Social Unrest and TCP Congestion

Kinga Farkas

Monday, March 23, 2015

Introduction

The aim of this project is to answer the question: "Did the social unrest in Egypt in January - February, 2011 coincide with significant increase in RTT, indicating TCP congestion?"

Getting the data

The General Method

I have experimented with using the "bigrquery" package in R that provides a direct interface with Google's big query. However, I ran into problems right away : the data I read in using the interface and the data I got by using the Google Big Query Web UI did not match up. It seems that there are issues with using the "bigrquery" interface for integers larger than 32 bit. Integers larger than 2^{32} were coerced not into NA's but rather into other integer values. Eventually I settled on using Google Big Query Web UI for accessing the M-lab data. However, since R does not deal well with integers larger than 2^{32}, I still had to examine the data to look for the variables that could potentially take on values that are too large to be imported into R as integers. The schema of the M-Lab dataset that I found on https://cloud.google.com/bigquery/docs/dataset-mlab, helped, but I needed further information on the Web100 variables.
I ended up writing a script that scours the tcp-kis.txt file (https://cloud.google.com/bigquery/docs/tcp-kis.txt) for the variable names and types, then it stores this information in a data frame. This data frame can now be used as a Web100 variables look-up table.

```
#Reading in the file downloaded from
https://cloud.google.com/bigquery/docs/tcp-kis.txt:
text_file = readLines("tcp-kis.txt")

#finding the variable names within the .txt file:
var_names <- text_file[grep("VariableName", text_file)]
get_var_name <- function(x){
        skip <- nchar("VariableName:\t")+1
        x_length <- nchar(x)
        var_name <- substr(x, skip, x_length)
        return(var_name)
}
var_names <- unlist(lapply(var_names, get_var_name))
```

```r
#finding the types of the variables within the .txt file
var_types <- text_file[grep("ProcType:", text_file)]
get_var_type <- function(x){
        skip <- nchar("Proctype:\t")+1
        x_length <- nchar(x)
        var_type <- substr(x, skip, x_length)
        return(var_type)
}
var_types <- unlist(lapply(var_types, get_var_type))


#creating a dataframe containing names and corresponing types of the
variables
web100_df <- data.frame(var_names, var_types)

#renaming the columns of the data frame
names(web100_df)<-c("variableName", "variableType")

#finding the BigQuery types of the variables
integer_types <- c("Integer32", "Integer", "INTEGER", "Gauge32",
"ZeroBasedCounter32",
                   "Unsigned32", "Unsigned16", "Counter32",
"ZeroBasedCounter64")
big_query_typer <- function(x){
        if (x %in% integer_types){
                return("integer")
        }
        else if (x == "Ip_Address"){
                return("string")
        }
        else {return("boolean")}
}

bq_types <- unlist(lapply(web100_df$variableType, big_query_typer))

#adding a third column containing the appropriate BigQuery type of each
variable
web100_df$bigQueryType = bq_types

#adding a fourth column that will represent the type the variable should have
in R
r_typer <- function(x){
        if (x %in% integer_types){
                return("numeric")
        }
        else if (x == "Ip_Address"){
                return("character")
        }
        else {return("logical")}
}
types_for_R <- unlist(lapply(web100_df$variableType, r_typer))

web100_df$typesForR = types_for_R

head(web100_df,10)
```

```
##        variableName variableType bigQueryType typesForR
## 1      LocalAddress   Ip_Address       string character
## 2  LocalAddressType      Integer      integer   numeric
## 3         LocalPort   Unsigned16      integer   numeric
## 4        RemAddress   Ip_Address       string character
## 5           RemPort   Unsigned16      integer   numeric
## 6           WAD_IFQ      Gauge32      integer   numeric
## 7      WAD_MaxBurst      Gauge32      integer   numeric
## 8   WAD_MaxSsthresh      Gauge32      integer   numeric
## 9          WAD_NoAI      Integer      integer   numeric
## 10   WAD_CwndAdjust    Integer32      integer   numeric
```

BigQuery Specifics

I used the M-Lab's PDEChartsNDT wiki page (https://code.google.com/p/m-lab/wiki/PDEChartsNDT) as a refernce when creating the BigQuery queries of the M-Lab dataset throught the BigQuery Web UI. I wanted to look at the Network Diagnostic Tool's RTT measurements in Egypt around the time of the Arab Spring, which began in January of 2011. I filtered out the incomplete test results and at the same time I included most of the variables in my query. This is what the form of the general query looks like:

SELECT * , connection_spec.client_geolocation.continent_code, connection_spec.client_geolocation.country_name,connection_spec.client_geolocation.city,

connection_spec.server_geolocation.continent_code, connection_spec.server_geolocation.country_name,

connection_spec.server_geolocation.city

FROM [measurement-lab:m_lab.year_month]

WHERE connection_spec.client_geolocation.country_name == "Egypt"

AND IS_EXPLICITLY_DEFINED(web100_log_entry.connection_spec.remote_ip)

AND IS_EXPLICITLY_DEFINED(web100_log_entry.connection_spec.local_ip)

AND IS_EXPLICITLY_DEFINED(web100_log_entry.snap.HCThruOctetsAcked)

AND IS_EXPLICITLY_DEFINED(web100_log_entry.snap.SndLimTimeRwin)

AND IS_EXPLICITLY_DEFINED(web100_log_entry.snap.SndLimTimeCwnd)

AND IS_EXPLICITLY_DEFINED(web100_log_entry.snap.SndLimTimeSnd)

```
AND IS_EXPLICITLY_DEFINED(project)
AND project = 0
AND IS_EXPLICITLY_DEFINED(connection_spec.data_direction)
AND connection_spec.data_direction = 1
AND IS_EXPLICITLY_DEFINED(web100_log_entry.is_last_entry)
AND web100_log_entry.is_last_entry = True
AND web100_log_entry.snap.HCThruOctetsAcked >= 8192
AND (web100_log_entry.snap.SndLimTimeRwin +
web100_log_entry.snap.SndLimTimeCwnd +
web100_log_entry.snap.SndLimTimeSnd) >= 9000000
AND (web100_log_entry.snap.SndLimTimeRwin +
web100_log_entry.snap.SndLimTimeCwnd +
web100_log_entry.snap.SndLimTimeSnd) < 3600000000
AND IS_EXPLICITLY_DEFINED(web100_log_entry.snap.MinRTT)
AND IS_EXPLICITLY_DEFINED(web100_log_entry.snap.CountRTT)
AND web100_log_entry.snap.CountRTT > 0
AND (web100_log_entry.snap.State == 1 OR (web100_log_entry.snap.State >= 5
AND web100_log_entry.snap.State <= 11));
```

As the query results were often over 5 MB's , I ended up saving each result as a table using Google Big Query Web UI, then exported it into Google Storage, from where I downloaded it as a .csv file.

Reading in the Data into R

In order to avoid information loss due to coercion resulting from having integers that are over the 32 bit limit, I read in the BigQuery M-Lab query results R as characters. Since for this project I ended up using only a few of the M-lab dataset's variables I did not need to use the Web100 Variables look up table, but rather I set the variable types individually.

Data Cleaning

I wrote a script (dataConverterFunction.R) that takes a BigQuery result and returns a data frame that contains only the variables relevant (or potentially relevant) for this project. This script also converts the UTC time of the *web100_log_entry_log_time* variable into a date and replaces the column names with easier to type names:

```
#This function takes as input an M-Lab BigQuery  dataframe of completed
tests,
# where each column is of  "character" class.
#This dataframe must have the following columns:
#web100_log_entry_log_time,
#web100_log_entry_snap_MinRTT,
#connection_spec_client_geolocation_continent_code,
#connection_spec_client_geolocation_country_name,
#connection_spec_client_geolocation_city,
#connection_spec_server_geolocation_continent_code,
#connection_spec_server_geolocation_country_name,
#connection_spec_server_geolocation_city

#Return: a dataframe with 7 columns
#Schema: "logTime"              - integer,  UTC time
#        "RTT"                  - integer, roundtrip time
#        "clientContinent"      - character, continent code of the client
(optional)
#        "clientCountry"        - character, country of client (optional)
#        "clientCity"           - character, city of client (optional)
#        "serverContinent"      - character, continent code of server
(optional)
#        "serverCountry"        - character, country of server (optional)
#        "serverCity"           - character, city of server (optional)
#        "Date"                 - POSIXlt, format "y-m-d h:min:sec"
dataConverter <- function(dfChar){
        library(dplyr)

        #selecting the variables that will be needed
        df <- select(dfChar, web100_log_entry_log_time,
                            web100_log_entry_snap_MinRTT,

connection_spec_client_geolocation_continent_code,
                            connection_spec_client_geolocation_country_name,
                            connection_spec_client_geolocation_city,

connection_spec_server_geolocation_continent_code,
                            connection_spec_server_geolocation_country_name,
                            connection_spec_server_geolocation_city)

        #converting the UTC time into date
       df$date <- as.POSIXlt(as.integer(df$web100_log_entry_log_time),
                                  origin = "1970-01-01", tz = "GMT")

       #renaming the variables, using more manageable names
       names(df)<- c("logTime", "RTT" , "clientContinent", "clientCountry",
                            "clientCity", "serverContinent", "serverCountry",
                     "serverCity", "date")
       #creating a column that has the date in a month day year format
       df$moDayYear <- format(df$date, "%b %d %Y")
        #changing the type of logTime from character to integer
       df$logTime <- as.integer(df$logTime)
        #changing the type of RTT from character to double
       df$RTT <- as.double(df$RTT)
        #returning the resulting data frame df
       return(df)
}
```

Data Exploration

I wanted to see what the *RTT* looked like as a function of time during the time interval of interest, so I wrote a script that takes a data frame - the output of the *dataConverter* function - and plots *RTT* versus *date* and plots the horizontal line through the monthly mean RTT.

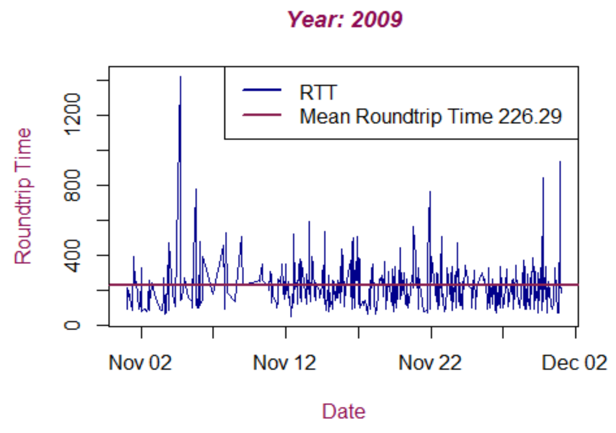
```
#This function takes as input a data frame with the schema provided below and
#it plots the Roundtrip Time v Date and the mean Roundtrip time

#Schema: "logTime"              - integer,  UTC time
#        "RTT"                  - integer, roundtrip time
#        "clientContinent"      - character, continent code of the client
(optional)
#        "clientCountry"        - character, country of client (optional)
#        "clientCity"           - character, city of client (optional)
#        "serverContinent"      - character, continent code of server
(optional)
#        "serverCountry"        - character, country of server (optional)
#        "serverCity"           - character, city of server (optional)
#        "Date"                 - POSIXlt, format "y-m-d h:min:sec"
plotter <- function(df){
        meanRTT<- mean(df$RTT)
        plot(df$date, df$RTT, type="l", col="darkblue", ylab="", xlab="")
        abline(h=meanRTT, col="violetred4", lwd=2)
        yr<- str_sub(as.character(df$moDayYear[1]), start=-4)
        title(main=paste("Year:", yr), xlab="Date", ylab="Roundtrip Time",
        col.main="deeppink4", font.main = 4, cex.main=1.1, col.sub =
"deeppink4",
        col.lab="deeppink4", cex.lab=1)
        legend('topright', c("RTT",paste("Mean Roundtrip Time",round(meanRTT,
digits=2))),
               lty=c(1,1), lwd=c(2.5,2.5),col=c("darkblue","violetred4"))
}
```

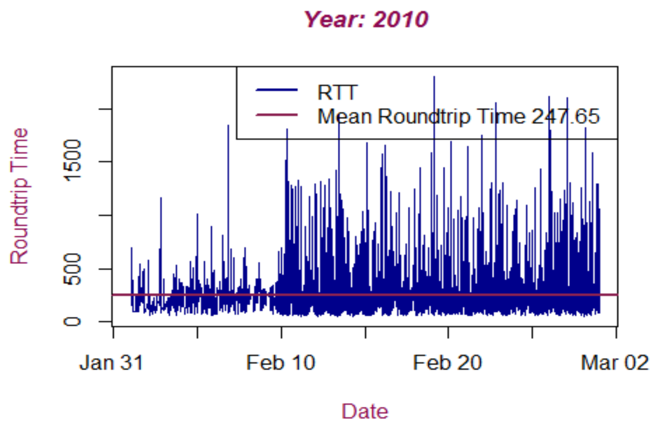
Then in the main part of the code I read in the .csv files and applied the dataConverter and plotter functions; I also calculated the monthly average RTT's for later use.

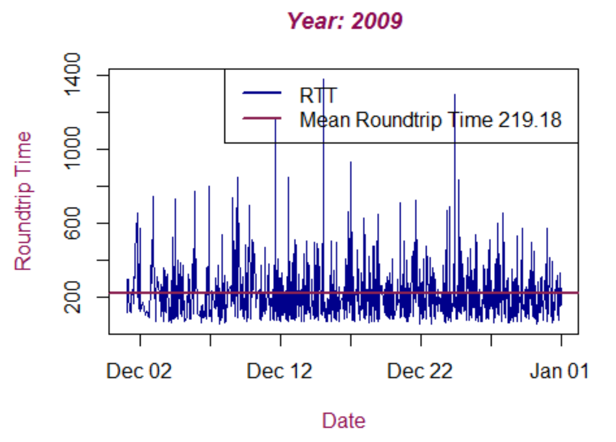
```
source("dataConverterFunction.R")
source("ByDayFunction.R")
```

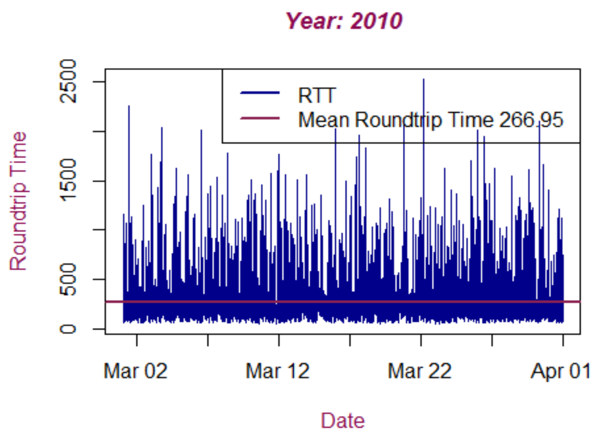
```
##
## Attaching package: 'dplyr'
##
## The following object is masked from 'package:stats':
##
##     filter
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

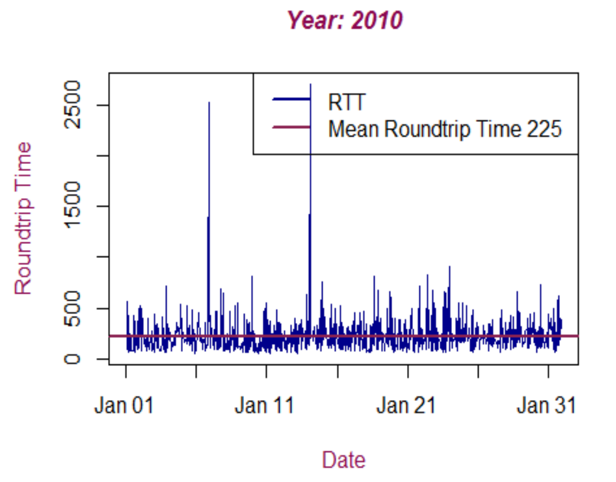
```
source("plotterFunction.R")

dfList<-list()
dfList[[1]]<-read.csv("EgyptNov2009RTTComplete.csv", colClasses="character")
dfList[[2]]<-read.csv("Egypt2009DecRTTComplete.csv", colClasses="character")
dfList[[3]]<-read.csv("Egypt2010JanRTTComplete.csv", colClasses="character")
dfList[[4]]<-read.csv("Egypt2010FebRTTComplete.csv", colClasses="character")
dfList[[5]]<-read.csv("Egypt2010MarRTTComplete", colClasses="character")
dfList[[6]]<-read.csv("Egypt2010AprRTTComplete.csv", colClasses="character")
dfList[[7]]<-read.csv("Egypt2010MayRTTComplete.csv", colClasses="character")
dfList[[8]]<-read.csv("June2010.csv", colClasses="character")
dfList[[9]]<-read.csv("Egypt2010JulyRTTComplete.csv", colClasses="character")
dfList[[10]]<-read.csv("Egypt2010AugRTTComplete.csv", colClasses="character")
dfList[[11]]<-read.csv("Egypt2010SeptRTComplete.csv", colClasses="character")
dfList[[12]]<-read.csv("Egypt2010OctRTComplete.csv", colClasses="character")
dfList[[13]]<-read.csv("Egypt2010NovRTTComplete.csv", colClasses="character")
dfList[[14]] <- read.csv("Egypt2010DecRTTComplete.csv",
colClasses="character")
dfList[[15]] <- read.csv("Egypt2011JanRTTComplete.csv",
colClasses="character")
dfList[[16]] <- read.csv("Egypt2011FebRTTComplete.csv",
colClasses="character")
dfList[[17]]<- read.csv("Egypt2011MarRTTComplete.csv",
colClasses="character")
dfList[[18]] <- read.csv("Egypt2011AprRTTComplete.csv",
colClasses="character")
workingDFList <- list()
for (i in (1:18)){
        workingDFList[[i]]<-  dataConverter(dfList[[i]])
}

for (i in (1:18)){
        plotter(workingDFList[[i]])
}
```

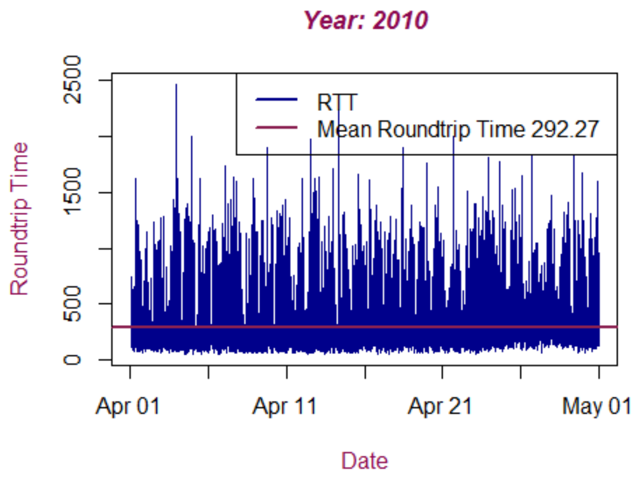

Year: 2009
RTT
Mean Roundtrip Time 226.29
Roundtrip Time
0
400
800
1200
Nov 02
Nov 12
Nov 22
Dec 02
Date

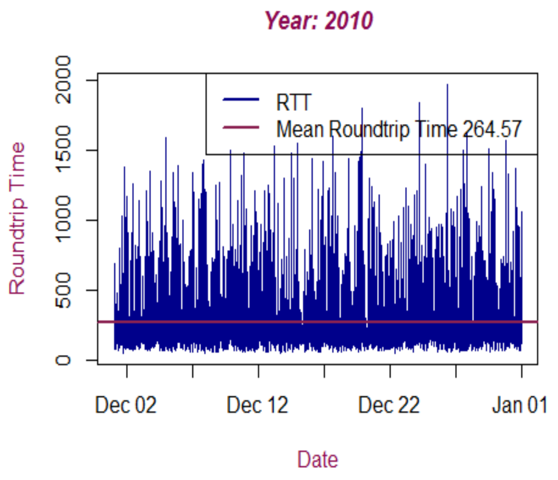

Year: 2010
RTT
Mean Roundtrip Time 247.65
Roundtrip Time
0
500
1500
Jan 31
Feb 10
Feb 20
Mar 02
Date

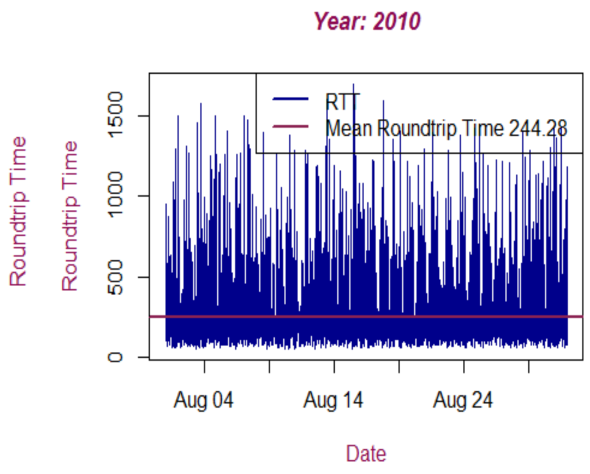

Year: 2009
RTT
Mean Roundtrip Time 219.18
Roundtrip Time
200
600
1000
1400
Dec 02
Dec 12
Dec 22
Jan 01
Date

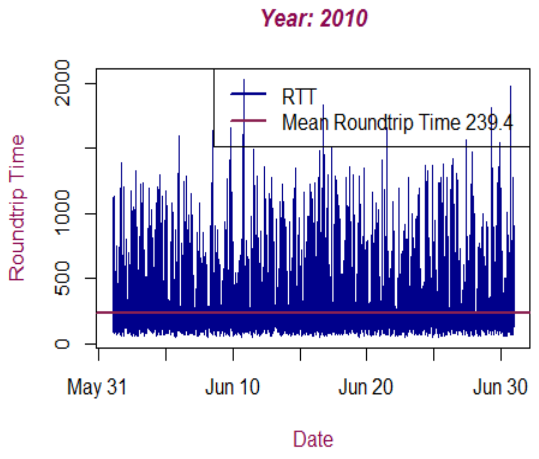

Year: 2010
RTT
Mean Roundtrip Time 266.95
Roundtrip Time
0
500
1500
2500
Mar 02
Mar 12
Mar 22
Apr 01
Date

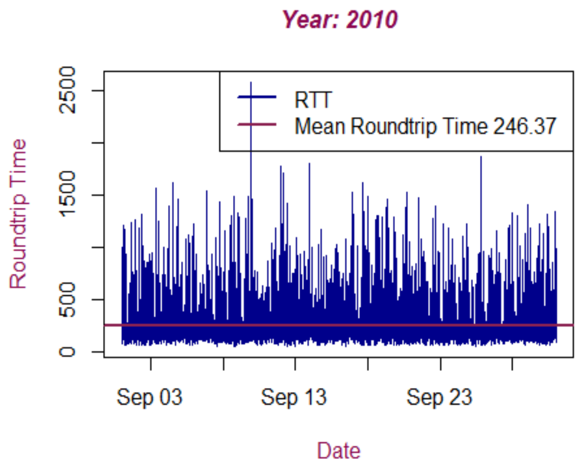

Year: 2010
RTT
Mean Roundtrip Time 225
Roundtrip Time
0
500
1500
2500
Jan 01
Jan 11
Jan 21
Jan 31
Date

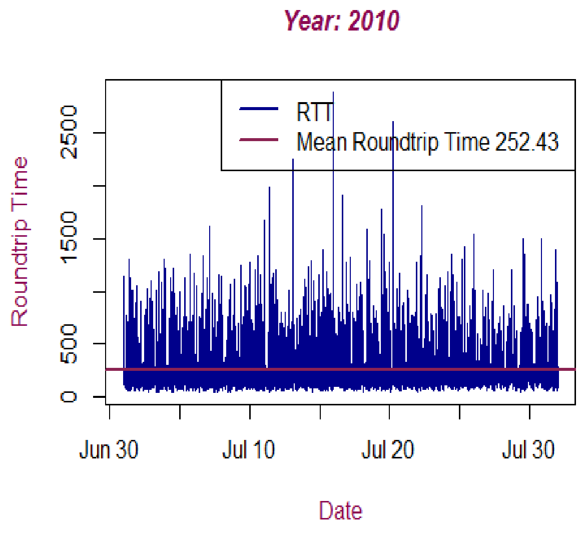

Year: 2010
RTT
Mean Roundtrip Time 292.27
Roundtrip Time
0
500
1500
2500
Apr 01
Apr 11
Apr 21
May 01
Date

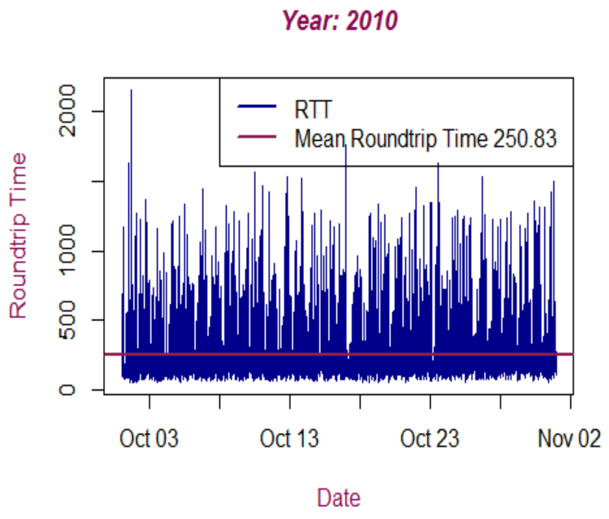

Year: 2010
RTT
Mean Roundtrip Time 264.57
Roundtrip Time
0
500
1000
1500
2000
Dec 02
Dec 12
Dec 22
Jan 01
Date

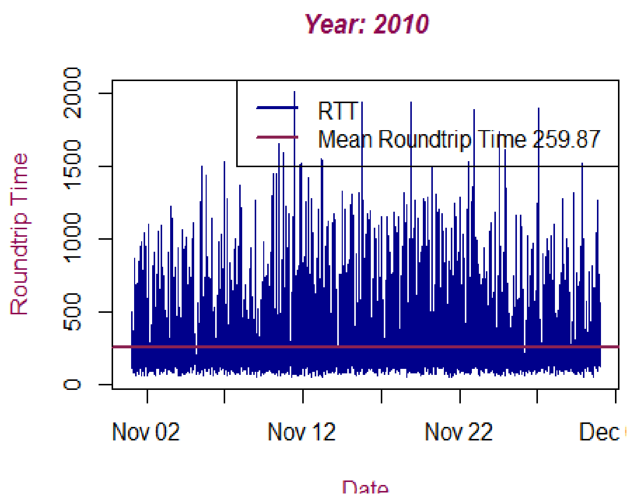

Year: 2010
RTT
Mean Roundtrip Time 244.28
Roundtrip Time
0
500
1000
1500
Aug 04
Aug 14
Aug 24
Date

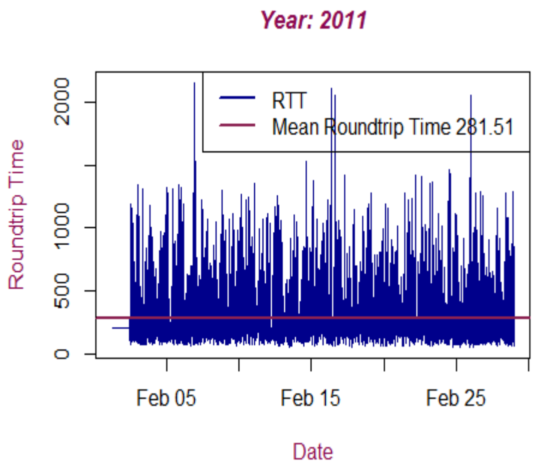

Year: 2010
RTT
Mean Roundtrip Time 239.4
Roundtrip Time
0
500
1000
2000
May 31
Jun 10
Jun 20
Jun 30
Date

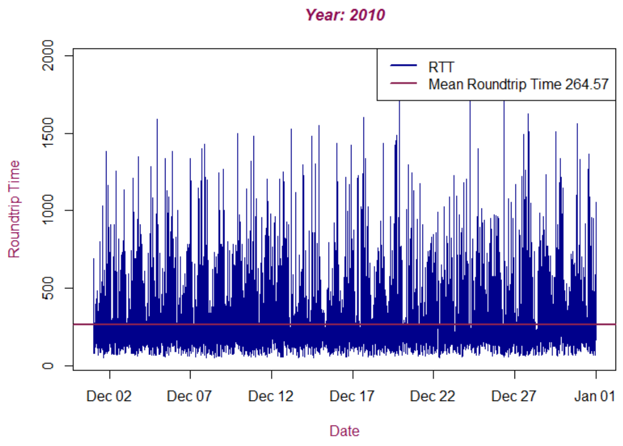

Year: 2010
RTT
Mean Roundtrip Time 246.37
Roundtrip Time
0
500
1500
2500
Sep 03
Sep 13
Sep 23
Date

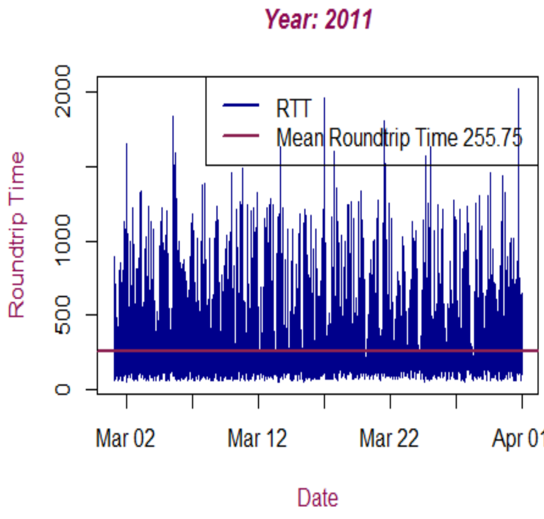

Year: 2010
RTT
Mean Roundtrip Time 252.43
Roundtrip Time
0
500
1500
2500
Jun 30
Jul 10
Jul 20
Jul 30
Date

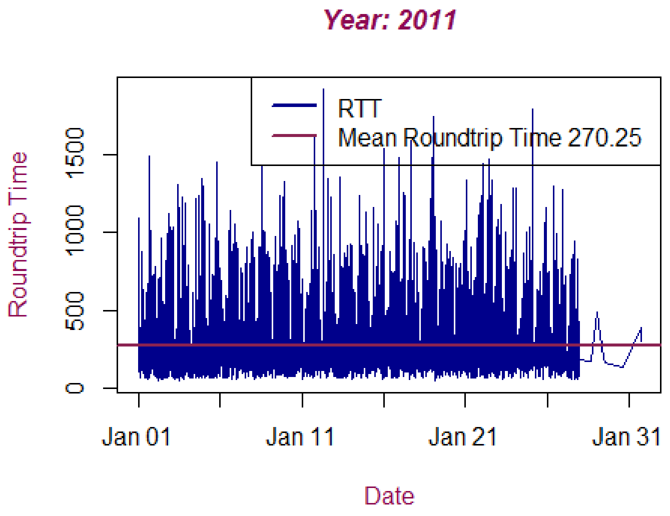

Year: 2010
RTT
Mean Roundtrip Time 250.83
Roundtrip Time
0
500
1000
2000
Oct 03
Oct 13
Oct 23
Nov 02
Date

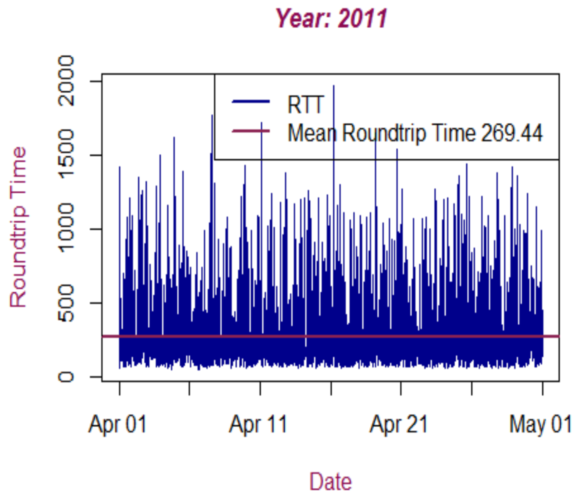

Year: 2010
Roundtrip Time
0
500
1000
1500
2000
RTT
Mean Roundtrip Time 259.87
Nov 02
Nov 12
Nov 22
Dec
Date


Year: 2011
Roundtrip Time
0
500
1000
2000
RTT
Mean Roundtrip Time 281.51
Feb 05
Feb 15
Feb 25
Date


Year: 2010
Roundtrip Time
0
500
1000
1500
2000
RTT
Mean Roundtrip Time 264.57
Dec 02
Dec 07
Dec 12
Dec 17
Dec 22
Dec 27
Jan 01
Date


Year: 2011
Roundtrip Time
0
500
1000
2000
RTT
Mean Roundtrip Time 255.75
Mar 02
Mar 12
Mar 22
Apr 01
Date


Year: 2011
Roundtrip Time
0
500
1000
1500
RTT
Mean Roundtrip Time 270.25
Jan 01
Jan 11
Jan 21
Jan 31
Date

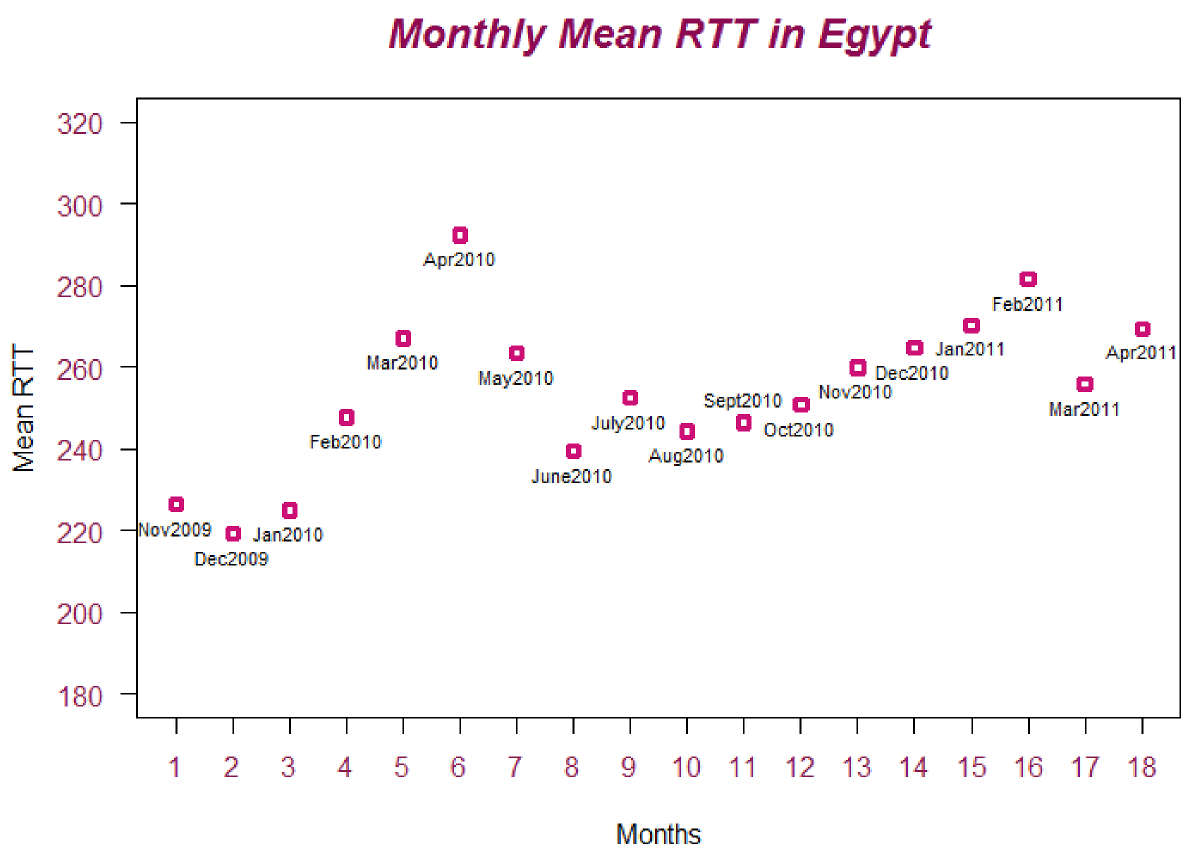

Year: 2011
Roundtrip Time
0
500
1000
1500
2000
RTT
Mean Roundtrip Time 269.44
Apr 01
Apr 11
Apr 21
May 01
Date

From the graphs it looks like there might be a slow down in internet speeds right around the time of the Arab Spring during January and February of 2011.

Monthly Mean RTT

To get a better understanding of what is going on, I calculated the monthly averages of the Roundtrip Times and plotted these means:

```
#calculating the means
avg_by_mo <-rep(0,18)

for (i in (1:18)){
        avg_by_mo[i] <- mean(workingDFList[[i]]$RTT)

}
mo_names <- c("Nov2009", "Dec2009", "Jan2010", "Feb2010", "Mar2010",
"Apr2010",
              "May2010", "June2010", "July2010", "Aug2010", "Sept2010",
"Oct2010",
              "Nov2010", "Dec2010", "Jan2011", "Feb2011", "Mar2011",
"Apr2011")
y_range<-seq(180,320, by =20)
x_range <-seq(1, 18, by=1)
x_ticks <-c("11-2009", "12-2009", "01-2010", "02-2010", "03-2010", "04-2011")
plot(avg_by_mo, pch = 22, cex = 1, lty = "solid", lwd = 3, col="deeppink3",
yaxt="n", xaxt="n",
      xlab="", ylab="", ylim = c(180,320), xlim=c(1, 18))
axis(2, at=y_range,labels=y_range, col.axis="violetred4", las=2)
axis(1, at=x_range,labels=x_range, col.axis="violetred4", las=1)
pos_vector <- rep(1, length(avg_by_mo))
pos_vector[11]=3
text(avg_by_mo, labels=mo_names, cex= 0.7, pos=pos_vector)
title(main="Monthly Mean RTT in Egypt",
      xlab="Months", ylab="Mean RTT", col.main="deeppink4", font.main = 4,
cex.main=1.5)
```



In the "Monthly Mean RTT in Egypt" graph RTT reaches peak values in April 2010 and in February 2011. The April 2010 peak coincides withincident when one of the undersea cables, SeaMeWe-4, a cable that connects Europe to the Middle East was cut around April 18th, 2010; the February 2011 peak coincides with the height of the unrest in Egypt during the Arab Spring.