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.

```r
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, data/SensorData.csv.

There are a total of 5225 ROWs and 35 COLUMNs in the combined flow data set. The 5225 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.

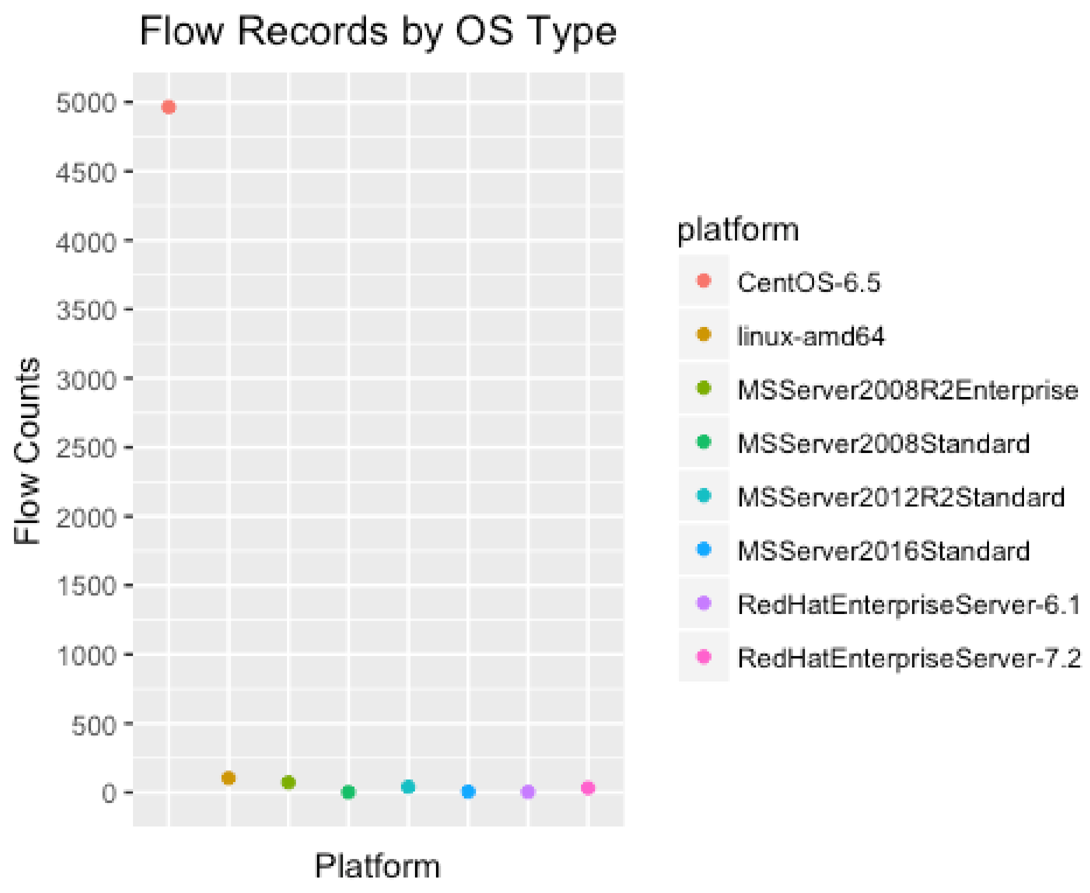
- Count the number of rows in the flow data set per platform type (the data set dependent variable).

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 120 flows attributed to Windows based hosts and 5105 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"))
```



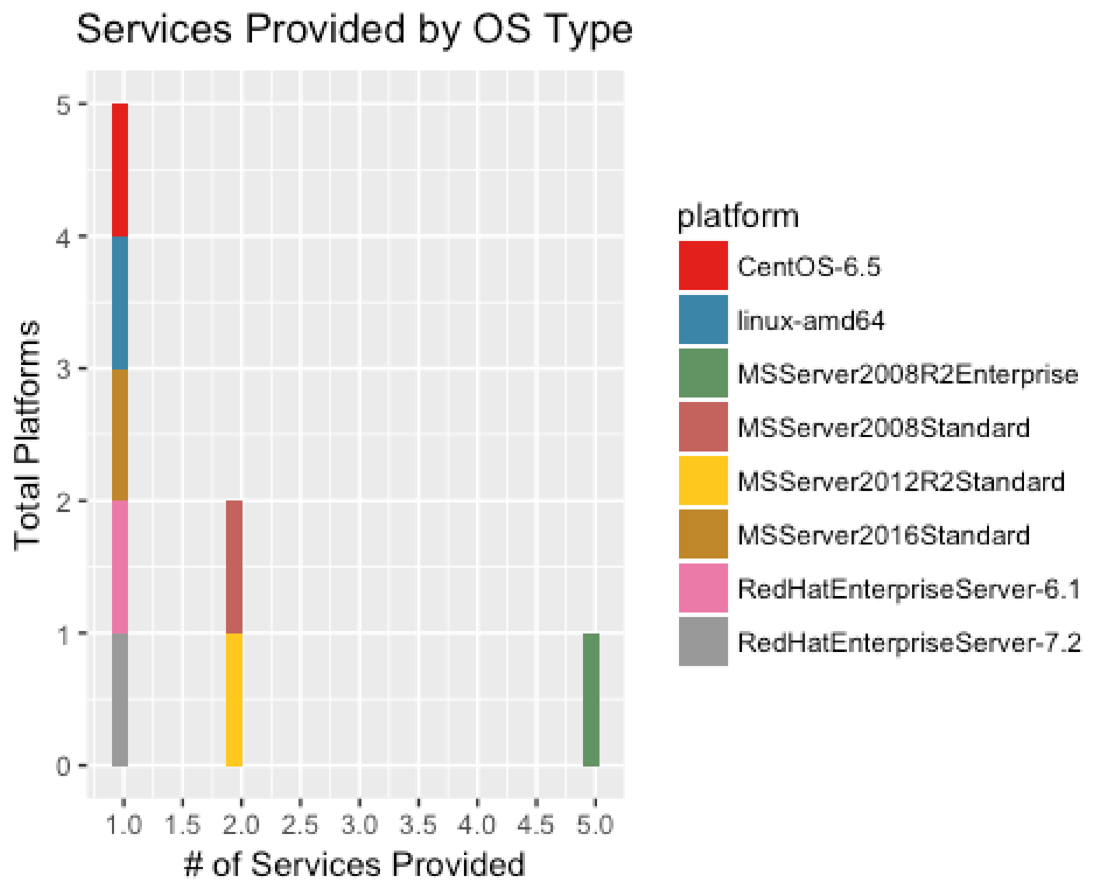
```
#data$timestamp <- factor(data$timestamp, levels=unique(data$timestamp))
```

As indicated in the table above, CentOS-6.5 is the most frequently utilized OS 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 number services provided by the guest OS type. 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 host bhavior 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))
```

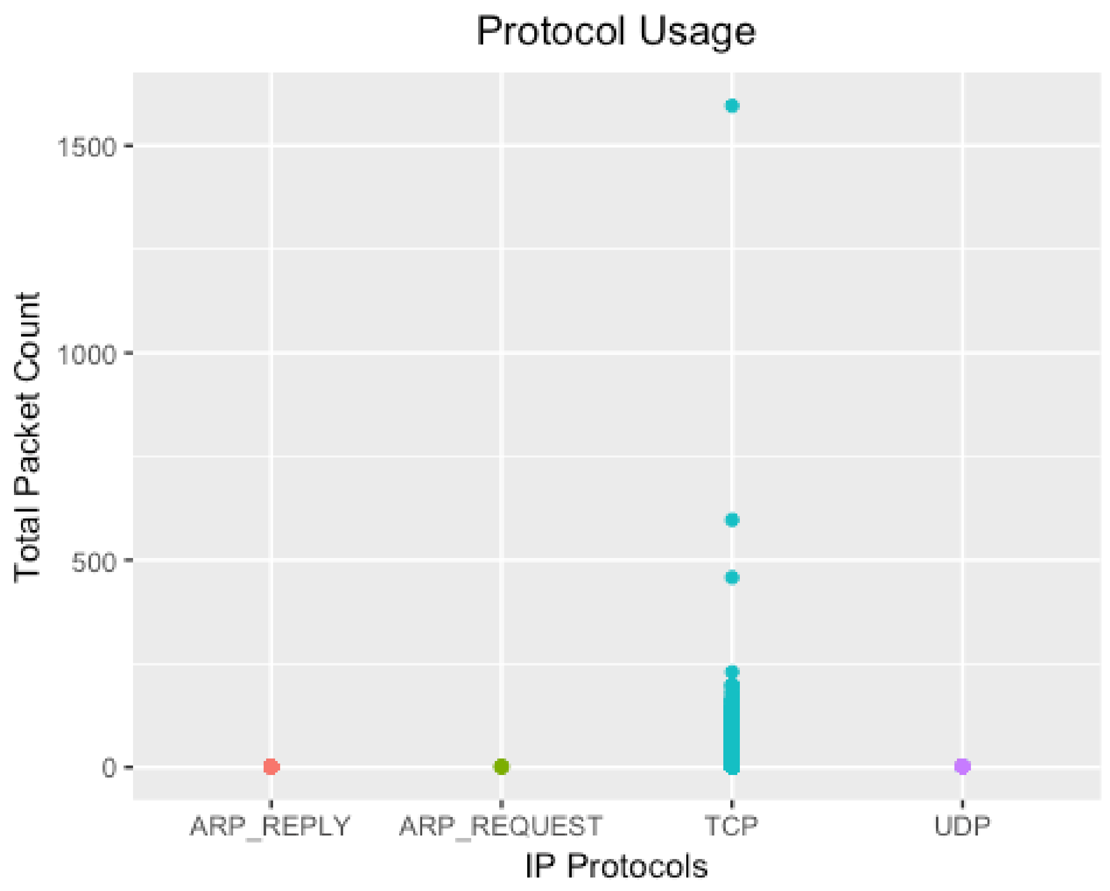


As indicated in the histogram above, we can determine that majority of the guest OS's are providing a single service. This can be interpreteded as the majority of the OS's analyzed within the flow analysis are providing serivce to a single application.

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"))
```

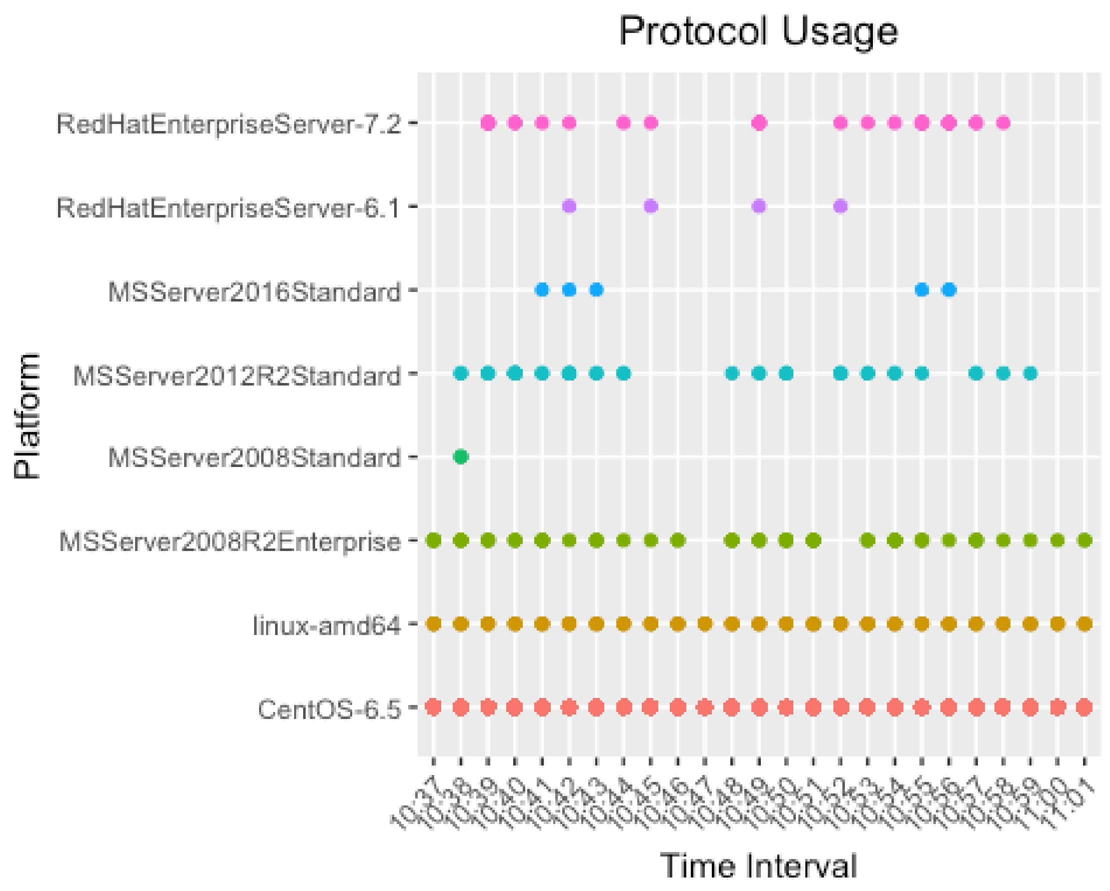


From the table above we can infer that TCP is predominately utilized via the host applications analysed during the capture period. The range of packets part of the TCP flows extend from 1 to 1500+ packets per flow.

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"))
```

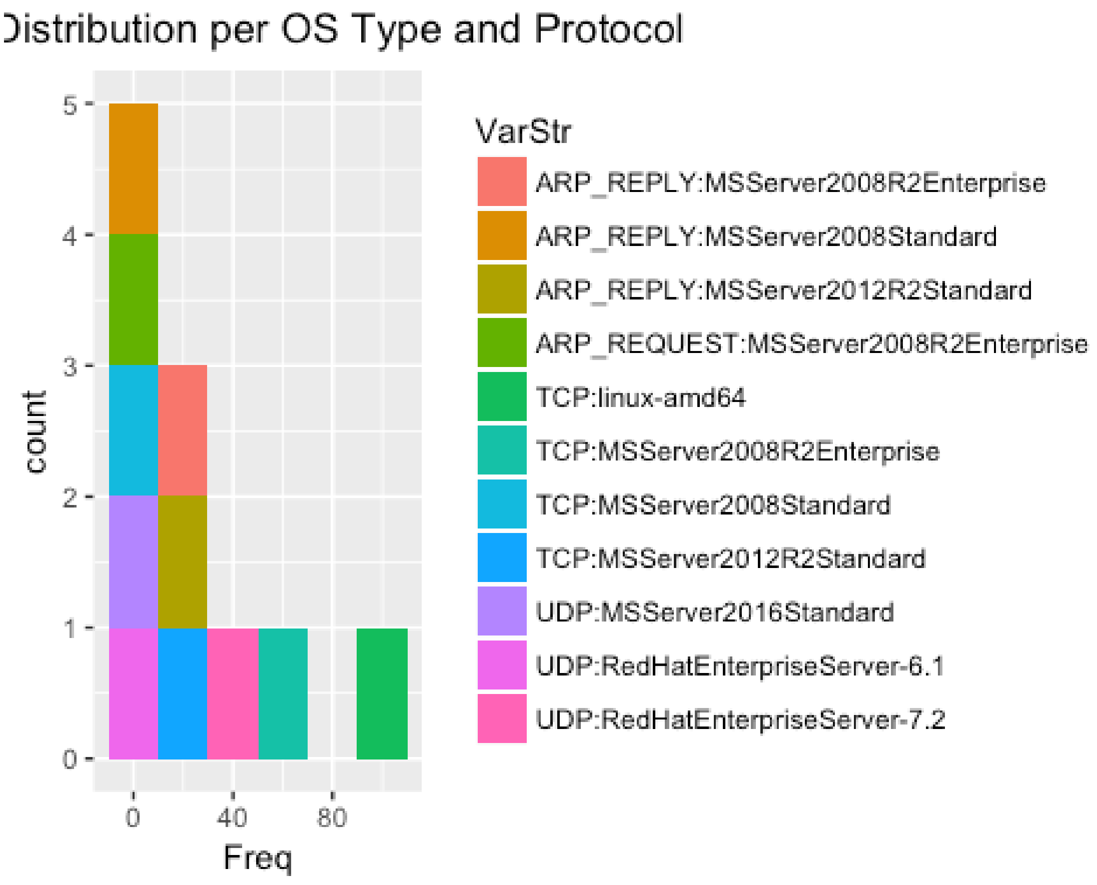


By graphing the usage of the infrastrcutre per host OS accross the collected time intervals, we can determine that majority of the OS distributions commincate throught the intireity of the interval with a couple of distriburtions only utilizing network resources on accasion.

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))

myFrame1 <- myFrame[apply(myFrame!=0, 1, all),]
myFrame1_sub <- subset(myFrame1, Freq<150)

myFrame1_sub$VarStr <- paste(myFrame1_sub$Var2, myFrame1_sub$Var1, sep=':')

myFrame1_sub %>% ggplot() +
  geom_histogram(aes(x=Freq, fill=VarStr), binwidth = 20) +
  ggtitle(paste("Protocol Distribution per OS Type and Protocol")) +
  theme(plot.title = element_text(hjust = 0.5))
```

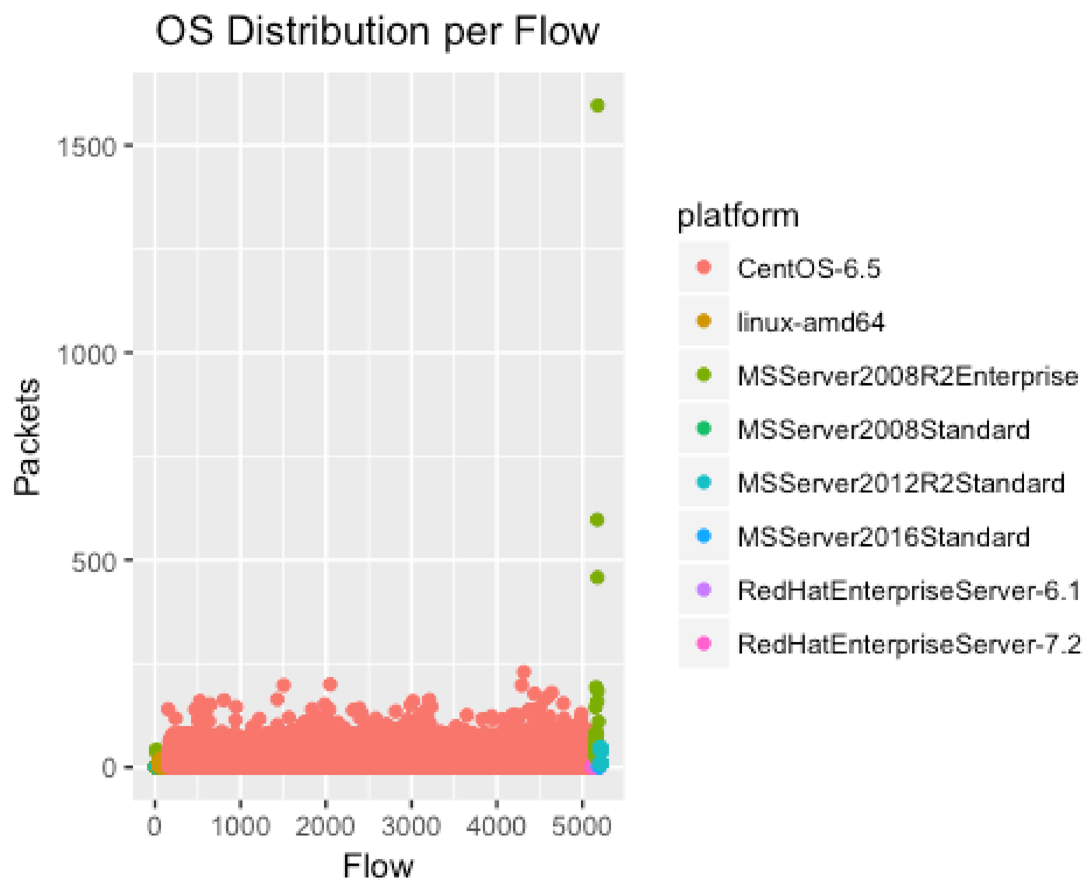


Packect distribution over flow period

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")
```



When exploring the total count of packets per OS type accross the capture, we are able to identify that CentOS-6.5 is responsible for majority of the flows captured. The volumen of CentOS-6.5 flows far out weights the all other OS distributions combined.

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
```

```
# 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

# 1 is Windows Platform
# 0 is Linux Platform

#table(dataTrain$platform_type)
#    0     1
#  224 87803

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

### Logistic Regression

## Model fitting
Trials = cbind(dataTrain$total_pkts, dataTrain$total_bytes)
model = glm(Trials ~ platform, data = dataTrain, family =
binomial(link="logit"))

## Coefficients and exponentiated cofficients
summary(model)

##
## Call:
## glm(formula = Trials ~ platform, family = binomial(link = "logit"),
##     data = dataTrain)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -10.8901    0.6451    2.0074    2.5457    4.0204
##
## Coefficients:
##                                     Estimate Std. Error   z value Pr(>|z|)
## (Intercept)                        -5.846934   0.004218 -1386.251  < 2e-16
## platformlinux-amd64                 0.740749   0.063201    11.721  < 2e-16
## platformMSServer2008R2Enterprise   -0.816106   0.025837   -31.587  < 2e-16
## platformMSServer2008Standard        0.156574   0.354159     0.442 0.658415
## platformMSServer2012R2Standard      0.407201   0.057068     7.135 9.65e-13
## platformMSServer2016Standard        1.347124   0.355537     3.789 0.000151
## platformRedHatEnterpriseServer-6.1  1.151009   0.710341     1.620 0.105155
## platformRedHatEnterpriseServer-7.2  0.915684   0.162860     5.623 1.88e-08
```

```
##
## (Intercept)                          ***
## platformlinux-amd64                  ***
## platformMSServer2008R2Enterprise     ***
## platformMSServer2008Standard
## platformMSServer2012R2Standard       ***
## platformMSServer2016Standard         ***
## platformRedHatEnterpriseServer-6.1
## platformRedHatEnterpriseServer-7.2 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 15162  on 2873  degrees of freedom
## Residual deviance: 13648  on 2866  degrees of freedom
## AIC: 24960
##
## Number of Fisher Scoring iterations: 4
```

```r
#confint(model)

# exponentiated coefficients
#exp(model$coefficients)

# 95% CI for exponentiated coefficients
#exp(confint(model))

## Analysis of variance for individual terms
library(car)
Anova(model, type="II", test="Wald")
```

```
## Analysis of Deviance Table (Type II tests)
##
## Response: Trials
##          Df  Chisq Pr(>Chisq)
## platform  7 1249.3  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
## Pseudo-R-squared
library(rcompanion)
nagelkerke(model)
```

```
## $Models
##
## Model: "glm, Trials ~ platform, binomial(link = \"logit\"), dataTrain"
## Null:  "glm, Trials ~ 1, binomial(link = \"logit\"), dataTrain"
##
## $Pseudo.R.squared.for.model.vs.null
##                              Pseudo.R.squared
## McFadden                            0.0571987
## Cox and Snell (ML)                  0.4093610
## Nagelkerke (Cragg and Uhler)        0.4094020
##
## $Likelihood.ratio.test
##  Df.diff LogLik.diff  Chisq       p.value
##       -7     -756.65 1513.3 1.185758e-322
##
## $Number.of.observations
##
## Model: 2874
## Null:  2874
##
## $Messages
## [1] "Note: For models fit with REML, these statistics are based on
refitting with ML"
##
## $Warnings
## [1] "None"

## Overall p-value for model

# update here produces null model for comparison
anova(model, update(model, ~1), test="Chisq")

## Analysis of Deviance Table
##
## Model 1: Trials ~ platform
## Model 2: Trials ~ 1
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1      2866      13648
## 2      2873      15162 -7  -1513.3 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

library(lmtest)
lrtest(model)

## Likelihood ratio test
##
## Model 1: Trials ~ platform
## Model 2: Trials ~ 1
##   #Df LogLik Df  Chisq Pr(>Chisq)
## 1   8 -12472
## 2   1 -13228 -7 1513.3  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

## Plot of standardized residuals
#plot(fitted(model), rstandard(model))
```

```
#Plotting the model
#plot(total_pkts ~ platform, data = dataTrain,
#     xlab="Platform",
#     ylab="Total Packets",
#     pch=19)
```

The result of the regression model identifes that the insignificant values in the model are limited to REHL-6.1 and MS2008Standard.

The distribution of the deviance residuals for individual cases used in the model range from -3.8272 to .3789. The model output shows how dst_pot and total_pkts are statistically signifcant while total_bytes is not and can be removed from the model. As a result, for every one unit change in total_pkts, the log odds of the platform type windows (versus linux) increases by 0.006.

Now we can say that for a one unit increase in total_pkts, the odds of being a windows source flow (versus linux sourced flow) increase by a factor of 1.07

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.

```
# http://thestatsgeek.com/2014/02/08/r-squared-in-logistic-regression/

mod <- glm(platform_type ~ proto + total_pkts, data = dataTrain, family =
binomial)
#mod <-glm(platform_type ~ total_pkts + total_bytes, data = dataTrain, family
= binomial)
#nullmod <- glm(platform_type ~ 1, data = training_set, family="binomial")
#(1-logLik(mod)/logLik(nullmod))

library(rcompanion)
nagelkerke(mod)

## $Models
##
## Model: "glm, platform_type ~ proto + total_pkts, binomial, dataTrain"
## Null:  "glm, platform_type ~ 1, binomial, dataTrain"
##
## $Pseudo.R.squared.for.model.vs.null
##                              Pseudo.R.squared
## McFadden                            0.3139560
## Cox and Snell (ML)                  0.0663651
## Nagelkerke (Cragg and Uhler)        0.3378100
```

```
##
## $Likelihood.ratio.test
##  Df.diff LogLik.diff  Chisq    p.value
##       -4     -98.679 197.36 1.3901e-41
##
## $Number.of.observations
##
## Model: 2874
## Null:  2874
##
## $Messages
## [1] "Note: For models fit with REML, these statistics are based on
refitting with ML"
##
## $Warnings
## [1] "None"
```

```
#library("broom")
#(rsq_glance <- glance(mod)$r.squared)
```