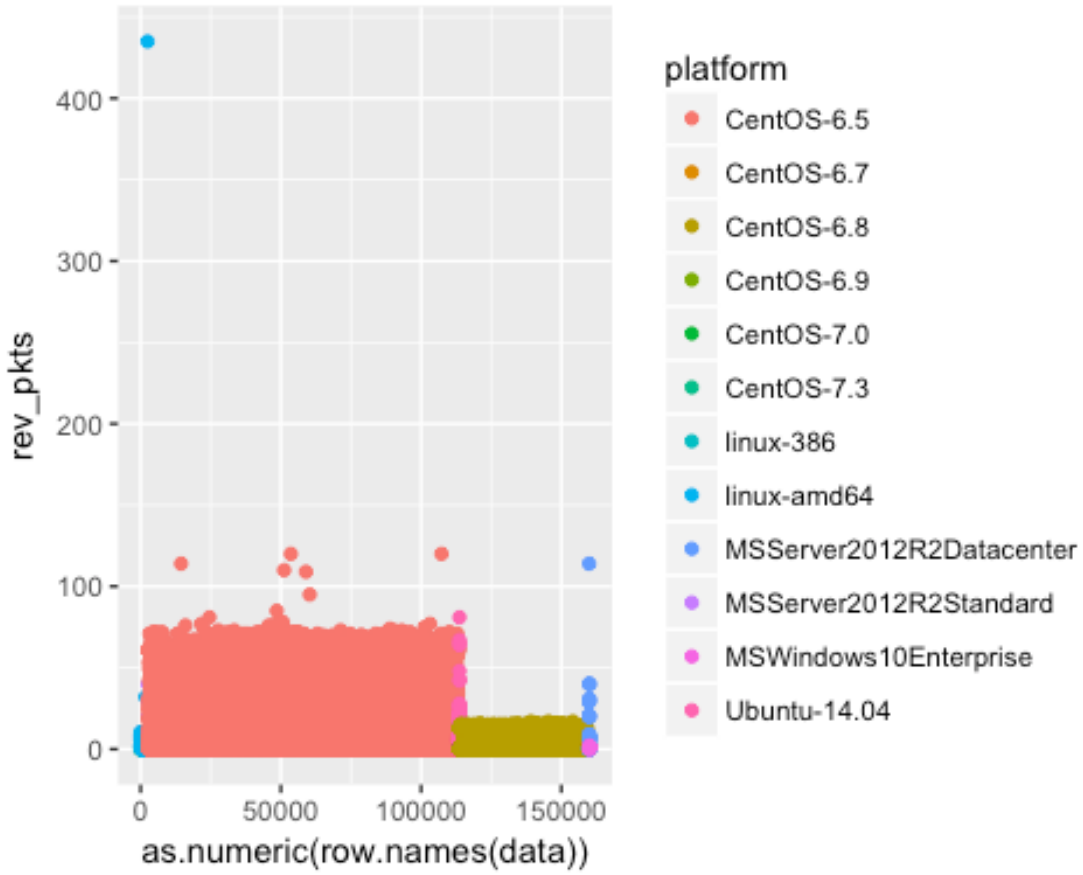
```
##                       2.5 %      97.5 %
## (Intercept)       1.9465091  2.53048821
## protoARP_REQUEST  0.1516072  1.08286176
## protoICMP        -6.8504937 -2.03757385
## protoTCP          4.2462729  5.08380967
## protoUDP         -0.2369815  0.52635869
## total_pkts        0.0386604  0.06487838

#exp(coef(platformLog))
#exp(cbind(OR = coef(platformLog), confint(platformLog)))

# Prediction
predictTrain = predict(platformLog, type="response")
summary(predictTrain)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1667  0.9991  0.9993  0.9975  0.9998  1.0000

# Display the mean of the Training set
tapply(predictTrain, dataTrain$platform_type, mean)

##         0         1
## 0.9219886 0.9976479

# 1 is Windows Platform
# 0 is Linux Platform
```

From the output we can see that:

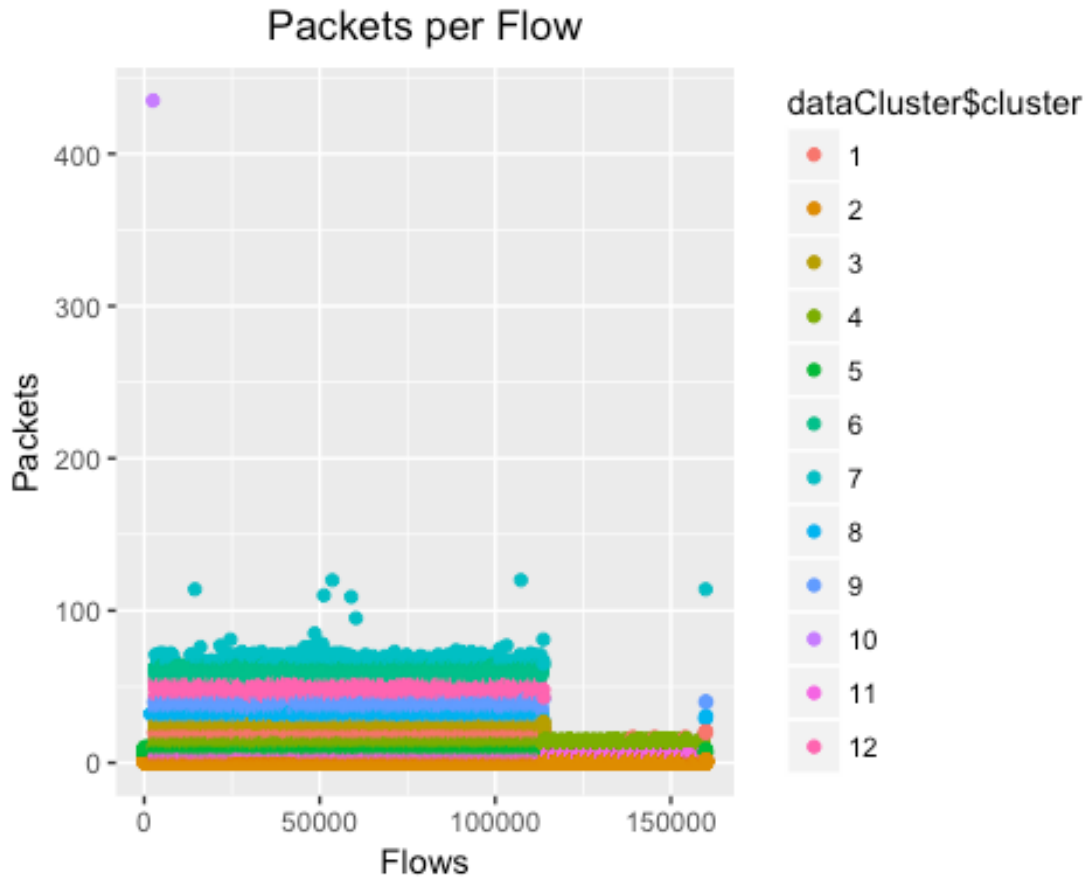
Data Analysis with K Means Clustering

K Means Clustering is an unsupervised learning algorithm that tries to cluster data based on their similarity. Unsupervised learning means that there is no outcome to be predicted, and the algorithm just tries to find patterns in the data. In k means clustering, we have to specify the number of clusters we want the data to be grouped into. The algorithm randomly assigns each observation to a cluster, and finds the centroid of each cluster.

```
# K Means Clustering in R

#explore data
#head(data)

#plot data (before)
ggplot(data, aes(as.numeric(row.names(data)), rev_pkts, color = platform)) +
  geom_point()
```




```r
#clustering data
set.seed(20)
dataCluster <- kmeans(data[, 11], 12, nstart = 20)
#dataCluster

#compare cluster output with platform
#table(dataCluster$cluster, data$platform)

#plot data (after)
dataCluster$cluster <- as.factor(dataCluster$cluster)
ggplot(data, aes(as.numeric(row.names(data)), rev_pkts, color =
dataCluster$cluster)) +
  geom_point() +
  ggtitle(paste("Packets per Flow")) +
  theme(plot.title = element_text(hjust = 0.5)) +
  ylab("Packets") +
  xlab("Flows")
```

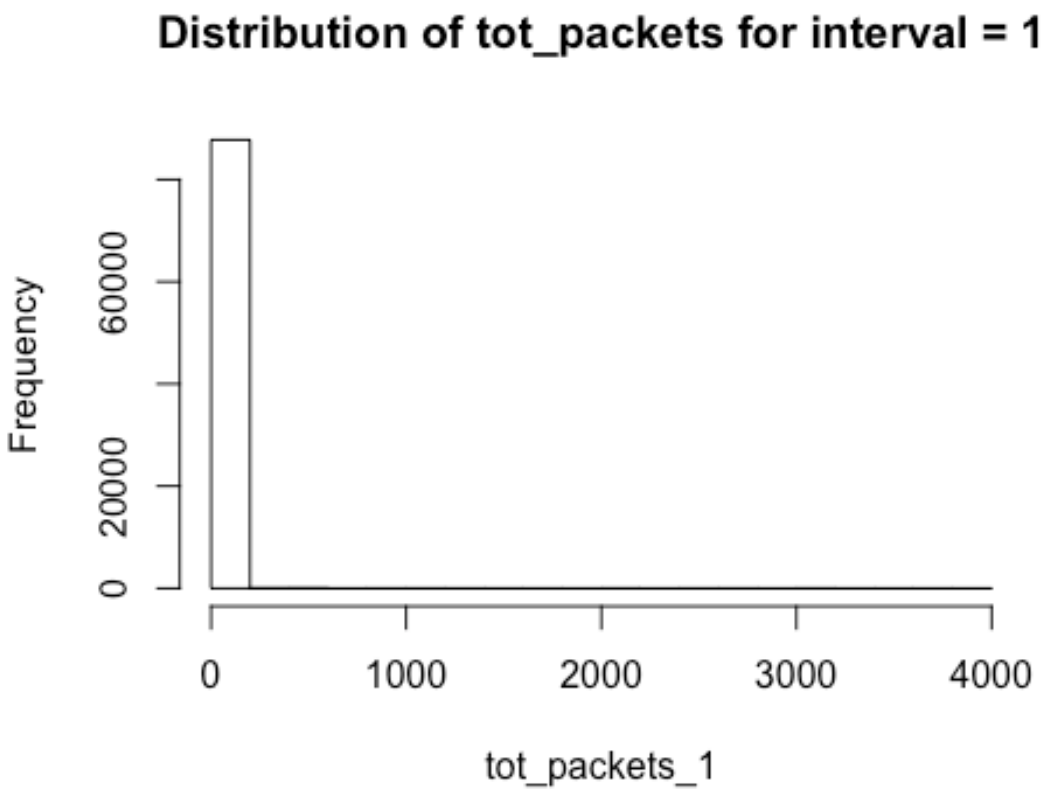


From the output we can see that:

Data Analysis with Box-Cox

Transforming data with a box-cox regression set.

```
set.seed(100)
split = sample.split(data$platform_type, SplitRatio = 0.55)
df_1 = subset(data, split == TRUE)
#nrow(tot_packets_1)
df_2 = subset(data, split == FALSE)
#nrow(tot_packets_2)

tot_packets_1 = df_1$total_pkts
tot_packets_2 = df_2$total_pkts

hist(tot_packets_1, main="Distribution of tot_packets for interval = 1")
```




```r
# how to check it automatically : Shapiro Test
sample1 = sample(tot_packets_1, min(5000, length(tot_packets_1)))

# shapiro test fails if all vector elements ar ethe same
test = shapiro.test(sample1)
if (test$p.value < 0.05) {
  # Data not Normal. Perform log-transform
  cat(sprintf("Data for %s are NOT normal\n", "sample1"))
} else {
  # Data not Normal. Perform log-transform
  cat(sprintf("Data for %s are normal\n", "sample1"))
}

## Data for sample1 are NOT normal

# In case if data NOT normal there are two options
# - perform NON-Parametric test : Wilcoxon-Mann-Whitney test(when we dont
know the parameters of the distribution)
#we assume (called Null-hypothesis, Ho) that two samples are identical (or
similar).
# if p_value < 0.05 - we REJECT Null-hypothesis and accept Alternative
test = wilcox.test(tot_packets_1, tot_packets_2)
if (test$p.value < 0.05) {
```

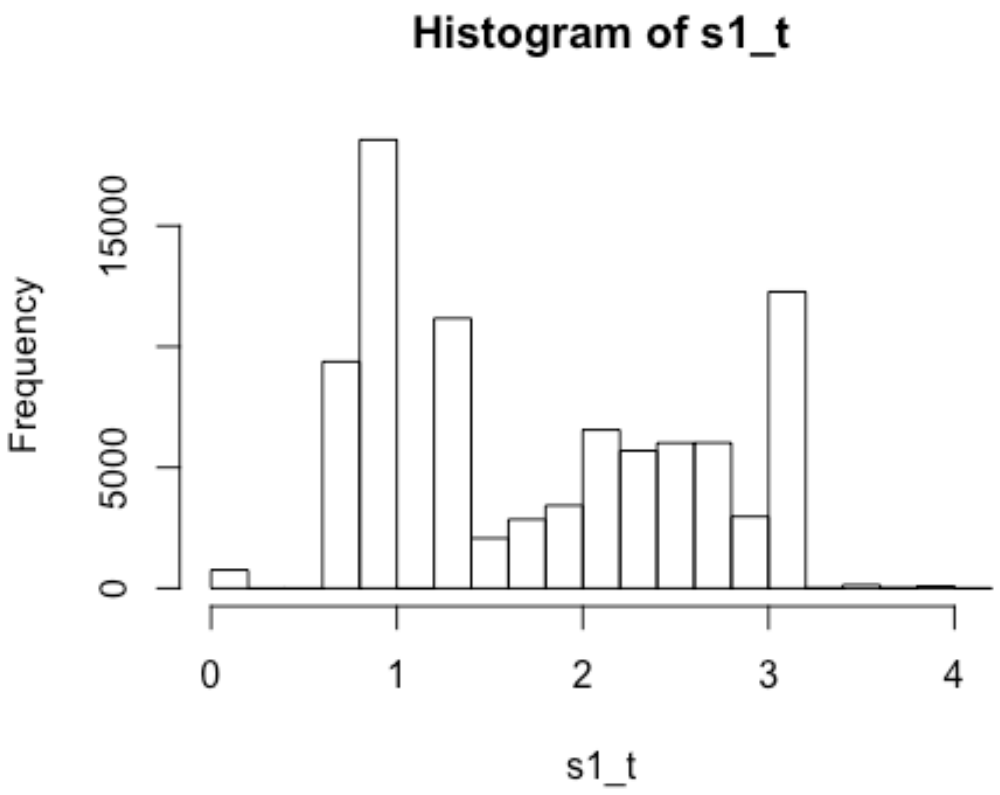
```
  print("Two samples are NOT identical")
} else {
  print("Two samples are identical")
}
```

```
## [1] "Two samples are identical"
```

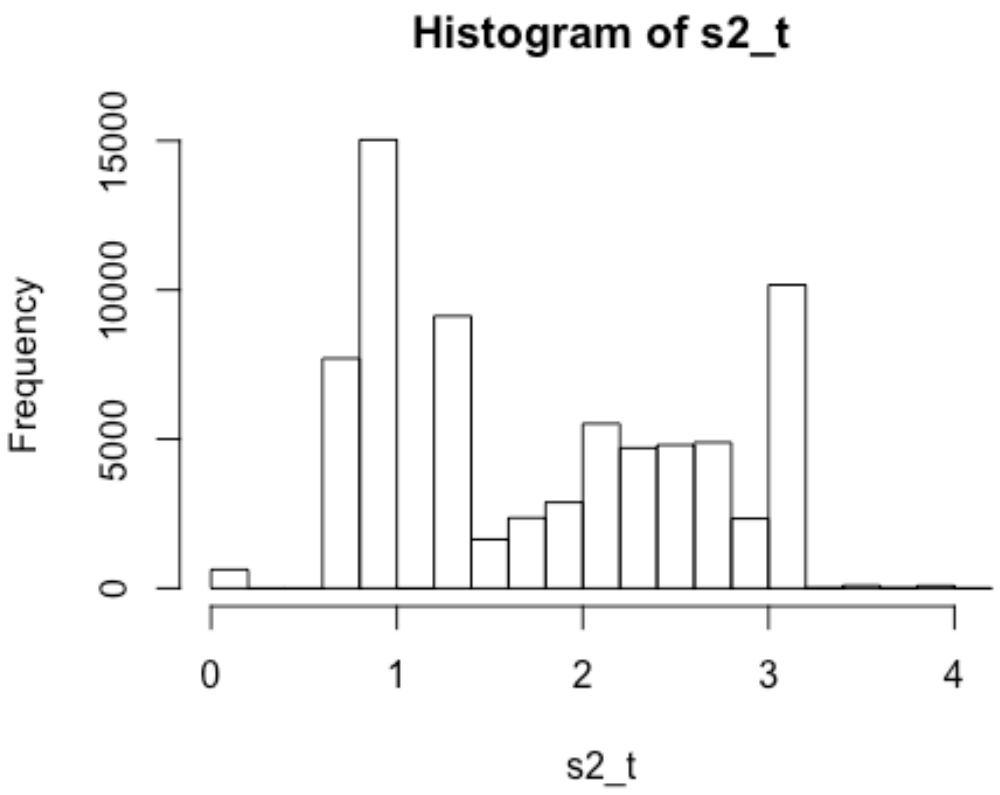
```
# How it looks like when THEY ARE IDENTICAL ?
# we take samples from the SAME data set
test = wilcox.test(sample(tot_packets_1, 100), sample(tot_packets_1,100))
if (test$p.value < 0.05) {
  print("Two samples are NOT identical")
} else {
  print("Two samples are identical or very similar")
}
```

```
## [1] "Two samples are identical or very similar"
```

```
# - transform data so that they become more or less normal
# it is called Box-Cox transformation :
https://en.wikipedia.org/wiki/Power_transform (see Box Cox trans)

# here's implementation of BoxCox transformation
box_cox <- function(v) {
  ob = MASS::boxcox(v ~ 1, plotit = F)
  lambda = ob$x[which.max(ob$y)]

  if (lambda == 0) {
    t_v = log(v) # lambda CAN be zero.
  } else {
    t_v = (v ^ lambda - 1) / lambda
  }
  return(t_v)
}

# transform data
s1_t = box_cox(tot_packets_1)
s2_t = box_cox(tot_packets_2)

hist(s1_t)
```




```r
hist(s2_t)
```




```
# Null hypothesis Ho is : tow samples are equal
t.test(s1_t, s2_t)

##
##  Welch Two Sample t-test
##
## data:  s1_t and s2_t
## t = -0.46154, df = 153810, p-value = 0.6444
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.010546209  0.006526029
## sample estimates:
## mean of x mean of y
##  1.795077  1.797087
```

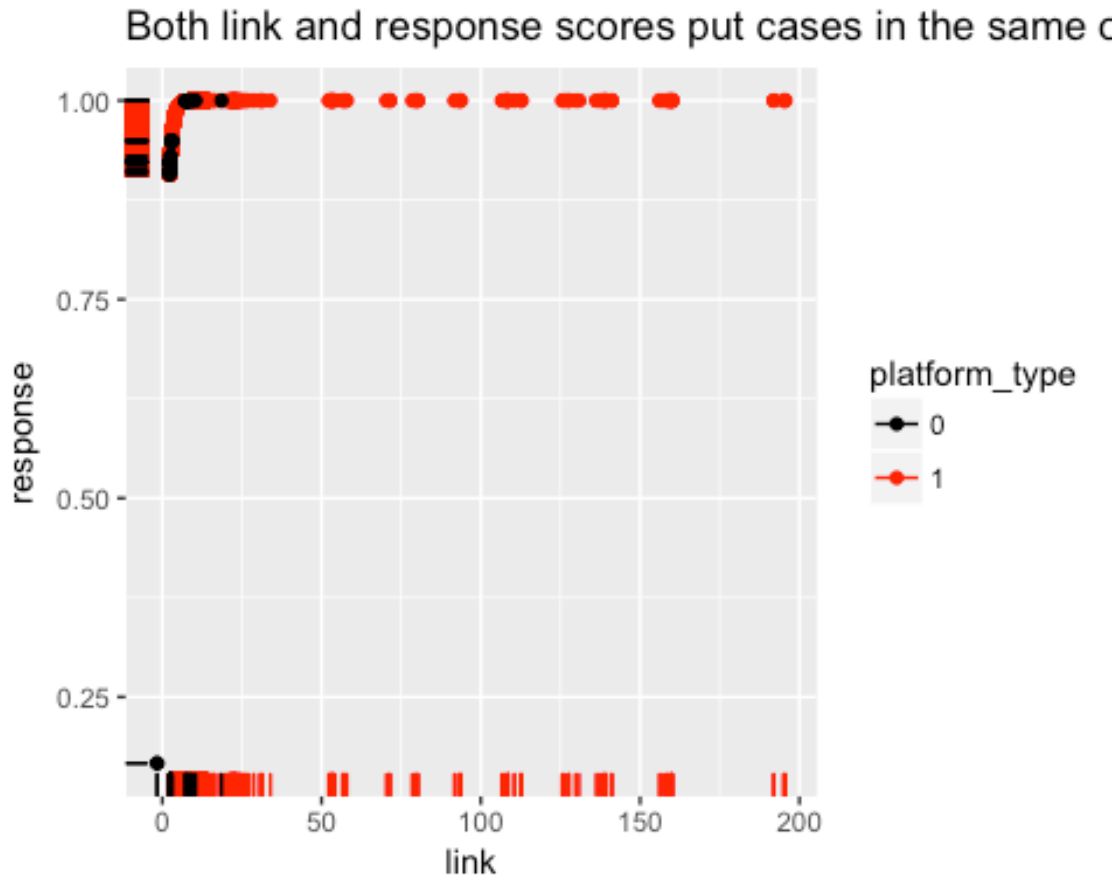
From the output we can see that:

Data Analysis with a ROC Curve

ROC curves are commonly used to characterize the sensitivity/specificity tradeoffs for a binary classifier. Most machine learning classifiers produce real-valued scores that correspond with the strength of the prediction that a given case is positive. Turning these real-valued scores into yes or no predictions requires setting a threshold; cases with scores

above the threshold are classified as positive, and cases with scores below the threshold are predicted to be negative.

```
# Generate a TEST AND TRAINING SET
set.seed(100)
split = sample.split(data$platform_type, SplitRatio = 0.55)
training_set = subset(data, split == TRUE)
test_set = subset(data, split == FALSE)

# Logistic Regression
fit_glm = glm(platform_type ~ proto + total_pkts, data = training_set, family
= binomial)

glm_link_scores <- predict(fit_glm, test_set, type="link")

glm_response_scores <- predict(fit_glm, test_set, type="response")

score_data <- data.frame(link=glm_link_scores,
                         response=glm_response_scores,
                         platform_type=test_set$platform_type,
                         stringsAsFactors=FALSE)

score_data %>% ggplot(aes(x=link, y=response, col=platform_type)) +
  scale_color_manual(values=c("black", "red")) +
  geom_point() +
  geom_rug() +
  ggtitle("Both link and response scores put cases in the same order")
```



From the output we can see that: