Estimable Proof-of-Work (EPoW)

- **Adviser**: 薛智文教授
- **Presenter**: 羅文斌

- **Date**: 2017-06-14

Background

- Users send transactions to each other by broadcasting the transactions as Internet packets to miners for confirmation.
- Miners have to find a hash value of the mining block, called **nonce**, to be less than a number, **target**.
- Once the nonce is found, the new block is mined and waits for confirmation.
- The miners get rewards (a fixed amount of bitcoins + transaction fees) once the block is confirmed.

- **Proof of Work (PoW)** is to prove that some substantial computing work has been done so that the winner can append a new block.
- **Estimable PoW** qualitatively estimate how much work is done in closed formula at once instead of just in average statistically in a long run.
- The hash range in Bitcoin is $\mathbf{2^{256} \cong 1.158 \times 10^{77}}$
- EPoW asks to provide two nonce values, **low nonce** and **high nonce**.
- **Low nonce** and **high nonce** are the lowest and the highest hash value ever generated, respectively.
- How many times the nonces have been tried can be estimated from the **trial ranges**.

Lemma

Integers $\mathbf{1}$ to N are fairly generated with the same possibility $p = 1/N$, at the m-th generation, the possibility, i.e. $P(i, j|m)$, of the lowest number ever generated being i and the highest being j, where $1 \leq i \leq j \leq N$, $m \geq 1$, and the range size $n = j - i + 1 \geq 1$, is

$$P(i, j|m) = \begin{cases} p^m \text{ if } i = j \\ (n^m - 2(n-1)^m + (n-2)^m)p^m \text{ otherwise} \end{cases}$$

$$P(n|m) = \begin{cases} p^{m-1} \text{ if } n = 1 \\ (N - n + 1)(n^m - 2(n-1)^m + (n-2)^m)p^m \text{ otherwise} \end{cases}$$

Problem Formulation

Integers 1 to N are fairly generated with the same possibility $p = 1/N$. Given a trial range size n, what is the statistical property of m?

Let's first consider $P(m|n) = P(n|m)P(m)/P(n)$

If we assume the prior, i.e. $P(m) = c$, follows uniform distribution, then

- $P(n, m)$ can be approximated by $cP(n|m)$
- $P(m|n)$ can be approximated by $cP(n|m)/P(n)$

Mean and Variance

$$P(n) = \sum_{m=1}^{\infty} P(n,m) = \begin{cases} \frac{c}{1-p} \text{ if } n = 1 \\ c(N-n+1)[\frac{np}{1-np} - \frac{2(n-1)p}{1-(n-1)p} + \frac{(n-2)p}{1-(n-2)p}] \text{ otherwise} \end{cases}$$

$$P(m|n) = \begin{cases} p^{m-1}(1-p) \text{ if } n = 1 \\ (n^m - 2(n-1)^m + (n-2)^m)p^m/(\frac{np}{1-np} - \frac{2(n-1)p}{1-(n-1)p} + \frac{(n-2)p}{1-(n-2)p}) \text{ otherwise} \end{cases}$$

$$\mathbf{E}[m|n] = \begin{cases} \frac{1}{1-p} \text{ if } n = 1 \\ (\frac{np}{(1-np)^2} - \frac{2(n-1)p}{(1-(n-1)p)^2} + \frac{(n-2)p}{(1-(n-2)p)^2})/(\frac{np}{1-np} - \frac{2(n-1)p}{1-(n-1)p} + \frac{(n-2)p}{1-(n-2)p}) \text{ otherwise} \end{cases}$$

$$\mathbf{Var}[m|n] = \begin{cases} \frac{p}{(1-p)^2} \text{ if } n = 1 \\ \text{trivial otherwise} \end{cases}$$

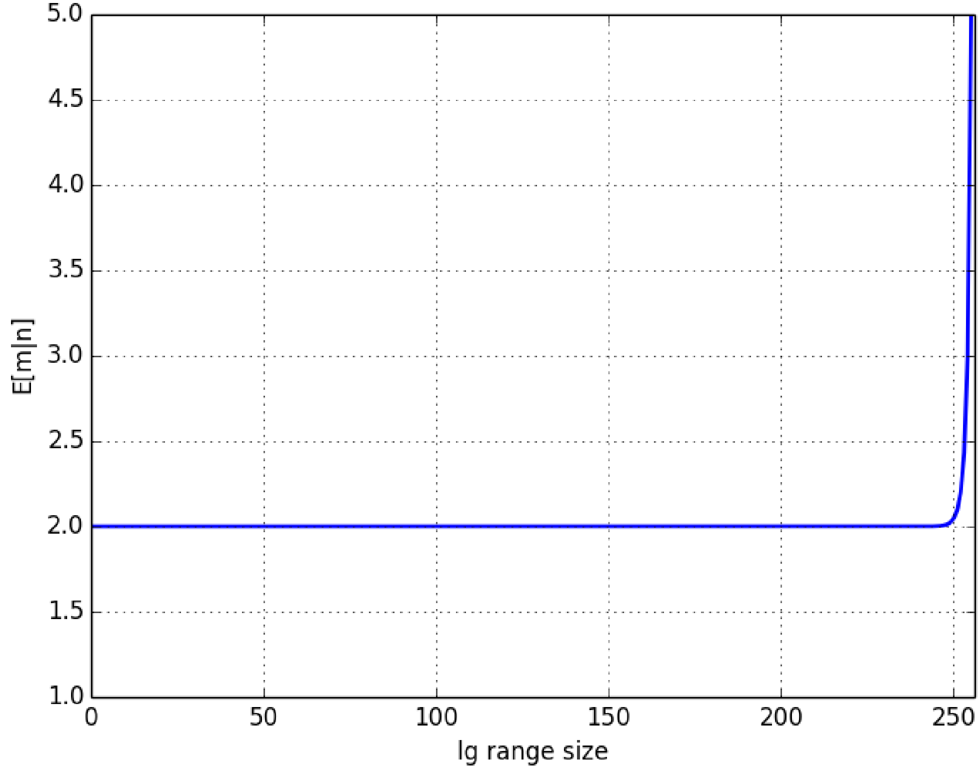
Script

```
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
def numerator(n):
    return N * n * (N-(n-1))**2 * (N-(n-2))**2 - \
           N * 2 * (n-1) * (N-n)**2 * (N-(n-2))**2 + \
           N * (n-2) * (N-n)**2 * (N-(n-1))**2
def denominator(n):
    return n * (N-n) * (N-(n-1))**2 * (N-(n-2))**2 - \
           2 * (n-1) * (N-n)**2 * (N-(n-1)) * (N-(n-2))**2 + \
           (n-2) * (N-n)**2 * (N-(n-1))**2 * (N-(n-2))
def mu(n):
    return numerator(n)/denominator(n)
```

Script

```
x = 256
N = 2 ** x
out = []
for i in range(x):
    out.append(mu(2 ** i))
line = plt.plot(range(x), out)
plt.setp(line, linewidth = 2)
plt.xlabel("lg range size")
plt.ylabel("E[m|n]")
plt.xlim(0, x)
plt.ylim(1, 5)
plt.show()
```

Plotting of $\mathrm{E}[m|n], N = 2^{256}$



Observations

- From the figures, it is obvious to see that $E[m|n]$ converges to 5 when n goes to $N/2$. But how about when n goes from $N/2$ to N?
- Intuitively, we would expect $E[m|n]$ to behave like an exponential function.
- Before moving on, we define n to be a function of x

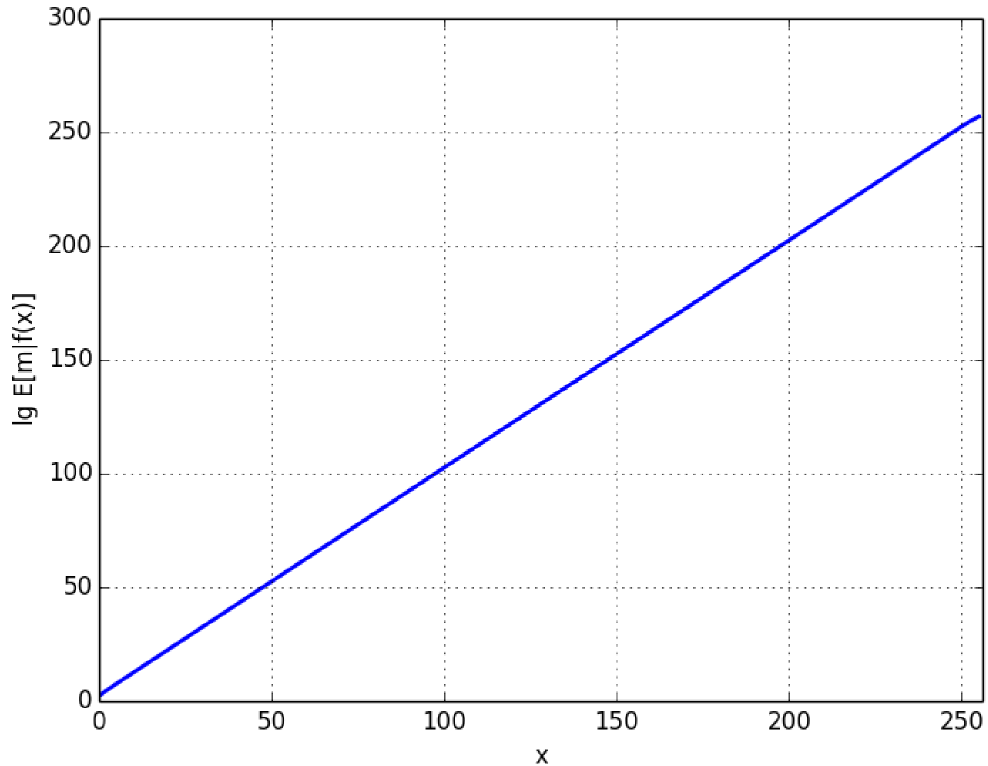
$$n = f(x) = \sum_{i=\lg N - x}^{\lg N - 1} 2^i = N - N/2^x$$

  where $x \in \{1, 2, \ldots, \lg N - 1, \lg N\}$.
- We noticed that $\lg E[m|f(x)]$ is a linear function of $x = f^{-1}(n) = \lg N - \lg(N - n)$.
- Remember in the case of Bitcoin, $N = 2^{256}$

Script

```
x = 256
N = 2 ** x
n = 0
out = []
for i in reversed(range(x)):
    n += 2 ** i
    out.append(mu(n))
line = plt.plot(range(x), np.log2(out))
plt.setp(line, linewidth = 2)
plt.xlabel("x")
plt.ylabel("lg E[m|f(x)]")
plt.xlim(0, x)
plt.show()
```

Plotting of $\lg E[m|f(x)]$



Wrapping Up

- Based on the results in the above, a model is proposed:

$$E[m|n] \cong 2^{f^{-1}(n)} = \frac{N}{N-n} = \frac{1}{1-n/N}$$

where

$$n = f(x) = \sum_{i=\lg N-x}^{\lg N-1} 2^i = N - N/2^x$$

- To sum up, $E[m|n] \cong \frac{1}{1-n/N}$ regardless of n. And from the formula, it is only trivial to show that the posterior $P(m|n)$ would have a geometric distribution.
- The BlockChain or Bitcoin system can give rewards to miners according to mean and variance directly derived from the geometric distribution.

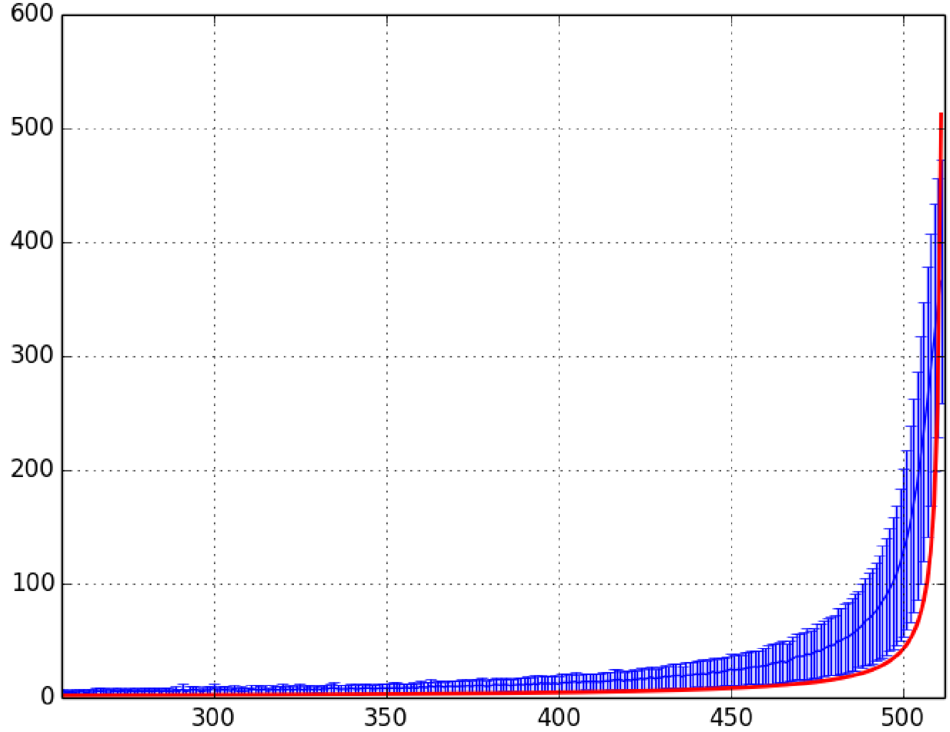
Simulation

```
from __future__ import division
import numpy as np
import random
N = 1024
log = []
for i in range(N):
    log.append([])
for i in range(N*4):
    for m in range(1, N+1):
        log[np.ptp(np.random.choice(N, m))].append(m)
def mu(n):
    return N/(N-n)
```

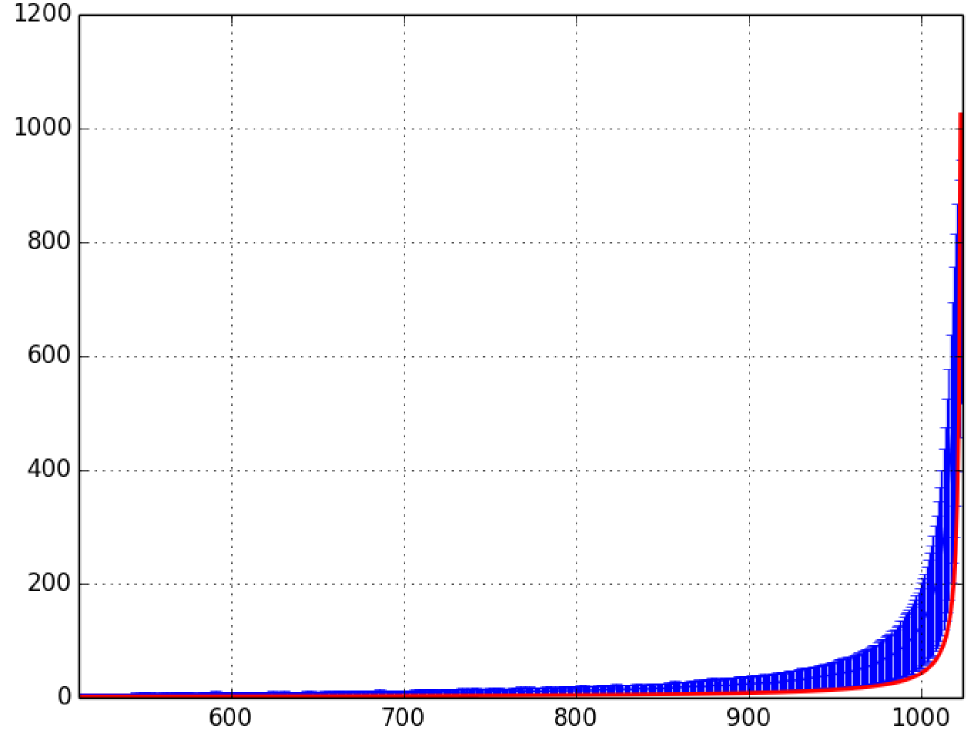
Simulation

```
mean = [np.mean(log[i]) if len(log[i]) > 0 else 2 for i in range(N)]
std = [np.std(log[i]) if len(log[i]) > 0 else 2 for i in range(N)]
plt.errorbar(range(N), mean, std)
plt.plot(range(N), np.vectorize(mu)(range(N)), 'r', lw = 2)
plt.xlim(N/2, N)
plt.grid()
plt.show()
```

Simulation of $\mathrm{E}[m|n], N = 512$



Simulation of $\mathrm{E}[m|n], N = 1024$



Simulation of $\mathrm{E}[m|n], N = 1024$