Simulation Task One

In [23]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import os,sys
```

In [3]:
```
closed = pd.read_csv("ClientClosedReduced.txt", \
            names=['time', 'resp', 'proxy', 'router', 'originA', 'o
riginB'])
print("closed has", len(closed.index), "rows")
closed[:5]
```

```
closed has 9959 rows
```

Out[3]:

	time	resp	proxy	router	originA	originB
0	10.046999	0.298637	0.627693	0.147484	0.901917	0.623723
1	20.101925	0.266878	0.535165	0.125060	0.988784	0.589115
2	30.129184	0.329273	0.551071	0.128381	0.993454	0.632332
3	40.205962	0.283763	0.547957	0.128278	0.995366	0.636136
4	50.249577	0.249656	0.555504	0.127985	0.996458	0.620764

In [53]:
```
closed[9900:9905]
```

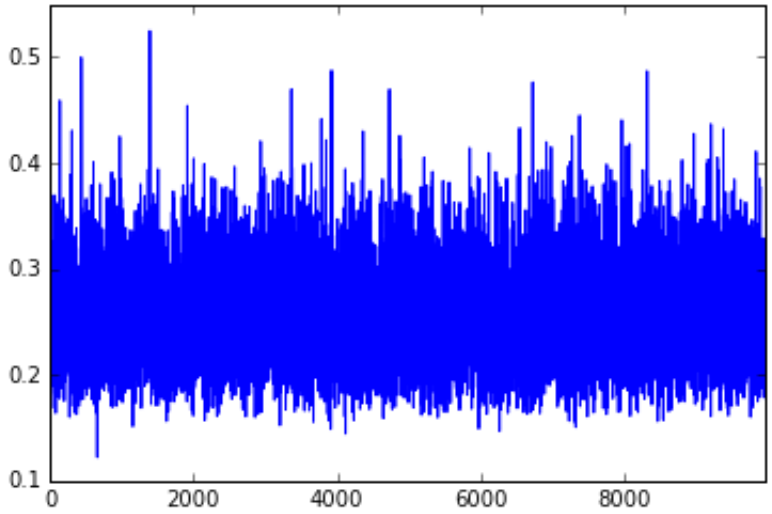
Out[53]:

	time	resp	proxy	router	originA	originB
9900	99441.796054	0.233496	0.568118	0.131151	0.984446	0.654708
9901	99451.822899	0.176960	0.568117	0.131152	0.984446	0.654715
9902	99461.853184	0.177911	0.568124	0.131152	0.984446	0.654723
9903	99471.896674	0.212076	0.568130	0.131155	0.984448	0.654736
9904	99481.957585	0.216220	0.568134	0.131157	0.984450	0.654742

Now, lets plot the response time and the utilization for one component

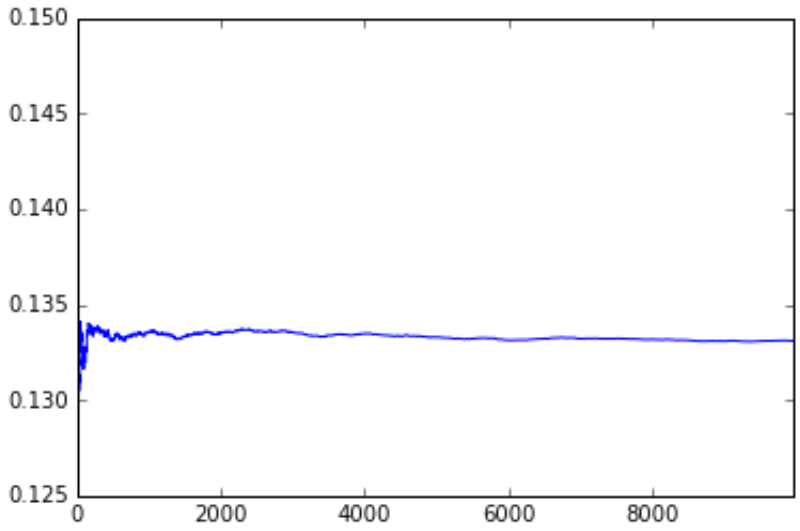
In [5]:
```
closed['resp'].plot()
```

Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x109f33c50>



In [9]:
```
closed['router'].plot()
```

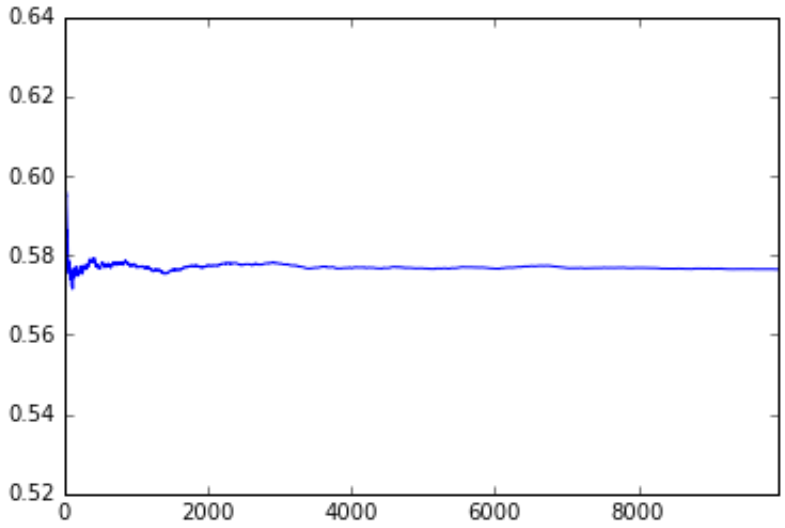
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x10b4a3128>



It's clear there's a transient for the first ~40,000 seconds (recall that we're looking at the data divided into 10 second averages)

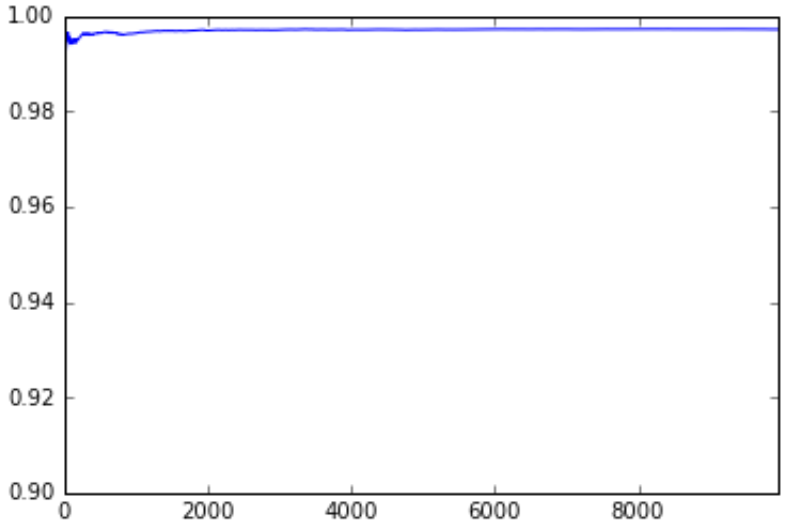
In [10]:
```
closed['proxy'].plot()
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x10b4bc588>



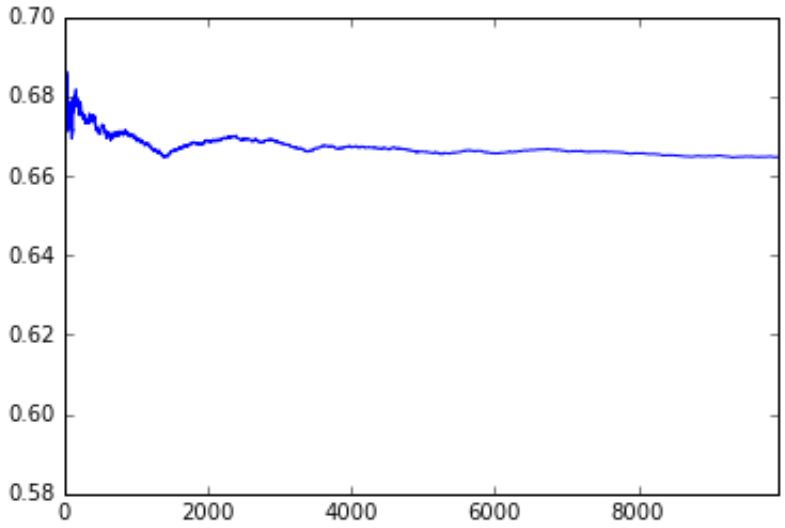
In [11]:
```
closed['originA'].plot()
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x11654e710>



In [12]:
```
closed['originB'].plot()
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x112636898>



Simulation Task Two

#1.1 Answer: I did this twice two different ways because I was uncertain of my first results.

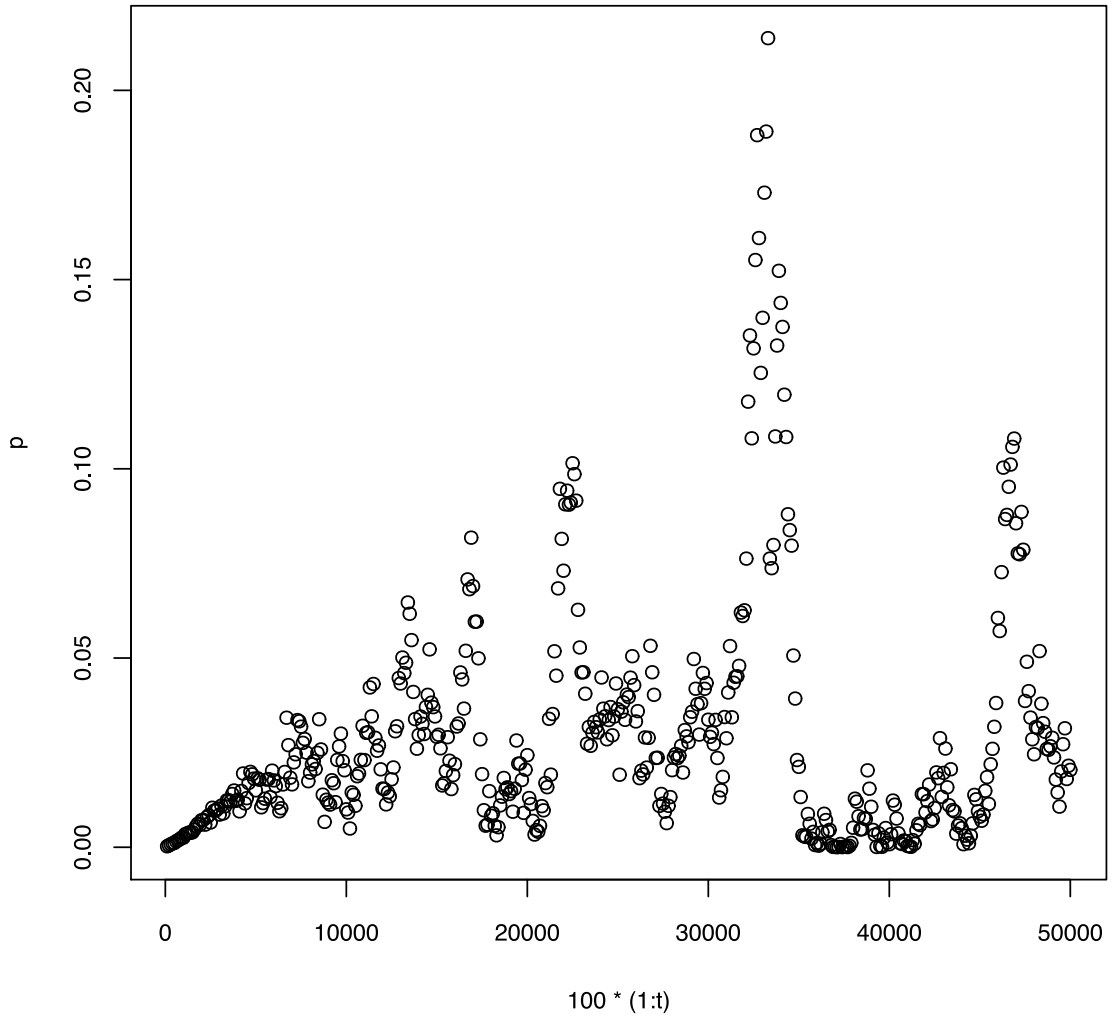
First way: Using the method for the lecture slides, I started with the 100 second batch. Then I grouped thoses batches to get data in batch sizes that were multiples of 100 seconds. For each of those data series, I calculated the variance. The batch size time that produced the maximum variance on the data series is the time of the transient. I found the transient to occur at 33300 seconds.

Second way: But I wasn't sure if the average of averages was acceptable. So I redid it using MovingAverage file to actually calculate the batch mean. IT gave me the same result of 33000 seconds.

In [11]:
```
collect <- function(x, n){
    m = length(x)/n
    r = c()
    for (i in 1:m){
        r = append(r, sum(x[(n*(i-1) + 1 ):(n*i)]))
    }
    r
}

batch100 = read.csv("batch100.csv")$resp
t = 500
p = c()
for (j in 1:t){
    p = append(p, var(collect(batch100, j)))
}
plot(100*(1:t), p)
100*(1:t)[which.max(p)]
```

Out[11]: 33300



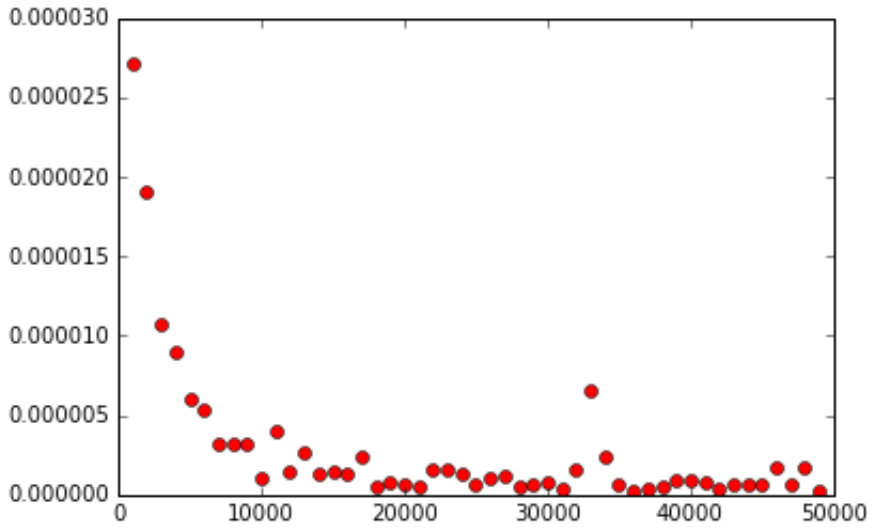
In [31]:
```
from MovingAverage import batchAverage
from statistics import variance

def batchVar(timeWindow):
    if timeWindow % 10000 == 0:
        print("Progress...")
    file = open("ClientClosedOutput.csv")
    avgs = []
    while file:
        try:
            avg = [ f for f in batchAverage(file, timeWindow) ]
            avgs.append(avg[1]) #Index of 1 is resp time
        except Exception as err:
            break
        #variances = [variance([avgs[j][i] for j in range(len(avg
s))]) for i in range(len(cols))]
    return variance(avgs)

times = range(1000, 50000, 1000)
result = [batchVar(i) for i in times]
plt.plot(times, result, 'ro')
```

```
Progress...
Progress...
Progress...
Progress...
```

Out[31]:
```
[<matplotlib.lines.Line2D at 0x1126e8fd0>]
```



In [44]:
```
times[result.index(max(result[20:50]))]
```

Out[44]:
```
33000
```

#1.2 Answer: Using the method for the lecture slides, I started with a batch size of 100 seconds. Then I used MovingAverage.py to calculated batch sizes for powers of two times 100 seconds. For each of those batches, I calculated the covariance of the data series and the previous batch size data series. I divide this by the variance of the original data series. This help me find which batch size produces the smallest covariance with respect to variance. The smallest one is the optimal batch size. I found the optimal batch size to be 3200 seconds.

In [68]:
```
batch100 = read.csv("batch100.csv")$resp
batch200 = read.csv("batch200.csv")$resp
batch400 = read.csv("batch400.csv")$resp
batch800 = read.csv("batch800.csv")$resp
batch1600 = read.csv("batch1600.csv")$resp
batch3200 = read.csv("batch3200.csv")$resp
batch6400 = read.csv("batch6400.csv")$resp
cov(batch100[1:length(batch200)], batch200)/var(batch100)
cov(batch200[1:length(batch400)], batch400)/var(batch200)
cov(batch400[1:length(batch800)], batch800)/var(batch400)
cov(batch800[1:length(batch1600)], batch1600)/var(batch800)
cov(batch1600[1:length(batch3200)], batch3200)/var(batch1600)
cov(batch3200[1:length(batch6400)], batch6400)/var(batch3200)
```

Out[68]: -0.0122439760785554

Out[68]: -0.0419037282834467

Out[68]: -0.108266564114926

Out[68]: -0.101290067724043

Out[68]: -0.19538483120965

Out[68]: -0.000195386844430711

#1.3 Answer:

In [20]:
```
b = read.csv("batch3200.csv")
util = data.frame(mean(b$proxy), mean(b$router), mean(b$originA),
mean(b$originB))
n = length(b$proxy)
util = data.frame(b$proxy[n], b$router[n], b$originA[n], b$originB
[n])
utilc = qt(.975, n-2)*data.frame(var(b$proxy), var(b$router), var
(b$originA), var(b$originB))
util - utilc
util + utilc
```

Out[20]:

	b.proxy.n.	b.router.n.	b.originA.n.	b.originB.n.
1	0.5764596	0.1331009	0.9971586	0.664703

Out[20]:

	b.proxy.n.	b.router.n.	b.originA.n.	b.originB.n.
1	0.5764605	0.133101	0.9971594	0.6647232

#2 Answer: It takes a minimun of 25 jobs for the utilization of originA in the closed network to reach .9999 at steady-state. The last five measurements that I got were:

99999.898504887293, 0.013542259028, 0.5774258015, 0.13342318034, 0.99994001489, 0.666785676754

99999.957761095022, 0.412568491488, 0.57742588973, 0.133423115249, 0.999940922364, 0.666785281642

99999.97128515482, 0.046377573763, 0.577425946879, 0.133423097204, 0.999940787131, 0.666785191466

99999.972006946582, 0.045708794268, 0.577425949929, 0.133423096241, 0.999940779913, 0.666785186653

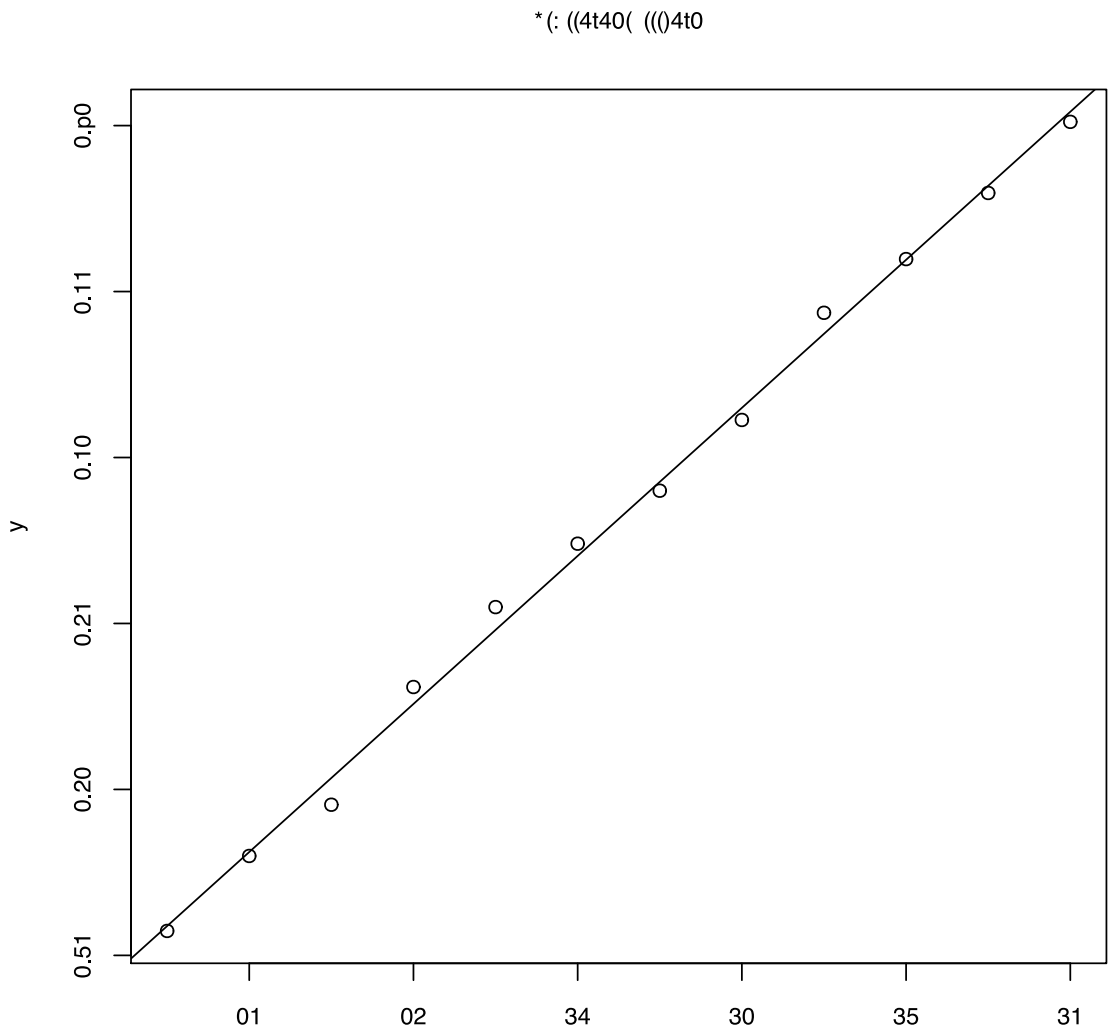
99999.975642919126, 0.048066143527, 0.577425965294, 0.133423150388, 0.999940743556, 0.666785162409

To calculate the plot below, I reran my simulation with the numJobs input as 25:36 and the sim time up to 50000 seconds because that's past the transient. Then I ran MovingAverage.py with time window 3200 seconds which is the batch mean I derived before. Then I plotted the last response time with respect to numJobs to make the plot below:

In [8]:
```
library(UsingR)
numJobs = c(25:36)
times = c(0.35736392434838193, 0.3799532965845909,
        0.3953908822645109, 0.43085228069522113,
        0.454930445140665, 0.47403588282719844,
        0.4899976667713914, 0.5113117856947006,
        0.5435983748031128, 0.5597807002140688,
        0.5796977483619459, 0.6011262416582664)
simple.lm(numJobs, times)
```

```
Out[8]: Call:
        lm(formula = y ~ x)

        Coefficients:
        (Intercept)            x
            -0.1987       0.0223
```



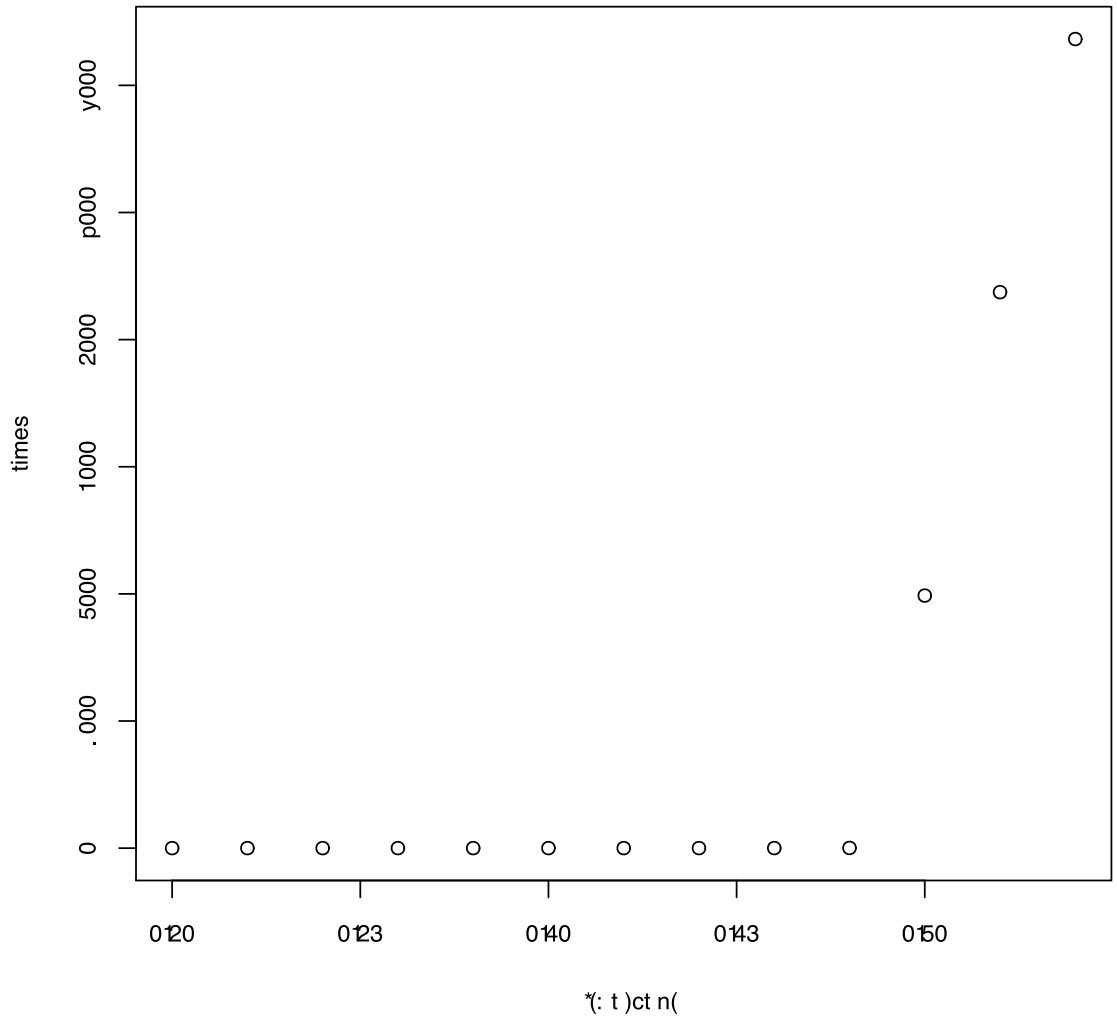
#3 Answer: It takes an interarrival time of 15ms for the utilization of originA in the closed network to reach .9999 at steady-state.

#4 Answer: To generate the plot below, I used the open network with interarrival time of 15ms. Dr. Grunwald said that this was okay on moodle: "If you did this using the open simulation, you can go ahead and use those numbers, but indicate that in the solution." I reran my simulation with the sim time up to 50000 seconds because that's past the transient. Each simulation run, I incremented the hit rate percent by .02 and the service time by 1 ms. Then I ran MovingAverage.py with time window 3200 seconds which is the batch mean I derived before. Then I plotted the last response time with respect to hitPercent to make the plot below:

For the open simulation, the hit rate of .70 and service time of 10ms worked best.

In [4]:
```
hitPercent = seq(.70, .94, by=.02)
times = c(0.17682893455494023,0.1999829636746834,
          0.19984689870743177, 0.2086989613189093,
          0.24214763206310033, 0.27465817553031496,
          0.30749232179942126, 0.3339057912440235,
          0.4884504116985327, 1.5972812484112746,
          1986.6670861810949, 4373.165901122005,
          6364.305258940425)

plot(hitPercent, times)
```



In [ ]:
```

```