Where is My Data Going?

Connor MacLeod

March 19, 2016

How bad is it really?

I decided to finally take on a problem for this assignment that's been nagging at me for a long time. I always assumed that Windows used a lot more traffic at idle than my Ubuntu did, but I wanted to know exactly how much, and if I could reduce it somehow. I have already turned off every telemetry setting I could find, but I keep hearing rumors about Windows sending data anyways. So I came up with a plan.

On each OS do the following:

- Close all windows and non-essential programs in the tray (for Windows), like Skype, Steam, Dropbox, etc.
- Run Wireshark for 300s
- Filter down to only traffic from involving the host and do general data cleaning
- Import into R and start digging

Step 1: Data Gathering

After running Wireshark on each OS, I have two csv files, 'TakeHomeWindows.csv' and 'TakeHomeUbuntu.csv'. The next step is importing them into R.

```
windows <- read.csv("C:/Users/Connor/Documents/SRT411/TakeHomeWindows.csv")
ubuntu <- read.csv("C:/Users/Connor/Documents/SRT411/TakeHomeUbuntu.csv")

typeof(windows)
```

```
## [1] "list"
```

```
typeof(ubuntu)
```

```
## [1] "list"
```

As you can see, the variables are in list form. In order to be able to best work with them, it is ideal to put them in dplyr's tbl_df format.

```
library(dplyr)
```

```
windf = tbl_df(data.frame(windows))
ubndf = tbl_df(data.frame(ubuntu))
```

Ok, now we can begin cleaning this data to make it useful for us.

Step 2: Cleaning Data

Let's take a look at the data in it's current state.

```r
# Sample first 10 lines
head(windf)
```

```
## Source: local data frame [6 x 7]
##
##     No.     Time             Source     Destination Protocol Length
##   (int)    (dbl)            (fctr)          (fctr)   (fctr)  (int)
## 1     1 0.000000     192.168.1.134 239.255.255.250     SSDP    369
## 2     2 0.039006 Cisco-Li_65:d1:20       Broadcast      ARP     42
## 3     3 0.101163     192.168.1.134 239.255.255.250     SSDP    369
## 4     4 0.207271     192.168.1.134 239.255.255.250     SSDP    378
## 5     5 0.309053     192.168.1.134 239.255.255.250     SSDP    378
## 6     6 0.413406     192.168.1.134 239.255.255.250     SSDP    433
## Variables not shown: Info (fctr)
```

```r
head(ubndf)
```

```
## Source: local data frame [6 x 7]
##
##     No.     Time             Source   Destination Protocol Length
##   (int)    (dbl)            (fctr)        (fctr)   (fctr)  (int)
## 1     1 0.000000 AsustekC_bc:42:24     Broadcast      ARP     60
## 2     2 0.765025     192.168.1.100 192.168.1.255      UDP     63
## 3     3 0.766154     192.168.1.100 192.168.1.255      UDP     63
## 4     4 0.769730     192.168.1.135 192.168.1.255      UDP     63
## 5     5 0.770953     192.168.1.135 192.168.1.255      UDP     63
## 6     6 4.009094 AsustekC_bc:42:24     Broadcast      ARP     60
## Variables not shown: Info (fctr)
```

```r
# Table stats
str(windf)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1308 obs. of  7 variables:
##  $ No.        : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Time       : num  0 0.039 0.101 0.207 0.309 ...
##  $ Source     : Factor w/ 34 levels "111.221.29.254",..: 12 28 12 12 12 12 12 12 11 12 ...
##  $ Destination: Factor w/ 37 levels "111.221.29.254",..: 19 30 19 19 19 19 19 19 21 19 ...
##  $ Protocol   : Factor w/ 15 levels "ARP","BROWSER",..: 12 1 12 12 12 12 12 12 13 12 ...
##  $ Length     : int  369 42 369 378 378 433 433 443 55 443 ...
##  $ Info       : Factor w/ 322 levels "[TCP Dup ACK 1117#1] 50089  >  443 [ACK] Seq=228 Ack=1461 Win=
```

```r
str(ubndf)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    960 obs. of  7 variables:
##  $ No.        : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Time       : num  0 0.765 0.766 0.77 0.771 ...
##  $ Source     : Factor w/ 14 levels "192.168.1.1",..: 8 2 2 5 5 8 8 2 2 5 ...
##  $ Destination: Factor w/ 17 levels "192.168.1.100",..: 13 3 3 3 3 13 13 3 3 3 ...
##  $ Protocol   : Factor w/ 12 levels "ARP","BROWSER",..: 1 12 12 12 12 1 1 12 12 12 ...
##  $ Length     : int  60 63 63 63 63 60 60 63 63 63 ...
##  $ Info       : Factor w/ 147 levels "[TCP Dup ACK 132#1] 32469 > 55412 [ACK] Seq=175 Ack=777 Win=665
```

So three things are immediately clear to me from this data. The first is that we have a superflous column. The 'No.' column is useless to us because R already provides a numbered column. The second is that there seems to be a lot of data picked up by Wireshark that doesn't even involve our host IP (192.168.1.109). Finally, and the total run times are slightly different. So let's fix all of those.

```
# Remove Column -> Only packets going to or coming from our host -> Ensure total run time is < 300s
cleanpcap = function(tbldf){
  tbldf = tbldf %>%
    select(-No.) %>%
    filter(
      (Source == "192.168.1.109" | Destination == "192.168.1.109")
      & Time <= 300)
}

windat <- cleanpcap(windf)
ubndat <- cleanpcap(ubndf)

head(windat)
```

```
## Source: local data frame [6 x 6]
##
##       Time        Source   Destination Protocol Length
##      (dbl)        (fctr)        (fctr)   (fctr)  (int)
## 1 0.713152 192.168.1.109 24.156.131.29      TCP     55
## 2 0.741258 24.156.131.29 192.168.1.109      TCP     66
## 3 6.631118 192.168.1.109 24.156.131.98  TLSv1.2    144
## 4 6.631487 192.168.1.109 24.156.131.98  TLSv1.2    100
## 5 6.651011 24.156.131.98 192.168.1.109      TCP     60
## 6 6.651201 24.156.131.98 192.168.1.109      TCP     60
## Variables not shown: Info (fctr)
```

```
head(ubndat)
```

```
## Source: local data frame [6 x 6]
##
##       Time        Source     Destination Protocol Length
##      (dbl)        (fctr)          (fctr)   (fctr)  (int)
## 1 6.982354 192.168.1.109 239.255.255.250     SSDP    209
## 2 6.988411 192.168.1.100   192.168.1.109      UDP    381
## 3 6.988510 192.168.1.109 239.255.255.250   IGMPv2     46
## 4 7.037509 192.168.1.109   192.168.1.100      TCP     74
## 5 7.046852 192.168.1.100   192.168.1.109      TCP     74
## 6 7.046891 192.168.1.109   192.168.1.100      TCP     66
## Variables not shown: Info (fctr)
```

```
str(windat)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    511 obs. of  6 variables:
```

```
##  $ Time       : num  0.713 0.741 6.631 6.631 6.651 ...
##  $ Source     : Factor w/ 34 levels "111.221.29.254",..: 11 19 11 11 20 20 20 20 11 20 ...
##  $ Destination: Factor w/ 37 levels "111.221.29.254",..: 21 11 22 22 11 11 11 11 22 11 ...
##  $ Protocol   : Factor w/ 15 levels "ARP","BROWSER",..: 13 13 14 14 13 13 14 14 13 13 ...
##  $ Length     : int  55 66 144 100 60 60 100 181 66 149 ...
##  $ Info       : Factor w/ 322 levels "[TCP Dup ACK 1117#1] 50089  >  443 [ACK] Seq=228 Ack=1461 Win=2
```

```
str(ubndat)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    165 obs. of  6 variables:
##  $ Time       : num  6.98 6.99 6.99 7.04 7.05 ...
##  $ Source     : Factor w/ 14 levels "192.168.1.1",..: 3 2 3 3 2 3 3 3 2 3 ...
##  $ Destination: Factor w/ 17 levels "192.168.1.100",..: 9 2 9 1 2 1 9 1 2 1 ...
##  $ Protocol   : Factor w/ 12 levels "ARP","BROWSER",..: 10 12 6 11 11 11 10 3 11 11 ...
##  $ Length     : int  209 381 46 74 74 66 211 241 234 66 ...
##  $ Info       : Factor w/ 147 levels "[TCP Dup ACK 132#1] 32469 > 55412 [ACK] Seq=175 Ack=777 Win=665
```

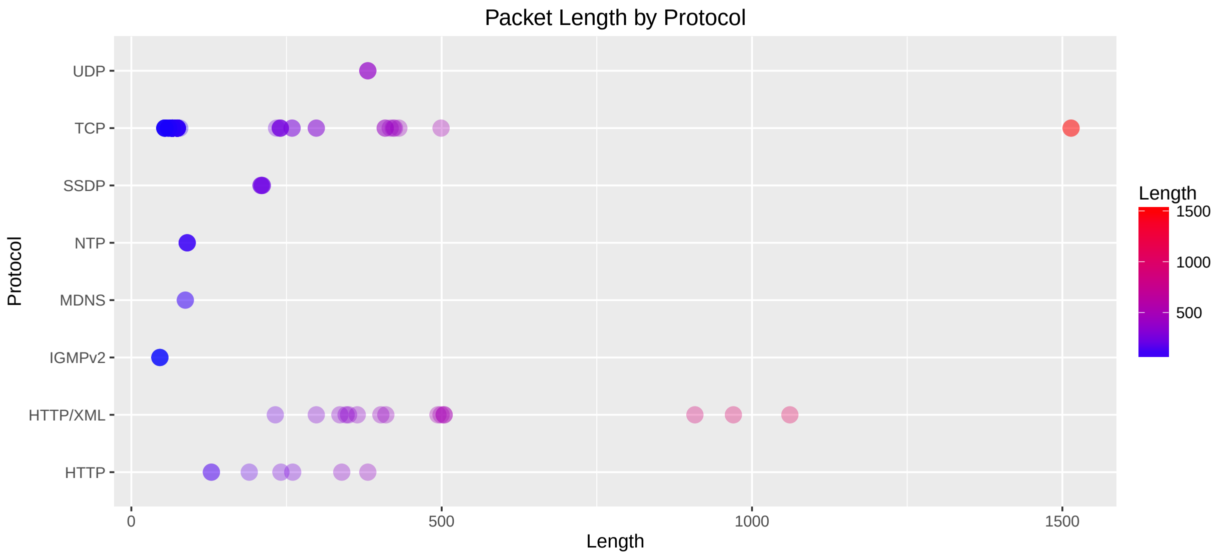
> Just as I expected. Windows is using more than **Three Times** as much data as Ubunut (511 packets vs 165). Let's see if we can track down what's causing it.

Step 3: Digging

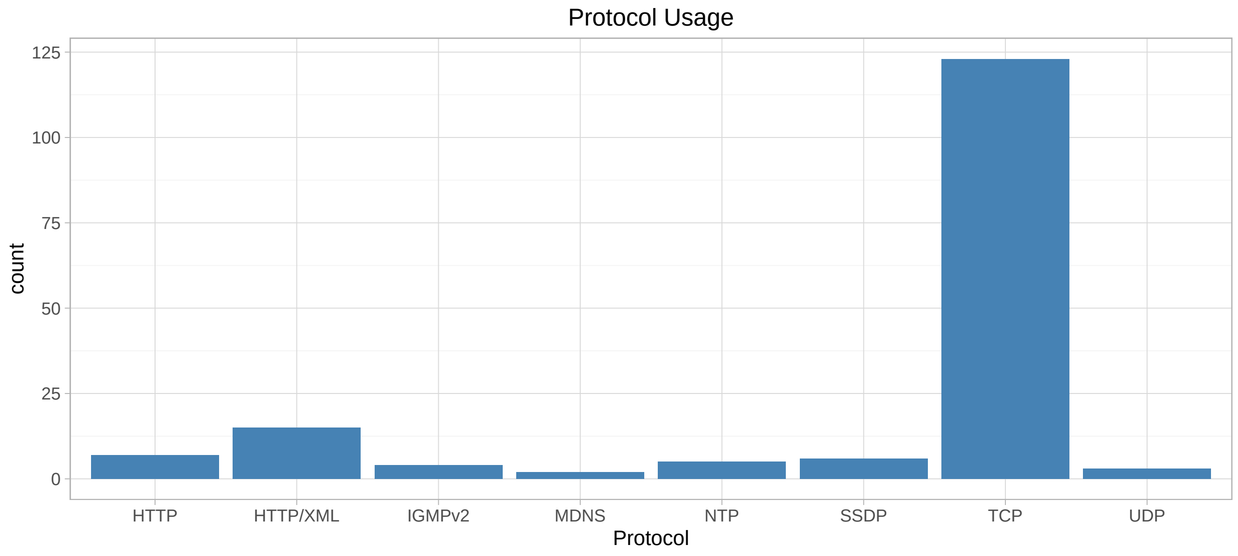
We need to start out by setting a baseline with our Ubuntu, so we can tell what is different about Windows. I plan to look at length, length over time, protocol, length vs protocol, and inbound vs. outbound.

```
library(ggplot2)
library(grid)
library(gridExtra)
```

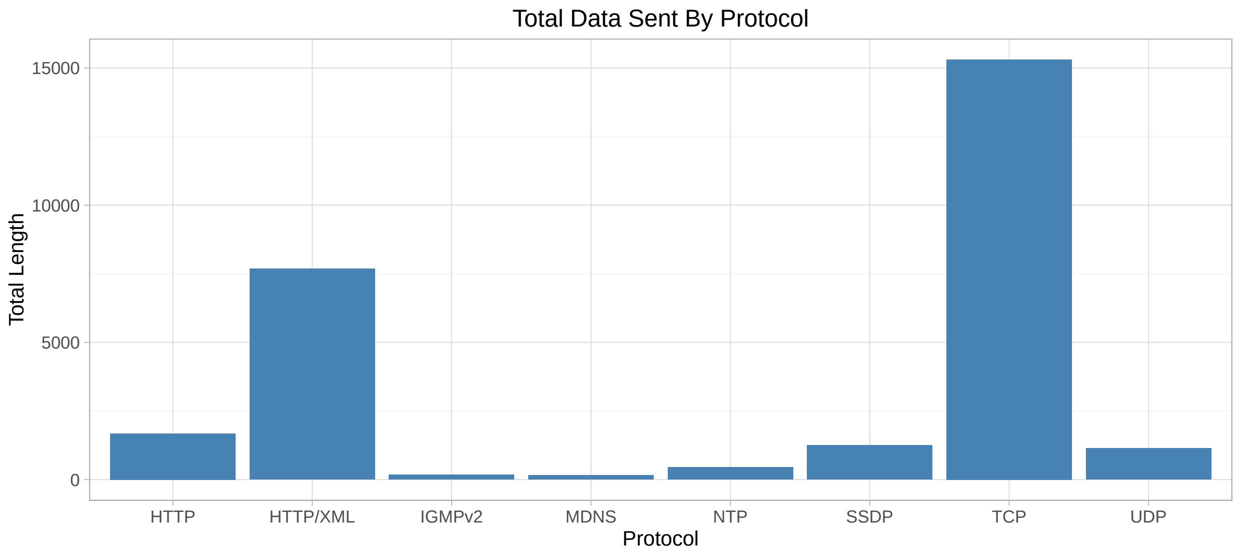
```
ulength <- ggplot(ubndat, aes(Length, Protocol)) +
  geom_point(aes(color = Length), size = 4, alpha = 1/3) +
  scale_color_gradient(low = "blue", high = "red") +
  ggtitle("Packet Length by Protocol")

uproto <- ggplot(ubndat, aes(Protocol)) +
  geom_bar(fill = "steelblue") +
  ggtitle("Protocol Usage") +
  theme_light()

uprolen <- ggplot(ubndat, aes(factor(Protocol), Length)) +
  geom_bar(stat = "identity", position = "stack", fill = "steelblue") +
  xlab("Protocol") +
  ylab("Total Length") +
  ggtitle("Total Data Sent By Protocol") +
  theme_light()

utimeline <- qplot(Time, Length, data = ubndat, geom = "area") +
  geom_area(fill = "steelblue") +
  theme_light() +
  ylim(0, 400) +
  ggtitle("Packet Length Over Time")


ulength
```



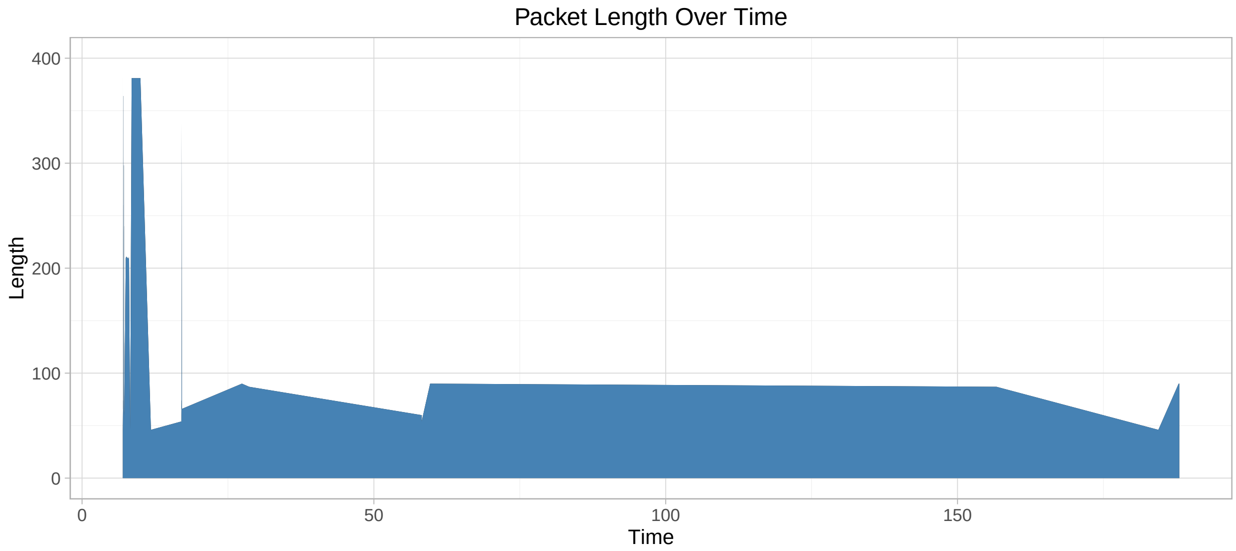
```
uproto
```



```
uprolen
```

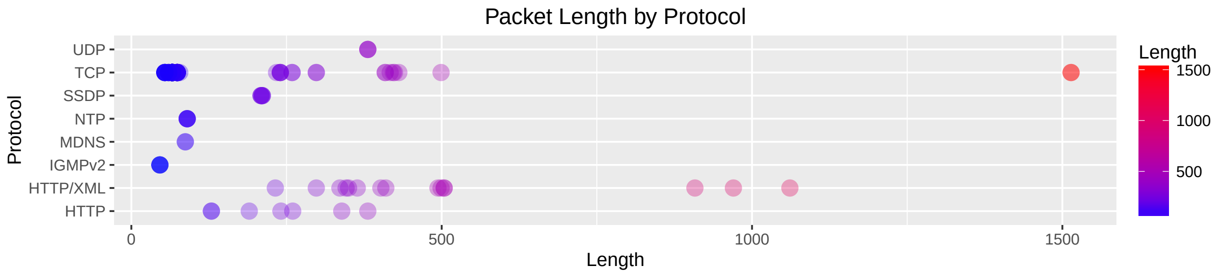


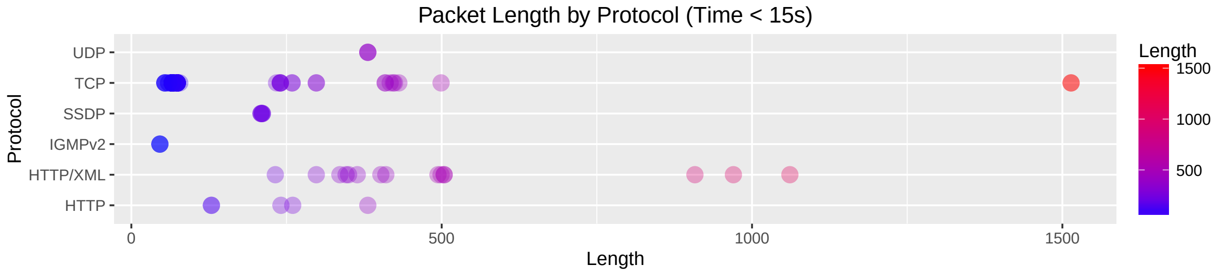
```
utimeline
```



So here we can see that while most of the traffic is TCP (the dense, solid color), the largest packets were from HTTP/XML. We also see that confirmed in the count graph. TCP has well over ten times the packets of any other protocol. However, when we look at total data sent per protocol, we see that HTTP/XML is actually responsible for more than it's fair share. Looking at the timeline, we see the large packets were all sent in the beginning. Let's find out what protocol did that (I think it's HTTP).

```
ulength2 <- ggplot(filter(ubndat, Time < 15), aes(Length, Protocol)) +
  geom_point(aes(color = Length), size = 4, alpha = 1/3) +
  scale_color_gradient(low = "blue", high = "red") +
  ggtitle("Packet Length by Protocol (Time < 15s)")

grid.arrange(ulength, ulength2, ncol = 1)
```

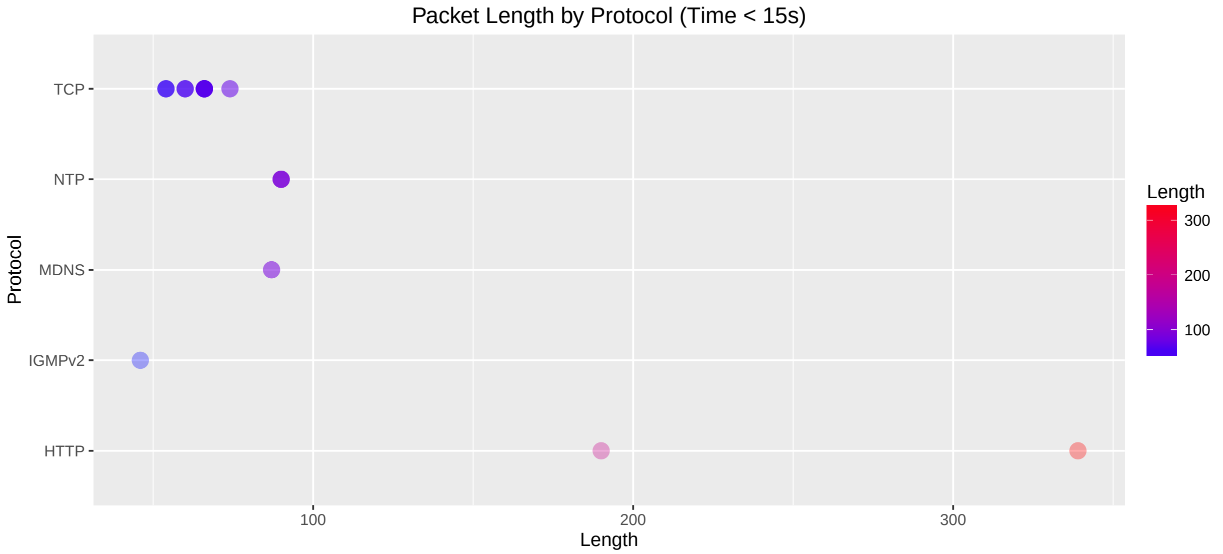




So as we can see, most of it looks the same. So the conclusion here is that the rest of the data after the intial burst was all of the other protocols we see.

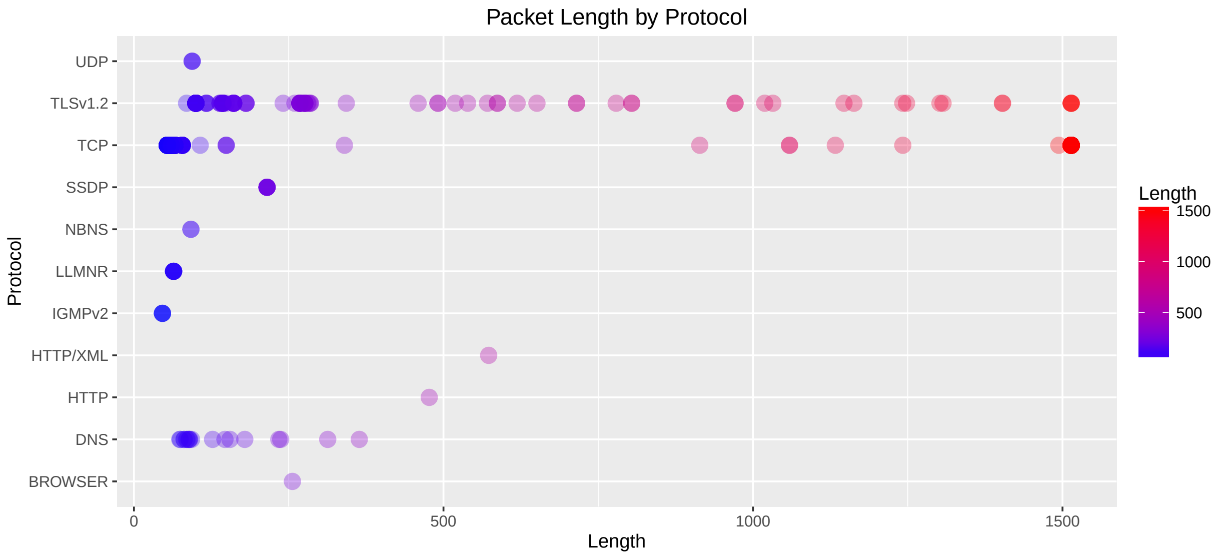
My guess as to the source of this traffic spike is that I ran wireshark immediately after booting into Ubunut to minimise background processes. This means that if anything runs on startup, wireshark would have caught the tail end of that, so in reality, the idle traffic is actually everything post 15s.

```r
ggplot(filter(ubndat, Time > 15), aes(Length, Protocol)) +
  geom_point(aes(color = Length), size = 4, alpha = 1/3) +
  scale_color_gradient(low = "blue", high = "red") +
  ggtitle("Packet Length by Protocol (Time < 15s)")
```

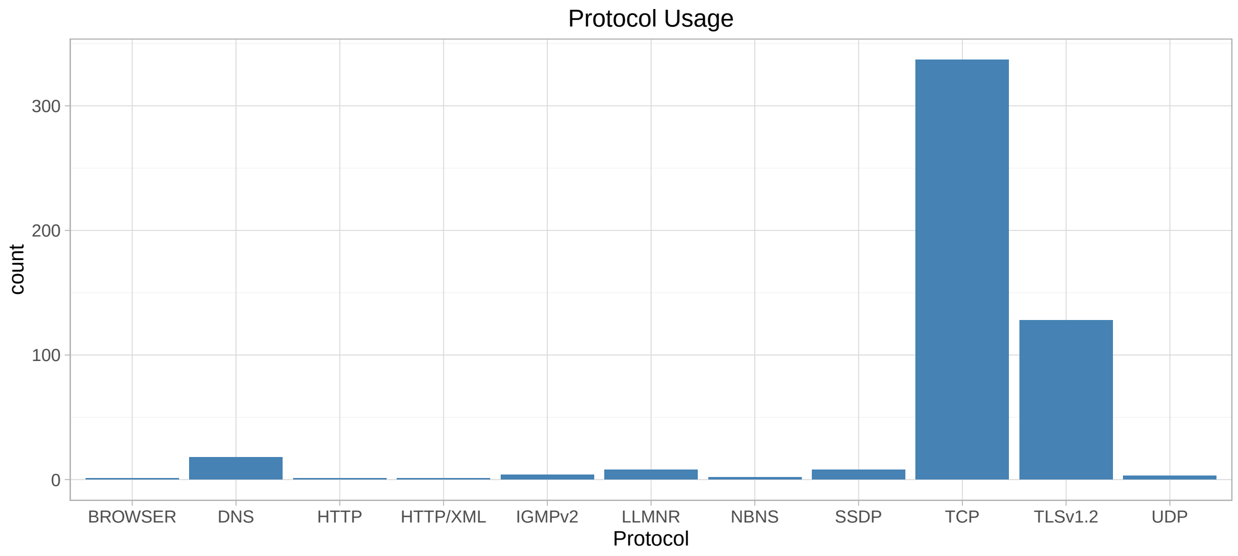


Wow. That's impressive, very little traffic at all. Let's see how Windows stands up.

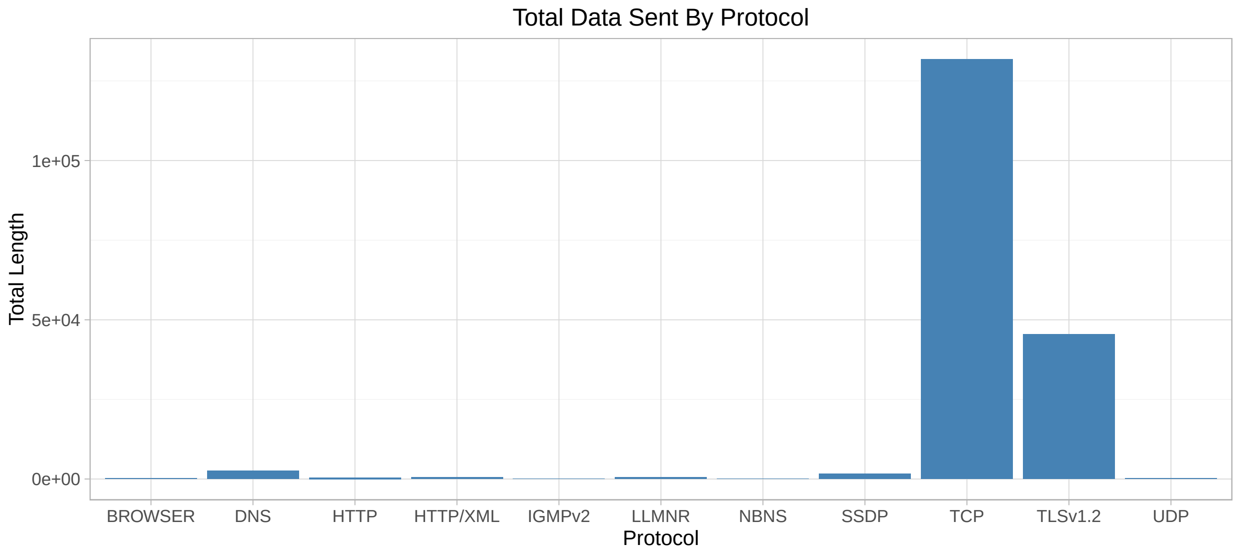
```
wlength <- ggplot(windat, aes(Length, Protocol)) +
  geom_point(aes(color = Length), size = 4, alpha = 1/3) +
  scale_color_gradient(low = "blue", high = "red") +
  ggtitle("Packet Length by Protocol")

wproto <- ggplot(windat, aes(Protocol)) +
  geom_bar(fill = "steelblue") +
  ggtitle("Protocol Usage") +
  theme_light()

wprolen <- ggplot(windat, aes(factor(Protocol), Length)) +
  geom_bar(stat = "identity", position = "stack", fill = "steelblue") +
  xlab("Protocol") +
  ylab("Total Length") +
  ggtitle("Total Data Sent By Protocol") +
  theme_light()

wtimeline <- qplot(Time, Length, data = windat, geom = "area") +
  geom_area(fill = "steelblue") +
  theme_light() +
  ylim(0, 400) +
  ggtitle("Packet Length Over Time")


wlength
```



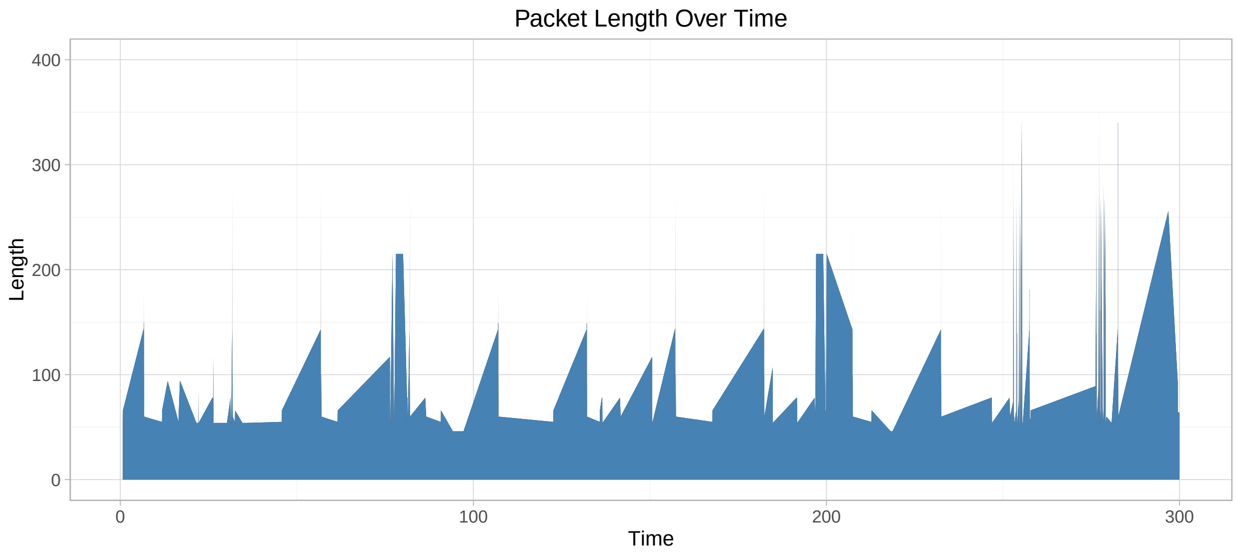
```
wproto
```



```
wprolen
```



```
wtimeline
```



Interesting. On the upside, at least most of the traffic being sent is encrypted through TLS. On the downside, there are **a lot** of TCP and TLS packets measuering in the ~1500 area, then they are scattered until the ~300 area. Looking at the protocol usage we can see that although both TLS and TCP had high packet length, that TCP is used twice as much and sent twice as much data as TLS. We can't even give saving grace in the timeline, it is intermittent thoughout. Let's see if we can find out what is causing this.

Step 4: The Cause

Let's start by just isolating TCP and TLS, since they are oviously the problem here.

```
# Get Sources for the biggest TCP and TLS packets
wins <- windat %>%
  filter(Protocol == "TCP" | Protocol == "TLSv1.2") %>%
  arrange(desc(Length), Source)
```

I am going to venture a guess and assume that it's only a handful of IP's causing this. Let's isolate them and see.

```
# Get list of source IP's using big packets
wins <- wins %>%
  filter(Length > 1400, Source != "192.168.1.109")

unique(wins$Source)
```

```
## [1] 111.221.29.254  13.107.5.88     131.253.61.66   134.170.108.200
## [5] 137.116.74.190  23.99.116.116   40.121.144.182
## 34 Levels: 111.221.29.254 13.107.5.88 131.253.61.66 ... fe80::ffff:ffff:fffe
```

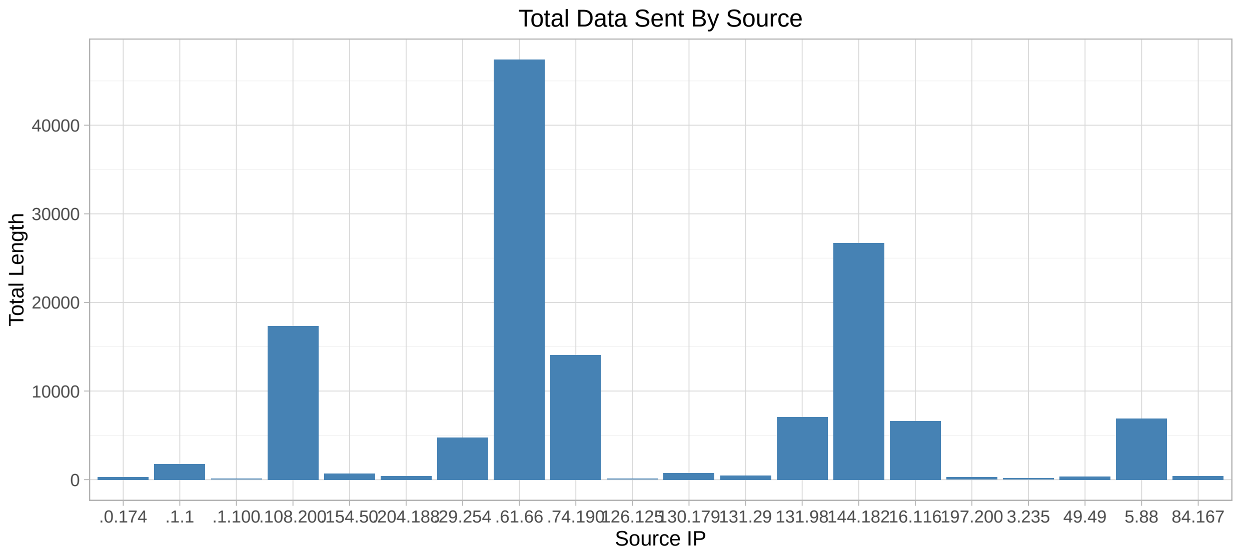
So those are the big bad IP's that are ruining my network. Here's what I could get on them.

IP	Details	Domain
111.221.29.254	Microsoft Corp, Singapore	vortex-win.data.microsoft.com
13.107.5.88	Microsoft Azure, US	e-0009.e-msedge.net
131.253.61.66	Microsoft Corp, US	N/A
134.170.108.200	Microsoft Corp, US	by3302-e.1drc.com
137.116.74.190	Microsoft Corp, US	N/A
23.99.116.116	Microsoft Azure, Hong Kong	N/A
40.121.144.182	Microsoft Acure, US	N/A

Ugh. Well that sucks. All of the biggest packet hogs are Microsoft demanding my data. If there was ever a reason to switch to a Linux or FreeBSD dsitro, this is it.

“But what if something is genreating a ton of smaller packets instead of a few large packets?” Good question, let’s see.

```r
wins <- windat %>%
  filter(Source != "192.168.1.109") %>%
  mutate(Labels = substr(Source, 8, 15))

ggplot(wins, aes(factor(Labels), Length)) +
  geom_bar(stat = "identity", position = "stack", fill = "steelblue") +
  xlab("Source IP") +
  ylab("Total Length") +
  ggtitle("Total Data Sent By Source") +
  theme_light()
```



And still, our culprits are (in order): 131.253.61.66, 40.121.144.182, 134.170.108.200, 137.116.74.190

AKA Micosoft

Conclusion

In the age of fast and plentiful data and internet, many argue that this may not be a ‘big deal’, but I personally disagree. The fact that I can’t turn this off is itself unsettling, but also the fact that they are using up my network traffic, just because I use their OS, is completely ridiculous to me. You also have to consider the privacy implications here. Do you know how much information can be transmitted per one packet of length ~1500? And now consider that 136606 bytes (~136 Kilobytes) were transmitted in the 5 minutes I was recording. That is approximately 17075 words in ASCII. A 10 page essay about your computer habits is being sent to Micrososft **Every 5 Minutes**. Looks like I’ll be using Ubuntu a lot more often.