Mesures de performance réseau

Arnaud Legrand

03/02/2015

Contents


```
knitr::opts_chunk$set(fig.align = "center",
                      fig.retina = 2,
                      fig.width = 6,
                      fig.ext='png',
                      cache = TRUE,
                      cache.lazy = FALSE)
```

Une première approche pour comprendre à quoi ressemble ce système

Dans le cadre de ces TPs, voici le genre de commandes shell qui peuvent être utilisées pour collecter des mesures à moindre coût:

```
ping -D liglab.imag.fr > liglab.log 2>&1
ping -D liglab.imag.fr -s 20480 -i .2
```

On peut alors leur mettre un coup de grep et de sed pour les reformater. Là, j'avais choisi de faire quelque chose d'un poil robuste (en perl) et qui me rapporte les lignes non conformes à ce qui est attendu.

```
  $#ARGV==1 or die "Usage: convert.pl <input.log> <output.csv> -->'$#ARGV'";
  $input = $ARGV[0];
  $output = $ARGV[1];
  open(INPUT,$input) or die;
  open(OUTPUT,"> $output") or die;
  $l=<INPUT>;
  print OUTPUT qw(time,size,name,ip,seq,ttl,delay)."\n";

  while(defined($l=<INPUT>)) {
      chomp($l);
      if($l eq "") { last; }
      if($l =~ /^\[(.*)\] (.*) bytes from (.*) \((.*)\): icmp_seq=(.*) ttl=(.*) time=(.*) ms$/) {
          my($time,$size,$name,$ip,$seq,$ttl,$delay)=($1,$2,$3,$4,$5,$6,$7);
          print OUTPUT (join(',',($time,$size,$name,$ip,$seq,$ttl,$delay))."\n");
      } else { print "---> $l\n"; }
  }
```

Quoi qu'il en soit, voici le genre de données que j'ai pu récupérer. Si vous n'avez pas pu faire de mesures par vous même (ce qui est dommage...), vous pouvez les utiliser...

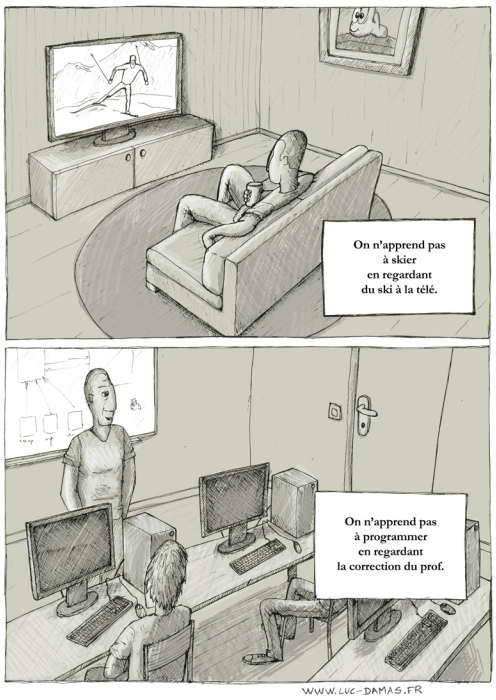


Figure 1: C'est en forgeant qu'on devient forgeron!

- RICM4_EP_ping/stackoverflow.csv.gz
- RICM4_EP_ping/stackoverflow.log.gz
- RICM4_EP_ping/google.csv.gz
- RICM4_EP_ping/liglab.log.gz
- RICM4_EP_ping/google.log.gz
- RICM4_EP_ping/liglab.csv.gz
- RICM4_EP_ping/liglab2.log.gz
- RICM4_EP_ping/liglab2.csv.gz

Analyses

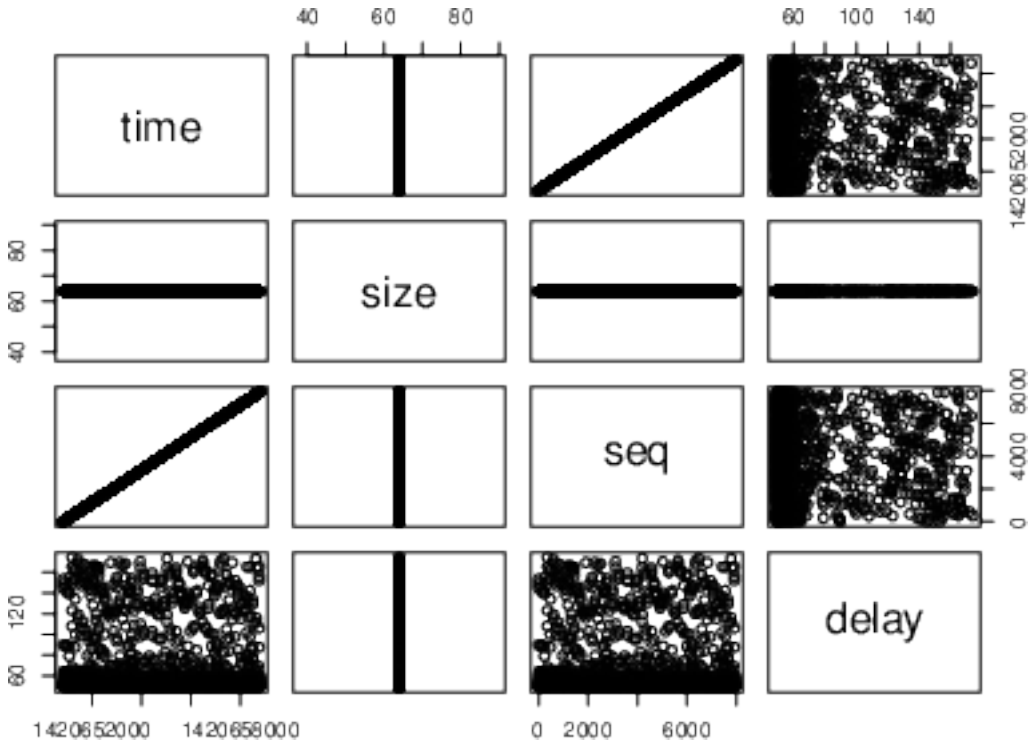
```
library(ggplot2)
library(plyr)
```

Je prend liglab, une première série de mesures vers le serveur web du labo.

```
liglab<-read.csv("liglab.csv.gz",header=T);
df = liglab
```

Allons-y pour une première visualisation:

```
plot(df[names(df)%in% c("time","size","seq","delay")]);
```

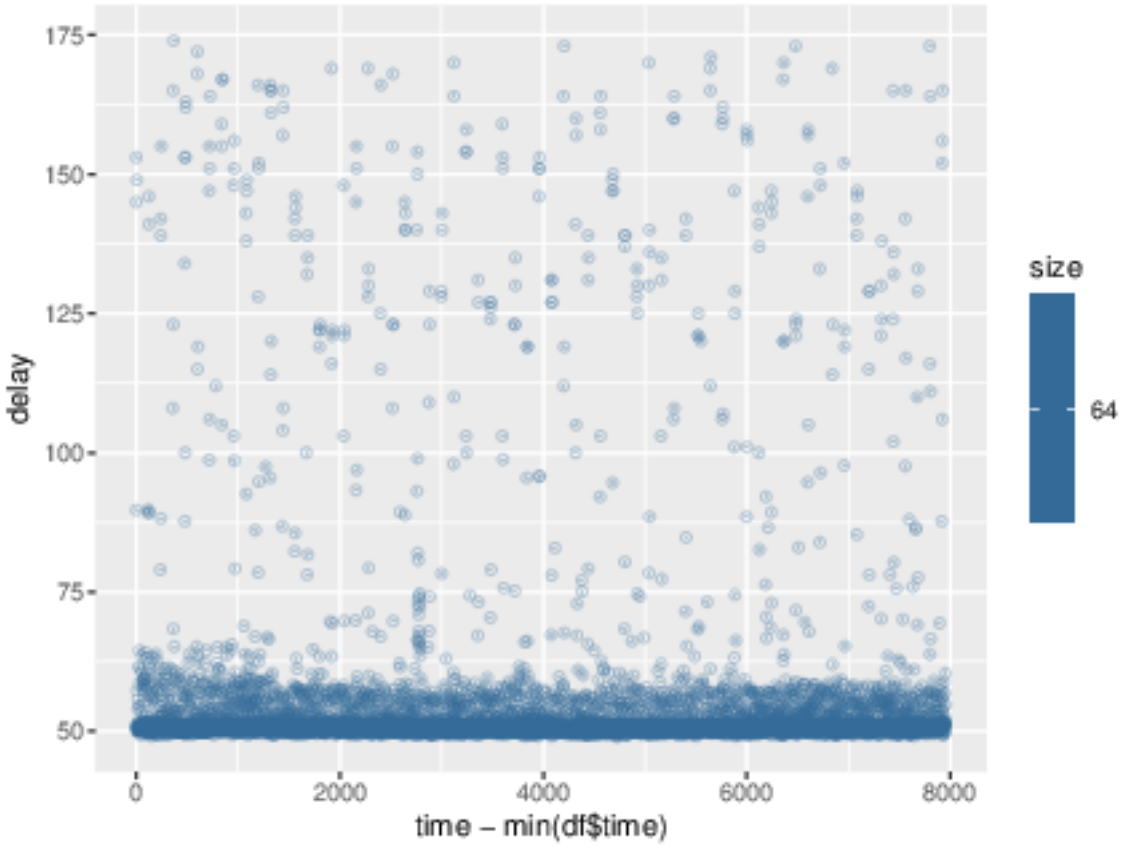


Que voit-on ?

- une seule taille utilisée
- le delay est vraiment très variable

Regardons ça de plus près

```
p = ggplot(data=df,aes(x=time-min(df$time),y=delay, color=size)) + geom_point(alpha=.3)
p
```

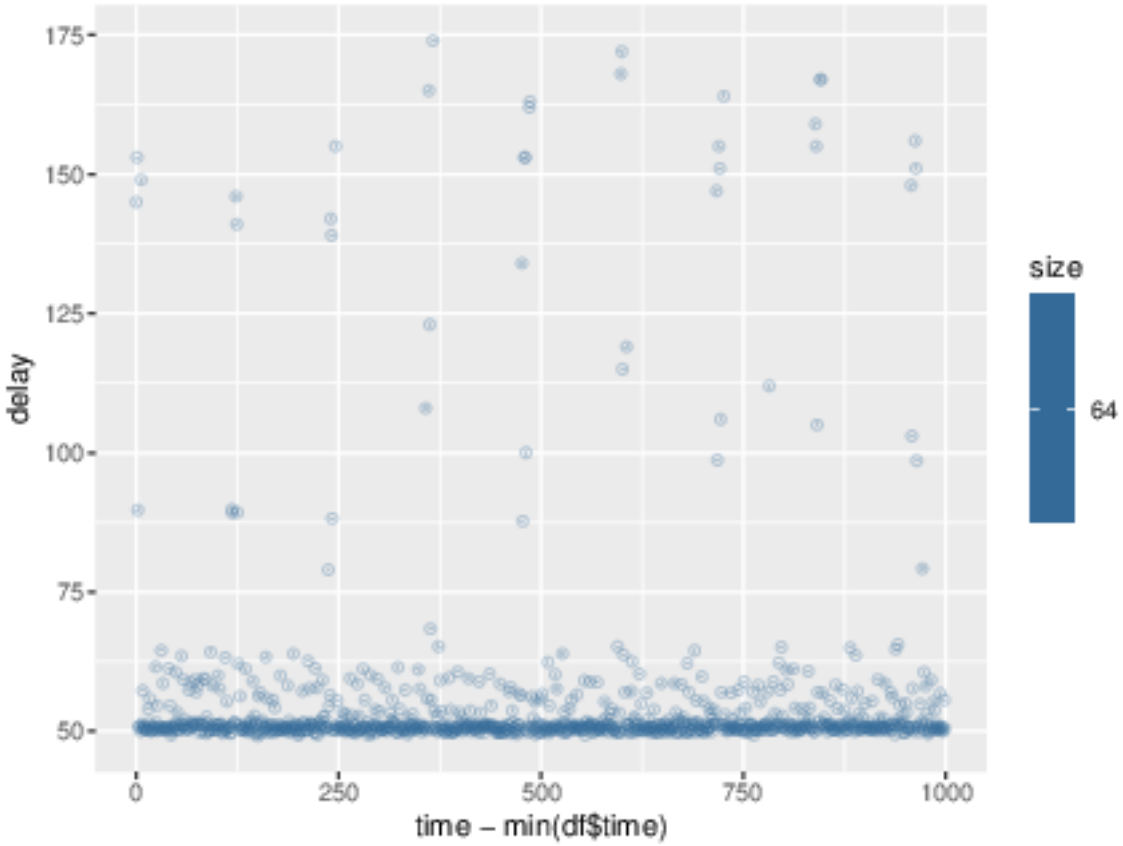


C'est à peu près stable dans le temps mais il y a l'air d'y avoir des aligements de points verticaux.

Regardons, d'encore plus près...

```
p + xlim(0,1000)
```

```
## Warning: Removed 6943 rows containing missing values (geom_point).
```

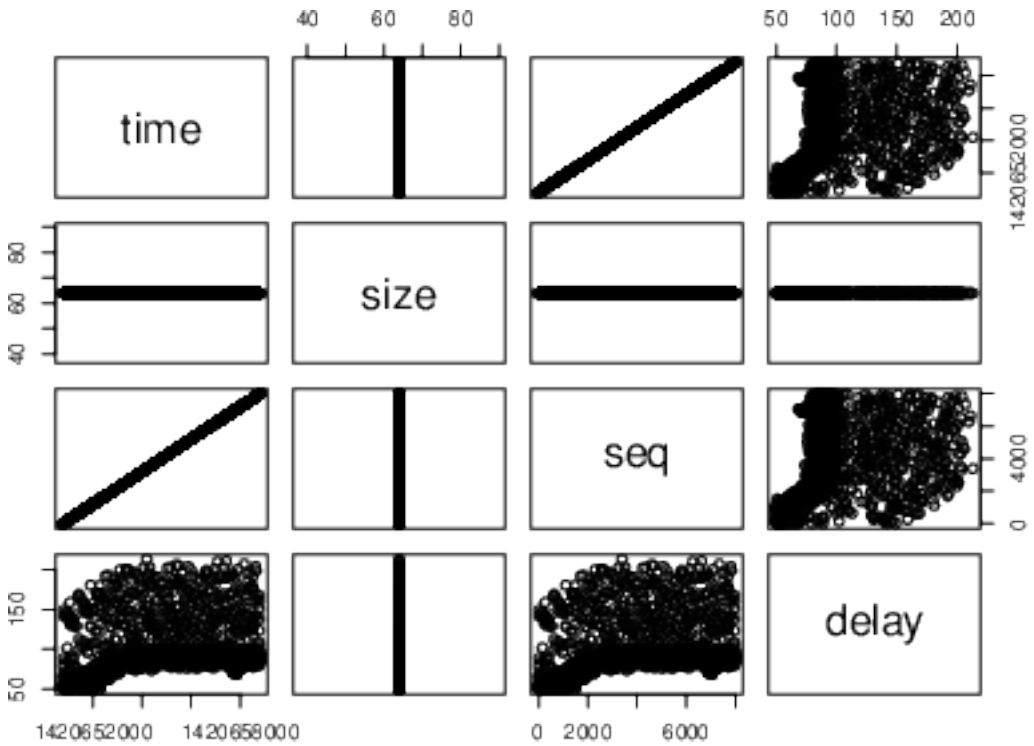


Il arrive qu'on ait des valeurs 2 à 3 fois plus grandes que les autres et ça arrive souvent en rafale. On a des séries de mesures qui peuvent être différentes du reste. Il faudra en tenir compte si on fait des expériences où certain paramètres (comme la taille) varient.

Regardons ce que donne le jeu de données vers google

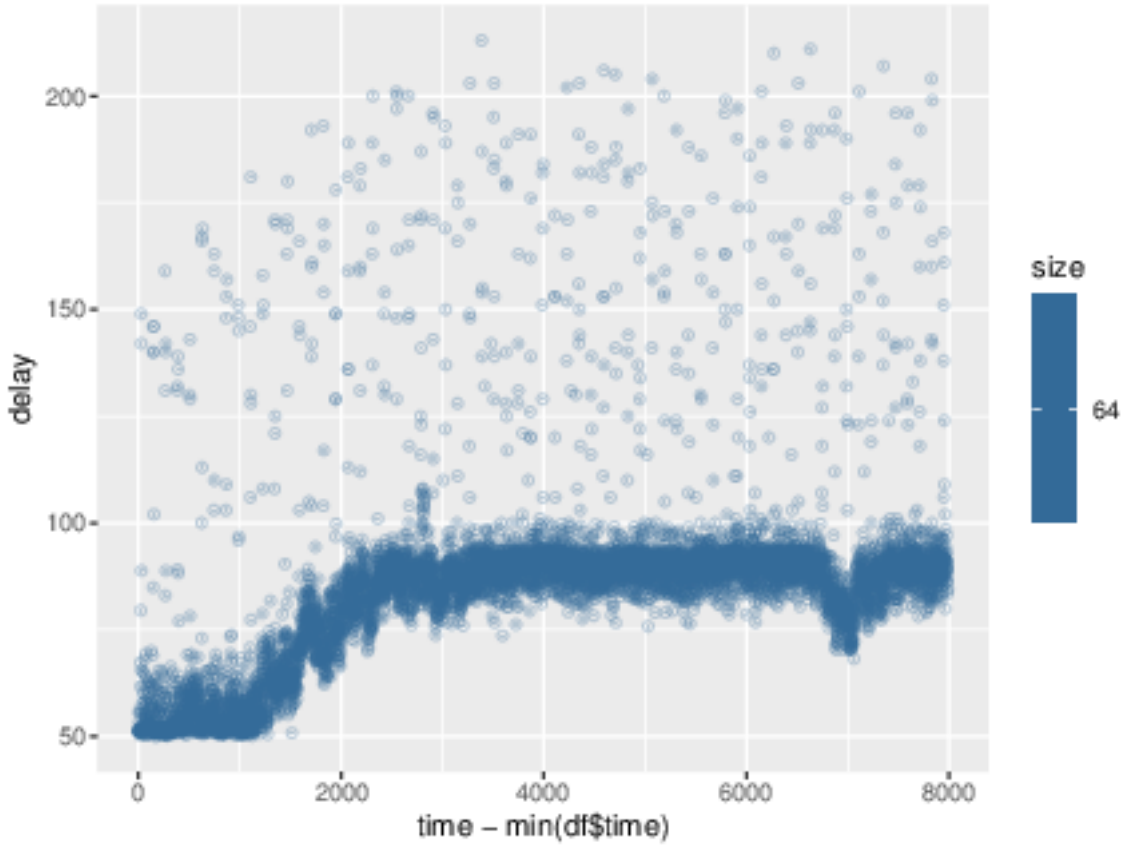
```
google<-read.csv("google.csv.gz",header=T);
df = google
```

```
plot(df[names(df)%in% c("time","size","seq","delay")]);
```



Encore une fois, une seule taille a été utilisée mais rien de surprenant de ce coté là. Wow, par contre, niveau delay, c'est très différent cette fois ci. Ça n'est plus du tout stable dans le temps. Le comportement évolue franchemet au cours du temps et, ce de façon relativement lisse apparemment. Regardons de plus près.

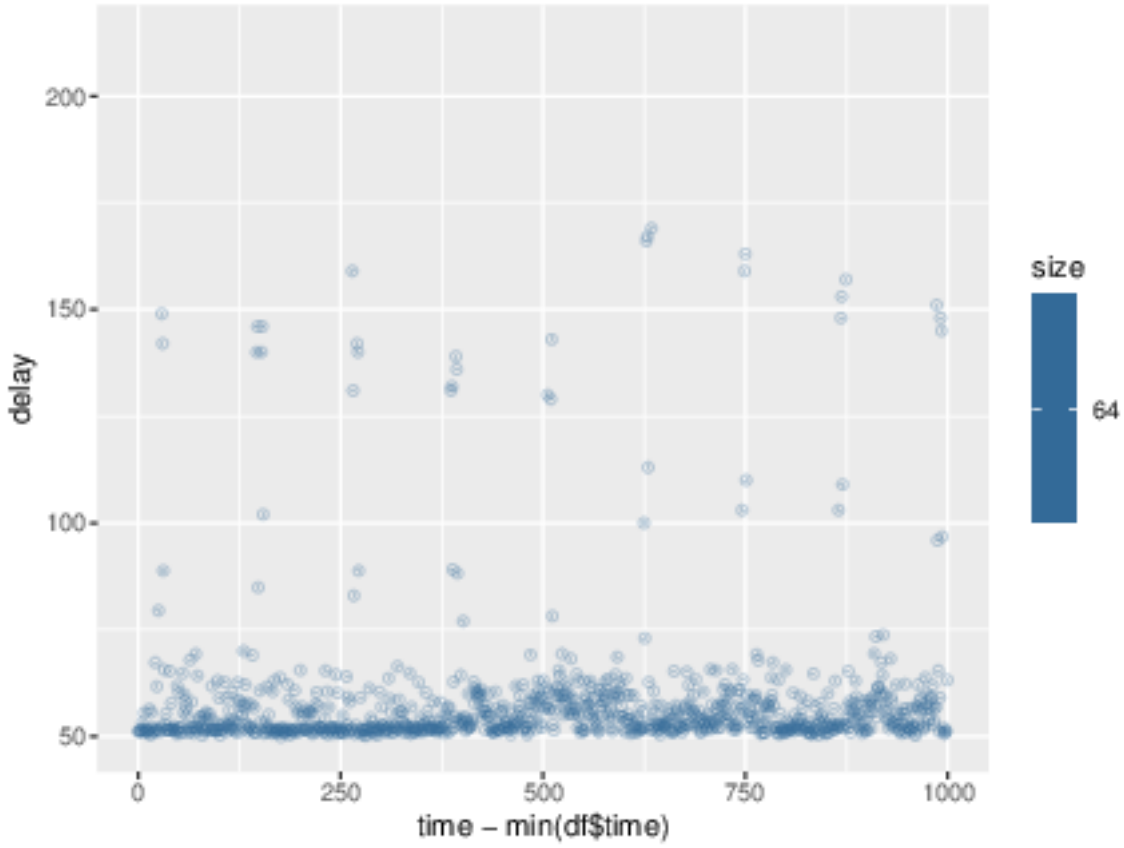
```
p = ggplot(data=df,aes(x=time-min(df$time),y=delay, color=size)) + geom_point(alpha=.3)
p
```



Ah, c'est lisse mais pas tant que ça. Il y a encore une autre échelle de temps qui fait apparaître des "dents" de scie. Regardons, d'encore plus près ?

```
p + xlim(0,1000)
```

```
## Warning: Removed 6970 rows containing missing values (geom_point).
```



Et encore une fois, il arrive qu'on ait des valeurs 2 à 3 fois plus grandes que les autres et ça arrive souvent en rafale. On a des séries de mesures qui peuvent être différentes du reste et il faudra en tenir compte si on fait des expériences où certain paramètres (comme la taille) varient.

Allons plus loin et étudions l'influence de la taille des données

Expériences

Le plus naturel est certainement de faire quelque chose comme ça:

```
while true ; do for i in 16 60 600 1600 ; do ping -c 2 -D liglab.imag.fr -s $i -i .2 | grep from; done
```

Seulement, vu qu'on peut avoir des rafales de mesures assez différentes des autres, on risque de ne pas arriver à savoir si les différences qu'on observe sont dûes à la taille ou à un bruit externe. Il vaut donc mieux randomiser les tailles de la façon suivante. L'approche suivante permet donc de se prémunir d'un certain nombre de biais et de collecter relativement rapidement des mesures (mettre le delay en dessous de .2 a souvent des conséquences facheuses et empêche de faire les mesures)

```
while true ; do \
    export j=$(($RANDOM%2000)) ; \
    ping -c 1 -D liglab.imag.fr -s $j  | grep from ; \
    sleep .2 ; \
done > /tmp/liglab2.log
```

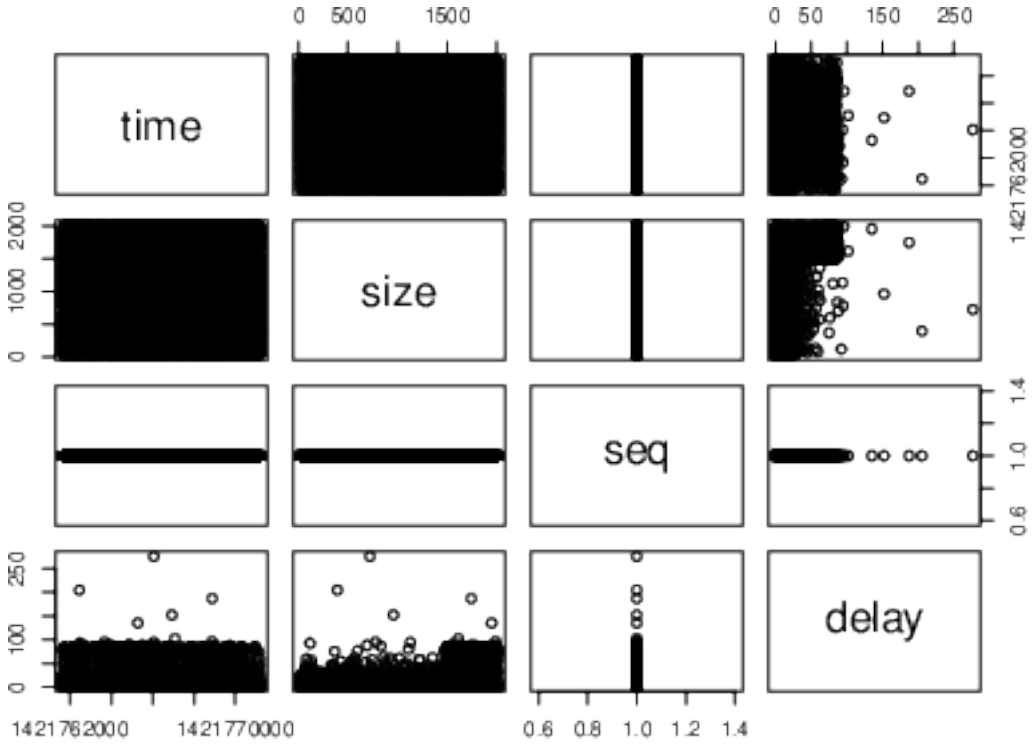
J'ai mis 2000 comme taille max car pour des valeurs vraiment plus grosses, je me faisais dropper quasiment à chaque fois. Et comme google ne voulait plus de mes paquets de taille non standard, j'ai cherché d'autres serveurs. J'ai gardé liglab et j'ai remplacé google par stackoverflow.

Analyse de liglab

```
liglab2<-read.csv("liglab2.csv.gz",header=T);
df = liglab2
```

Regardons tout ça:

```
plot(df[names(df)%in% c("time","size","seq","delay")]);
```



Que voit-on ?

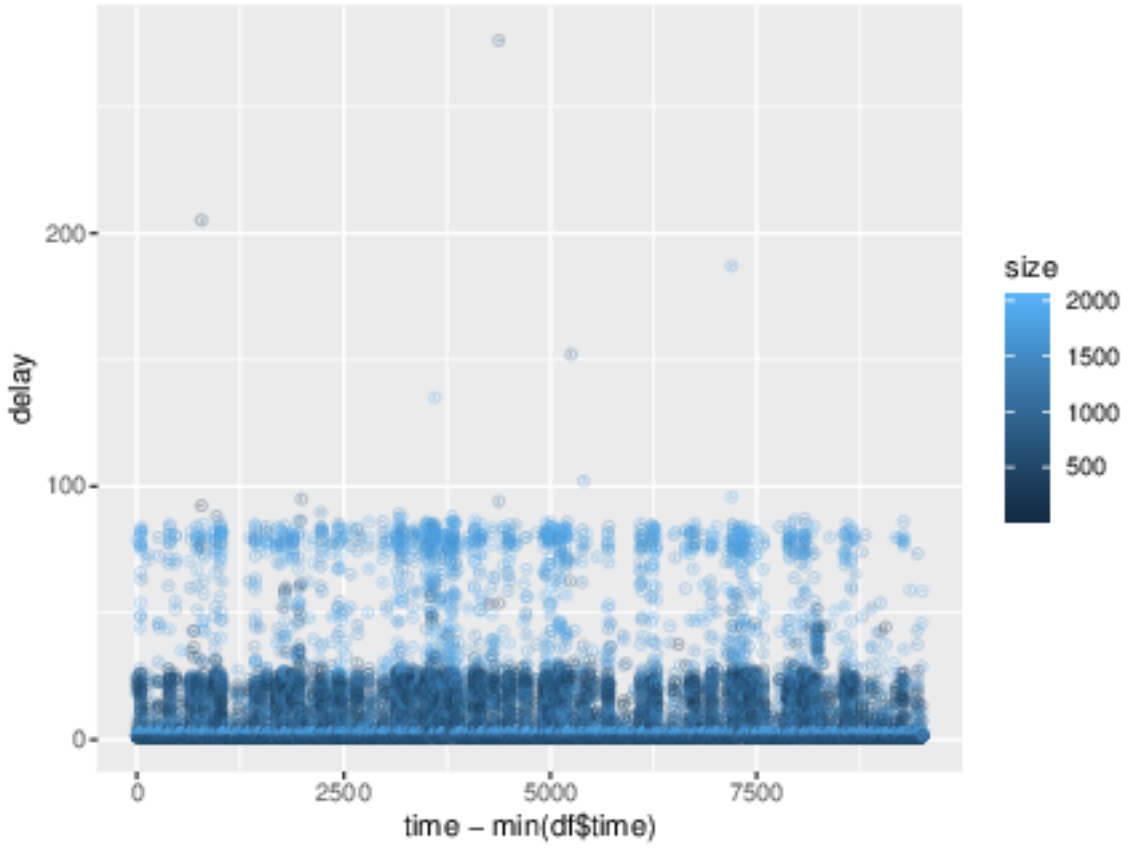
- size en fonction de time (graphe de coordonnées (2,1)): la size est bien indépendante de time. Pour bien faire, il faut vérifier que size est bien uniforme (on ne sait jamais)

  ```
  hist(df$size)
  ```

- delay en fonction de time (graphe de coordonnées (4,1)): le delay est vraiment très variable mais semble stable dans le temps. Il y a des points qui sortent vraiment du lôt (au delà de 100ms) mais ils sont peu nombreux.
- delay en fonction de size (graphe de coordonnées (4,2)): il y a une rupture. Au delà d'environ 1400, le comportement est très différent. Ceci mis à part, il y a une variabilité telle qu'il est difficile de dire si la size est significative...

Regardons ça de plus près

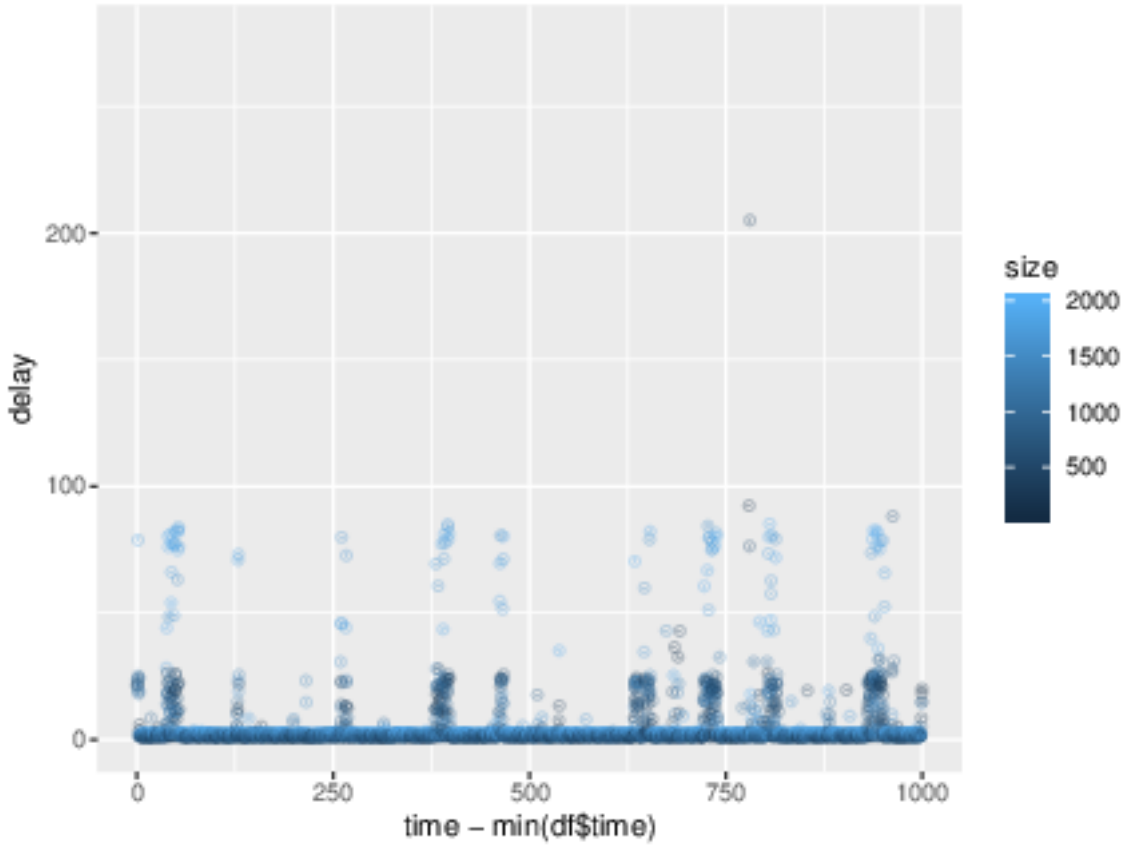
```
p = ggplot(data=df,aes(x=time-min(df$time),y=delay, color=size)) + geom_point(alpha=.3)
p
```



C'est à peu près stable dans le temps mais il y a encore une fois des paquets d'alignements verticaux. Ça va être conton à étudier. Je remarque aussi que les points clairs (ceux avec une size impotante) sont plutôt un peu plus vers le haut que les autres. Mais encore une fois avec une telle variabilité, ça va être difficile à mettre en évidence.

```
p + xlim(0,1000)
```

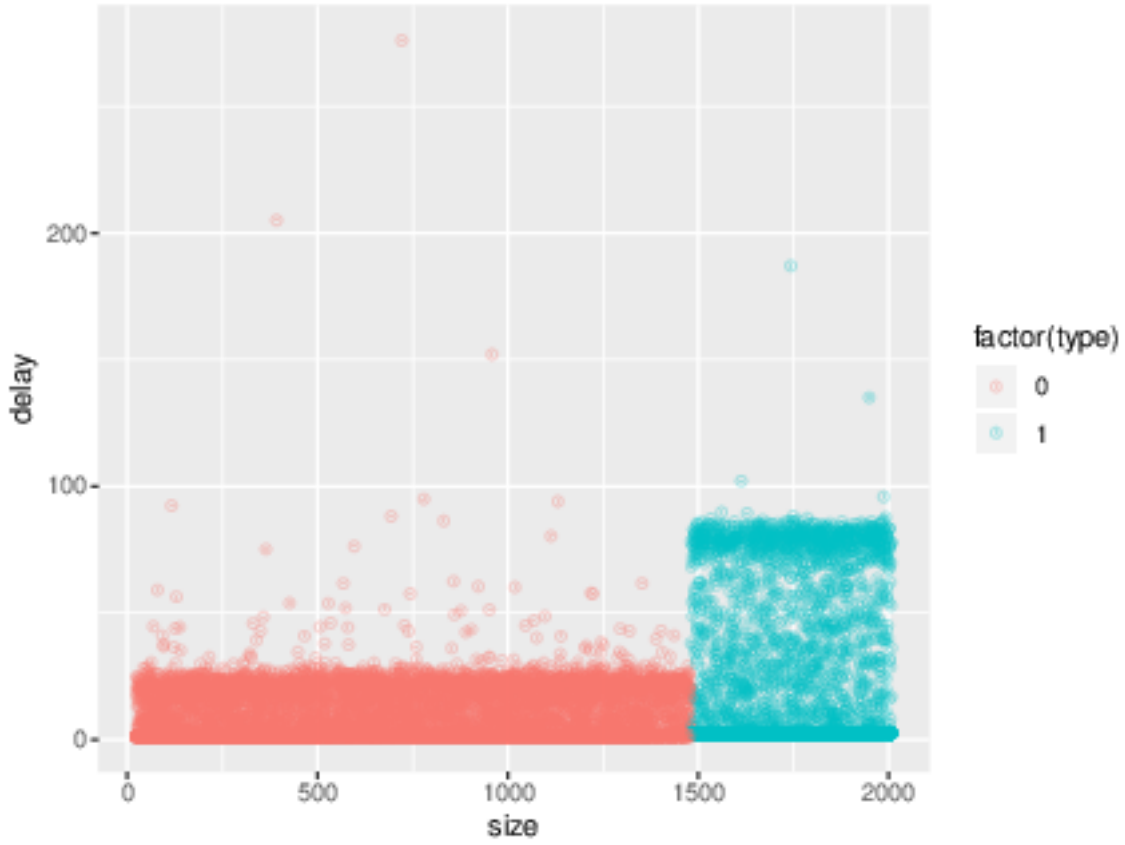
```
## Warning: Removed 39263 rows containing missing values (geom_point).
```



Ah oui, là, on voit bien les rafales et il n'y a pas de raison qu'elles correspondent à une taille particulière puisque la taille est randomisée. Elles correspondent donc bien à des perturbations du réseau locales dans le temps.

Bon, essayons déjà de séparer les "gros" des "petits" paquets.

```
df$type=0;
df[df$size>1480,]$type=1;
ggplot(data=df,aes(x=size,y=delay, color=factor(type))) + geom_point(alpha=.3)
```



Ainsi, on peut les traiter à part lors de l'analyse.

Je me souviens quand je faisais des mesures pour des petites tailles (64) et pour des tailles plus grosses (1000), je trouvais qu'on voyait la différence. Est-ce que c'est détectable.

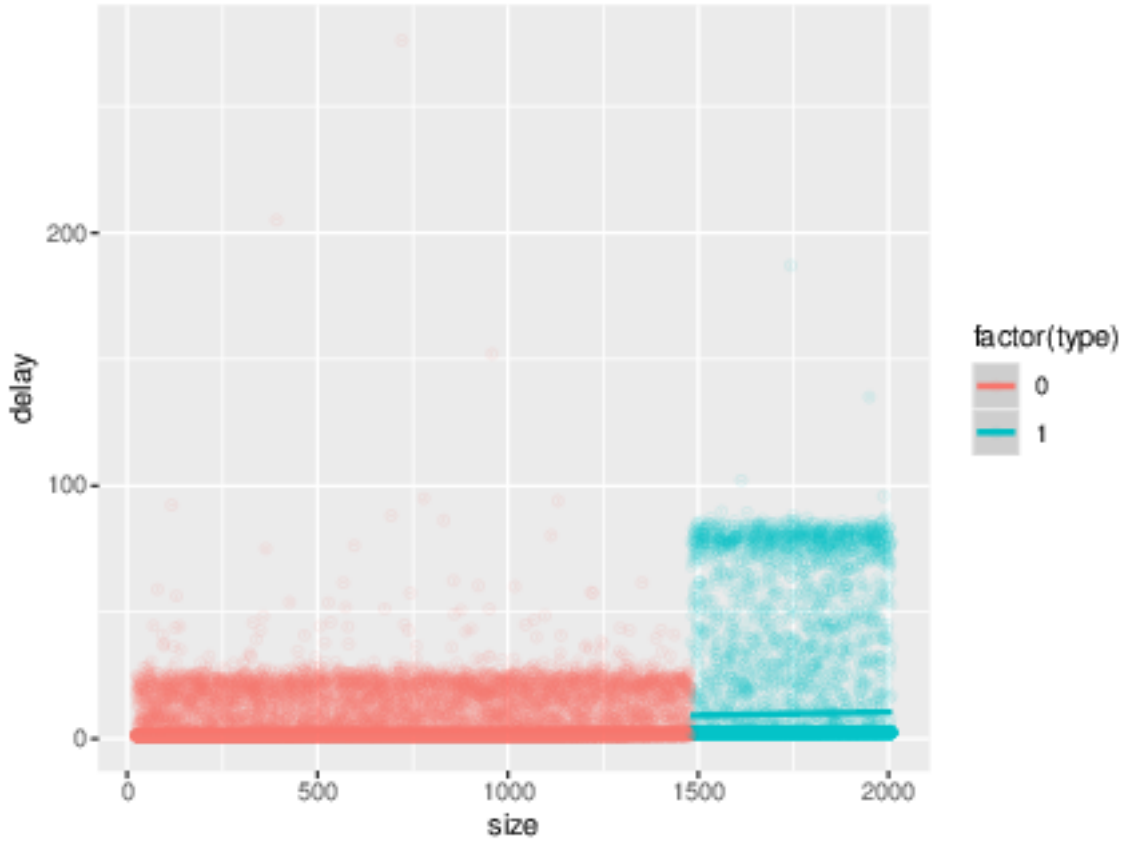
```
summary(lm(data=df[df$type==0,],delay~size))
```

```
##
## Call:
## lm(formula = delay ~ size, data = df[df$type == 0, ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -2.475  -2.246  -2.189  -2.087 272.490
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.275e+00  7.230e-02  45.295  < 2e-16 ***
## size        3.266e-04  8.496e-05   3.844 0.000121 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.431 on 32664 degrees of freedom
## Multiple R-squared:  0.0004522,  Adjusted R-squared:  0.0004216
## F-statistic: 14.78 on 1 and 32664 DF,  p-value: 0.0001213
```

Argh, si l'intercept et le coefficient pour size sont significativement différents de 0, on a un R^2 proche de 0. Il y a une variabilité énorme qu'on n'arrive pas à expliquer juste avec size. En même temps on le savait.

Mettons la prédiction de la régression en regard des mesures.

```
ggplot(data=df,aes(x=size,y=delay, color=factor(type))) + geom_point(alpha=.1) + geom_smooth(method="lm
```



Forcément avec autant de variabilité et une variabilité aussi asymétrique, notre régression est “trop haute”. Moralement, on comprend bien qu’il y a un temps de propagation sur les liens et un temps d’attente dans les buffers et c’est cette dernière composante qui cause toute la variabilité et nous empêche de mesurer le temps de propagation sur les liens. Mais de temps, en temps, on arrive à ne pas faire la queue, nos paquets atteignent leur cible sans trop attendre. Du coup, une idée pourrait être de s’intéresser au bas de la courbe, aux plus petites valeurs...

Allons-y:

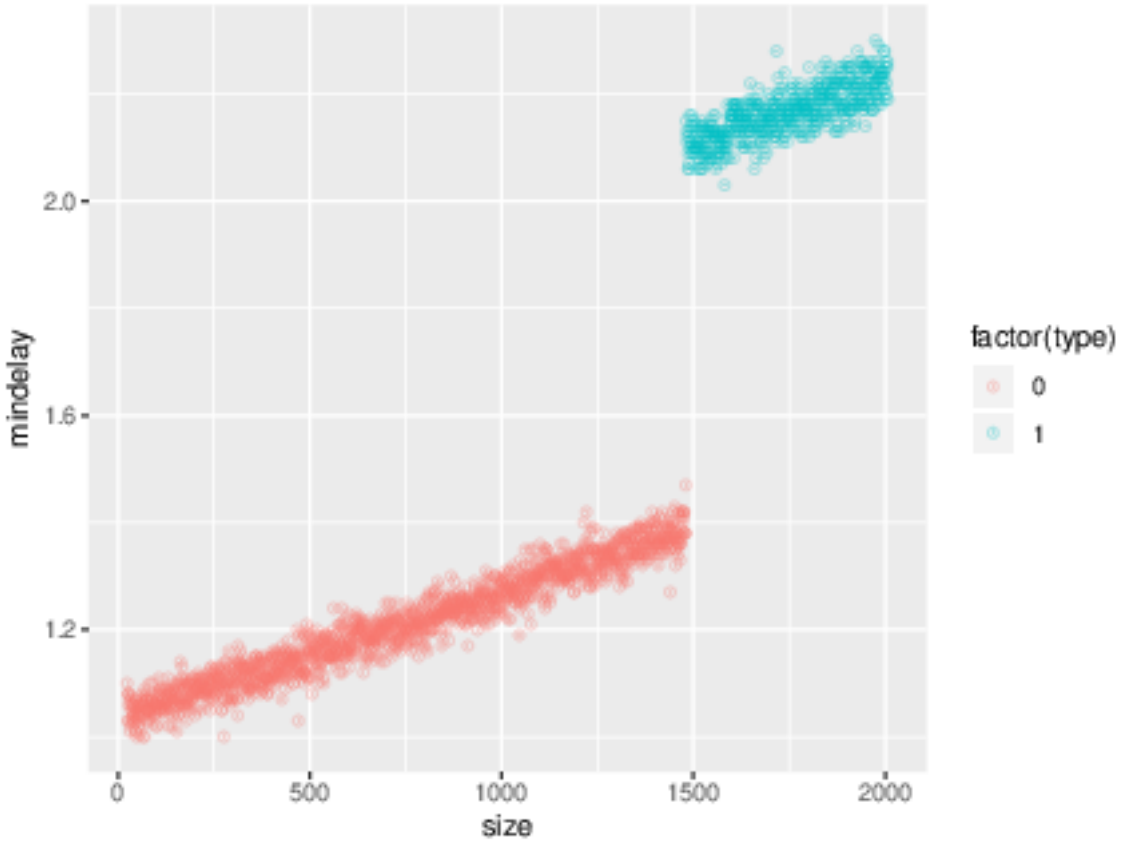
```
d = data.frame(size=c(),mindelay=c(),num=c())
for(s in unique(df$size)) {
  d = rbind(d,data.frame(size=s,mindelay=min(df[df$size==s,]$delay),
                         num=length(df[df$size==s,]$delay),
                         type=unique(df[df$size==s,]$type)))
}
```

Mais c’est lent... :) Si vous voulez faire sans boucle, la bonne façon de faire est d’utiliser plyr:

```
library(plyr)
d = ddply(df,c("size"), summarize, mindelay=min(delay),num=length(delay),type=unique(type))
```

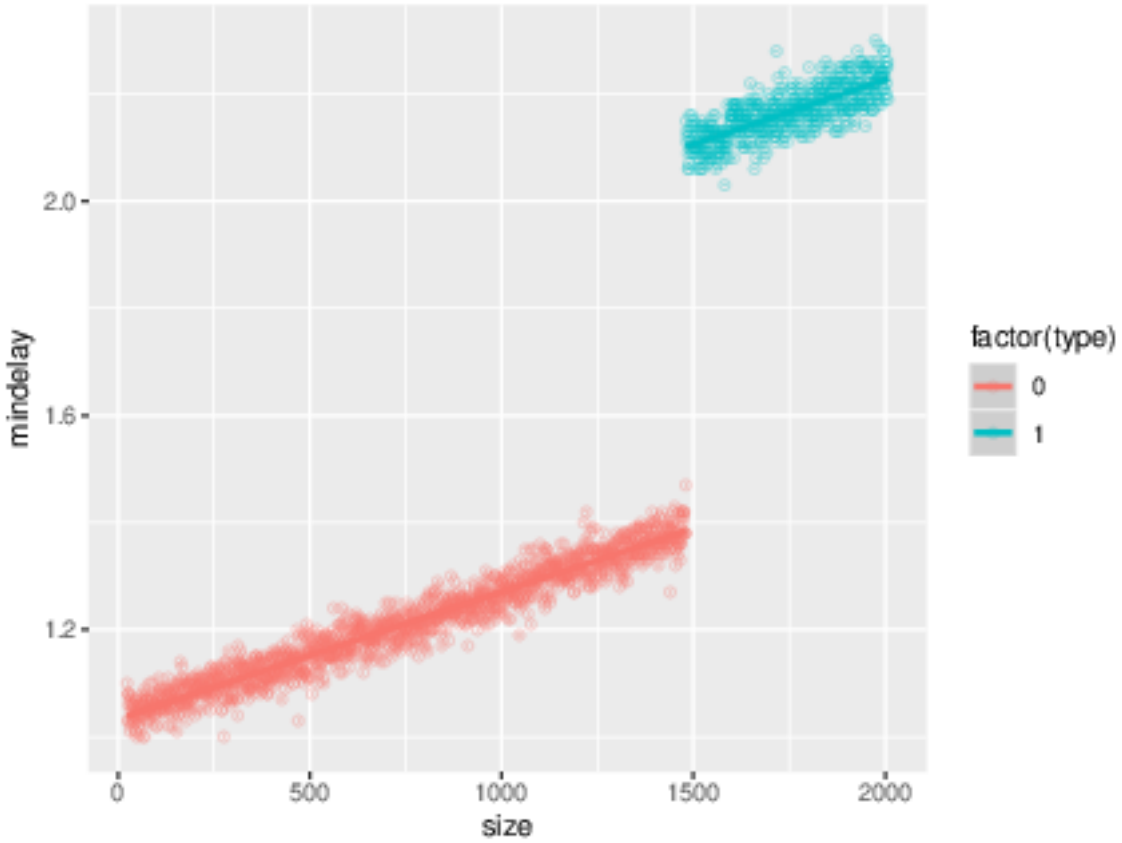
Bref, une fois que c’est fait, voilà ce qu’on obtient:

```
ggplot(data=d,aes(x=size,y=mindelay, color=factor(type))) + geom_point(alpha=.3)
```



Et ça, ça parait déjà, plus facile à étudier! :) Rajoutons la régression linéaire à ces points:

```
ggplot(data=d,aes(x=size,y=mindelay, color=factor(type))) + geom_point(alpha=.3) +
      geom_smooth(method="lm")
```



Nickel! Regardons quand même ce que disent les indicateurs des régressions linéaires:

```
summary(lm(data=d[d$type==0,],mindelay~size))
```

```
##
## Call:
## lm(formula = mindelay ~ size, data = d[d$type == 0, ])
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.114824 -0.014771  0.000258  0.016651  0.097126
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.033e+00  1.351e-03   764.6   <2e-16 ***
## size        2.371e-04  1.569e-06   151.2   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02518 on 1455 degrees of freedom
## Multiple R-squared:  0.9401, Adjusted R-squared:  0.9401
## F-statistic: 2.285e+04 on 1 and 1455 DF,  p-value: < 2.2e-16
```

Joie et félicité. Deux paramètres bien significatifs avec une très faible variance, et un R^2 de 0.94! Et maintenant, encore plus fort:

```
l = lm(data=d,mindelay~size*factor(type))
summary(l)
```

```
##
## Call:
## lm(formula = mindelay ~ size * factor(type), data = d)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.11482 -0.01658  0.00032  0.01811  0.12165
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)         1.033e+00  1.445e-03 715.067   <2e-16 ***
## size                2.371e-04  1.677e-06 141.354   <2e-16 ***
## factor(type)1       7.098e-01  1.357e-02  52.291   <2e-16 ***
## size:factor(type)1 4.822e-06  7.891e-06   0.611    0.541
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02693 on 1980 degrees of freedom
## Multiple R-squared:  0.9961, Adjusted R-squared:  0.9961
## F-statistic: 1.687e+05 on 3 and 1980 DF,  p-value: < 2.2e-16
```

Je prend soin de préciser que type doit être considéré comme un facteur et pas comme un nombre. Et là, c'est très intéressant (attention, c'est un peu fin). Non seulement le R^2 est très faible (nos valeurs sont plus grandes quand on considère les gros paquets mais la variabilité n'a pas changé...) mais on voit que non seulement intercept, size et factor(type) sont significatifs et très précis mais en plus que size:factor(type) n'est absolument pas significatif. En gros, ça veut dire que la pente pour les petits et pour les gros paquets est la même...

Si j'en crois ma régression, j'arrive donc à transmettre à une vitesse (en o/ms, c'est à dire en Ko/s) de:

```
1/l[1]$coefficients[2]
```

```
##     size
## 4217.893
```

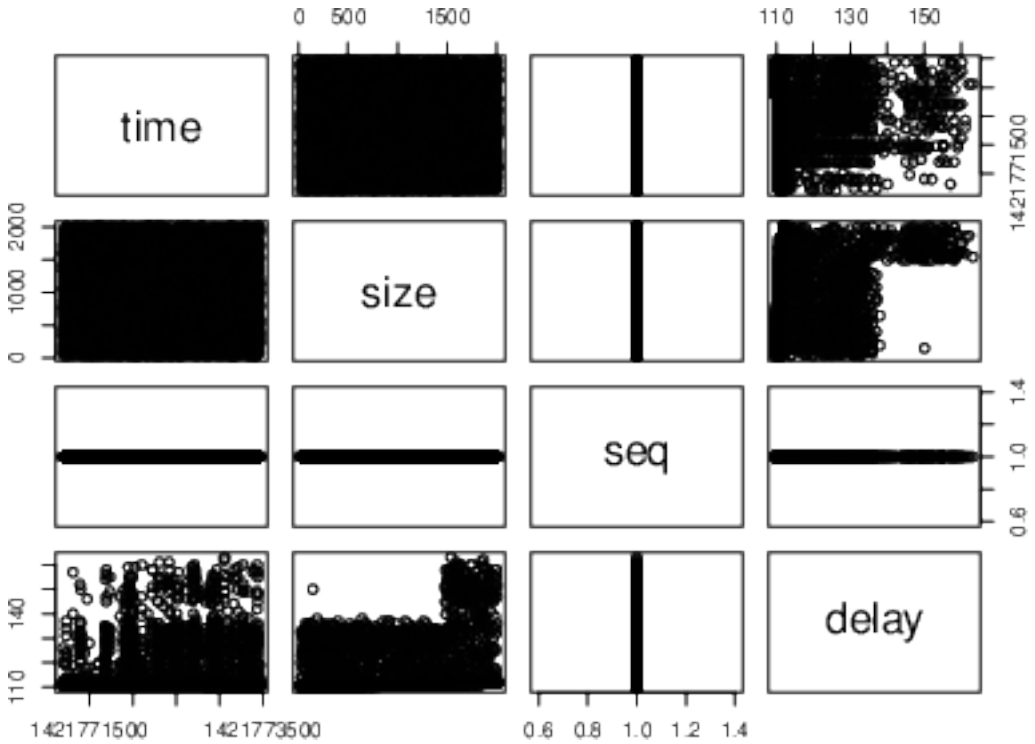
Analyse de stackoverflow

Bon, tentons de faire pareil avec stackoverflow.

```
  stack<-read.csv("stackoverflow.csv.gz",header=T);
  df = stack
```

Regardons tout ça:

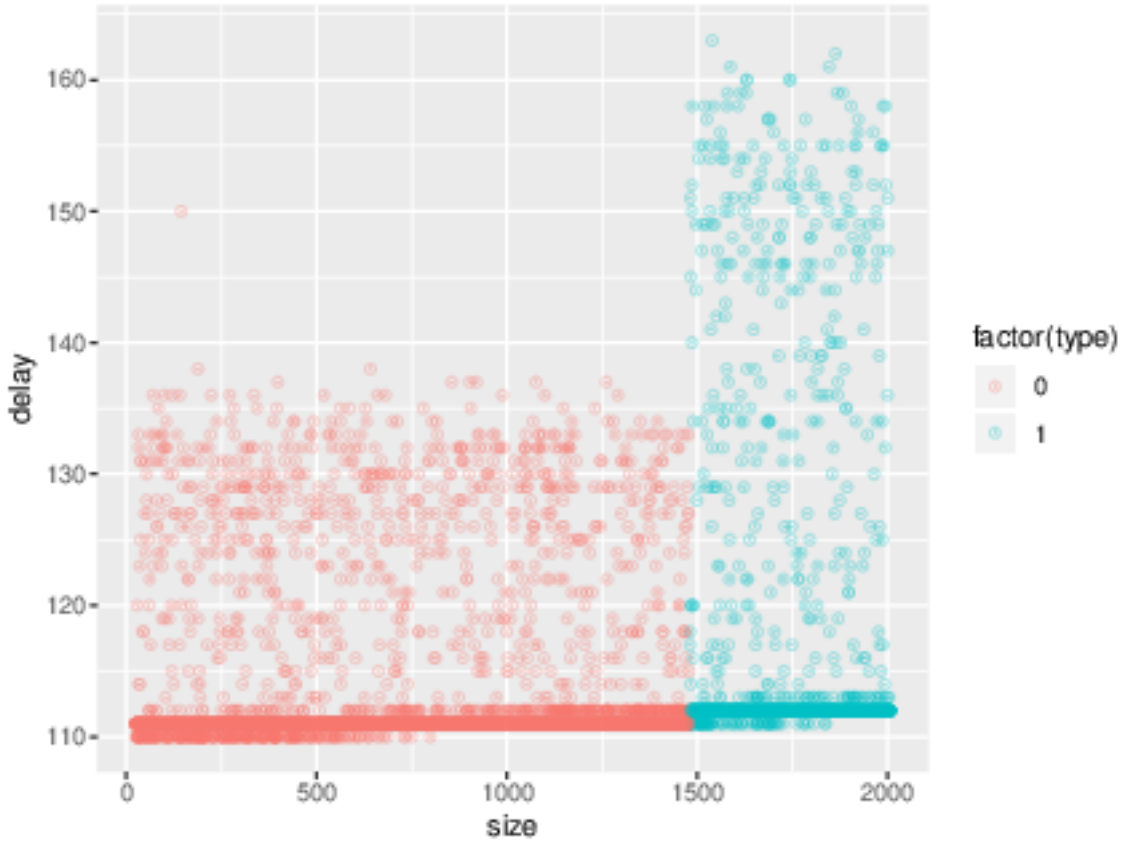
```
plot(df[names(df)%in% c("time","size","seq","delay")]);
```



Que voit-on ? * size ~ time : Indépendance de time et de size, c'est normal. * delay ~ time : Aouch, une fois encore il y a une variabilité importante et des moments où ça se passe bien plus mal qu'à d'autres. * delay ~ size : encore une fois, on retrouve ce phénomène des petits et des gros paquets.

Bon, essayons déjà de séparer les "gros" des "petits" paquets.

```
df$type=0;
df[df$size>1480,]$type=1;
ggplot(data=df,aes(x=size,y=delay, color=factor(type))) + geom_point(alpha=.3)
```



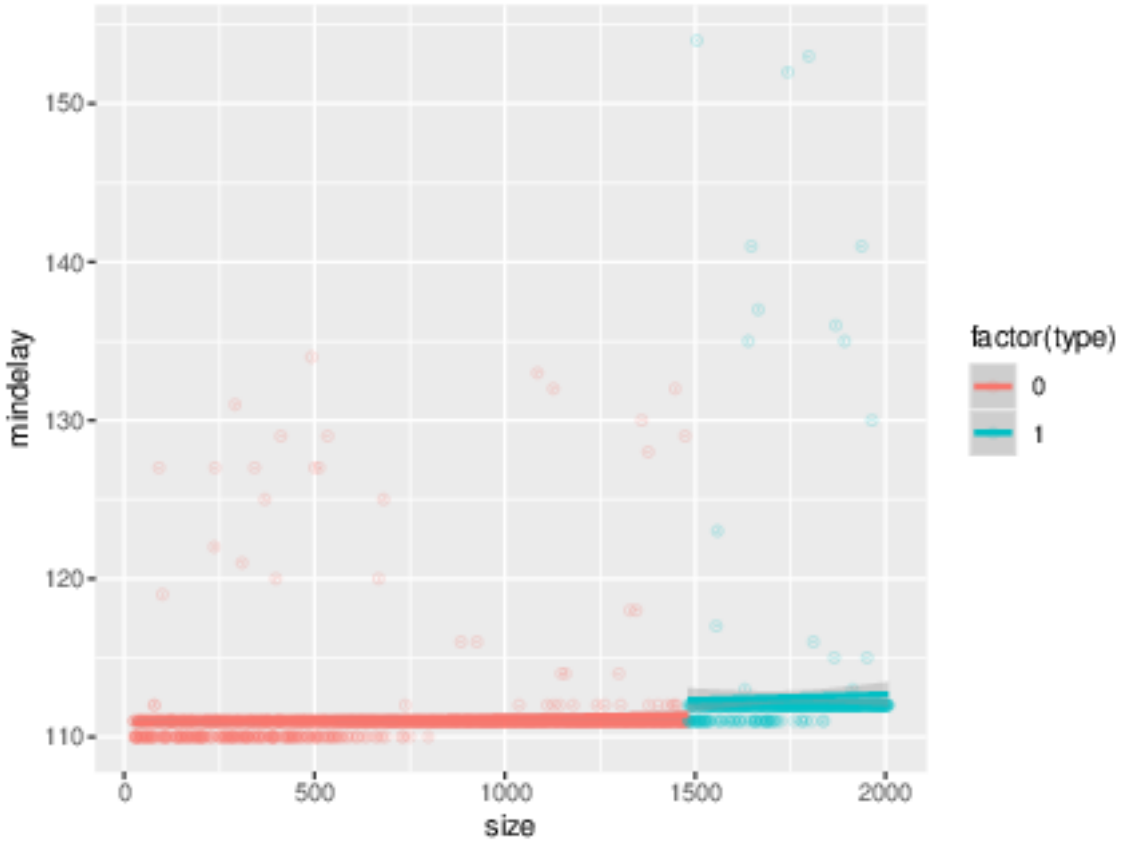
On arrive bien à séparer mais par contre, la variabilité est bizarre. La majorité des points sont à 110, 111, 112 ms. C'est beaucoup plus gros que ce que l'on avait tout à l'heure.... Si je prend les min comme j'avais fait avant, je risque de ne pas m'en sortir et il faudrait probablement utiliser une autre technique (régression de quantiles ?). :(

Mais bon, allons quand même jusqu'au bout pour voir ce que ça donne.

```
d = ddply(df,c("size"), summarize, mindelay=min(delay),num=length(delay),type=unique(type))
```

Bref, une fois que c'est fait, voilà ce qu'on obtient:

```
ggplot(data=d,aes(x=size,y=mindelay, color=factor(type))) + geom_point(alpha=.2) +
      geom_smooth(method="lm")
```



Mouif, en fait, c'est rigolo, là, on ne voit plus la distinction entre les gros et les petits (mis à part la couleur).

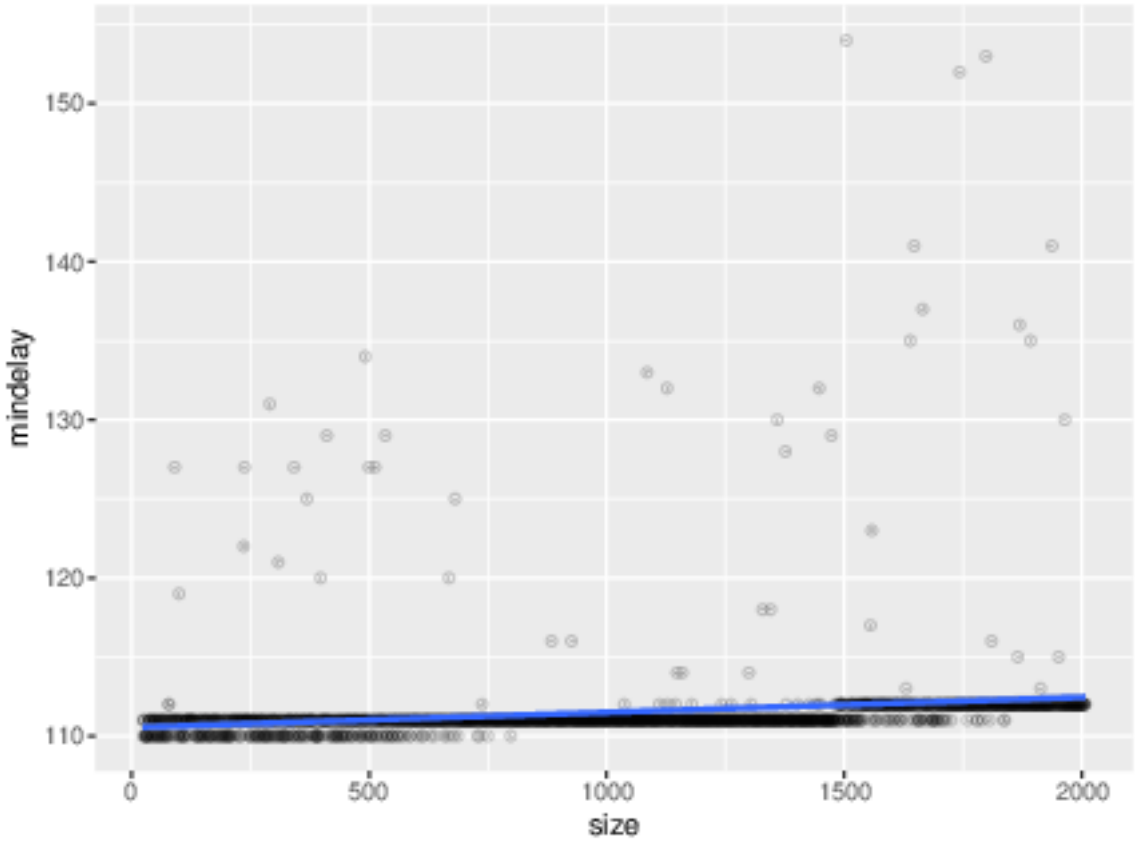
Regardons quand même ce que disent les régressions linéaires:

```
summary(lm(data=d[d$type==0,],mindelay~size))
```

```
##
## Call:
## lm(formula = mindelay ~ size, data = d[d$type == 0, ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1628 -0.3431 -0.2096 -0.0669 22.9453
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.109e+02  1.155e-01 959.887  < 2e-16 ***
## size        3.531e-04  1.341e-04   2.633  0.00856 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.126 on 1408 degrees of freedom
## Multiple R-squared:  0.004899,   Adjusted R-squared:  0.004192
## F-statistic: 6.931 on 1 and 1408 DF,  p-value: 0.008562
```

Bon, avec un R^2 de 0.004 on n'explique rien. On pourrait même ne plus considérer le type:

```r
ggplot(data=d,aes(x=size,y=mindelay)) + geom_point(alpha=.2) +
      geom_smooth(method="lm")
```




```r
l=lm(data=d,mindelay~size)
summary(l)
```

```
##
## Call:
## lm(formula = mindelay ~ size, data = d)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -1.301 -0.677 -0.341 -0.074 42.026
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.105e+02  1.339e-01 825.715   <2e-16 ***
## size        9.535e-04  1.151e-04   8.288   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.886 on 1914 degrees of freedom
## Multiple R-squared:  0.03464,    Adjusted R-squared:  0.03414
## F-statistic: 68.69 on 1 and 1914 DF,  p-value: < 2.2e-16
```

```r
1/l[1]$coefficients[2]
```

```
##     size
## 1048.731
```

De là à accorder du crédit à cette valeur… Franchement, je ne sais pas. On n'a vraiment fait qu'effleurer le problème. Pas facile de réduire la variance et d'avoir un modèle raisonnable pour tout ceci.