1 a) $0.05 \cdot 0.05 = 0.0025$ (probability that both servers are down)

$1 - 0.0025 = 0.9975$

b) $T(0{:}30) =$ S1 fail, S2 fail

$T(1{:}00) =$ S1 online, S online

Pretty much, if both servers failed for anytime in between the minute pings (and 1 managed to reboot), his script would still report good success. Enough occurrences would lead to faulty data.

c) Measure the servers' availability at random times.

d) The probability of a server going down doesn't affect the other server (independence of 2 events).

e) If the two servers were connected to the same power supply, a power surge would put both servers offline.

2 a) $f(x) = \frac{1}{80\sqrt{2\pi}} e^{-0.5\frac{(x-200)^2}{6400}}$

b) $\mu = 200, \ \sigma = 80$

c) $z(190) = \frac{190-200}{80} = -0.125$

$z(185) = \frac{185-200}{80} = -0.1875$

$(1 - P(-0.125)) - (1 - P(-0.1875)) = 0.0236$

3a) $f(x, 0.1) = \frac{0.1^x}{x!} e^{-0.1}$

b) $f(1,1) = \frac{1}{1!} e^{-1} = 0.368$

$f(0,1) = \frac{1^0}{0!} e^{-1} = 0.368$

$1 - P(x=1) - P(x=0) = 0.264$

c) $f(x, 0.1) = 0.1e^{-0.1x}$ (1 ms)

d) $f(x \leq 20) = 1 - e^{-0.1 \cdot 20} = 0.865$ (probability they will arrive 20ms apart)

$\Rightarrow \boxed{0.135}$

e) std dev $= \sqrt{var} = \sqrt{\frac{1}{\lambda^2}} = \frac{1}{0.1} = 10$ ms

f) $P(x < 1100)$

$\mu = 1000$

$\sigma = 10$

$Z(1100) = \frac{1100-1000}{10} = 10$

$\Rightarrow \boxed{100\%}$

(4) g) $1.0 = \lambda\, 0.008$

$\lambda = 125$ pkts/sec

h) $f(x, 125) = \dfrac{125^x e^{-125}}{x!}$

i) $w = \dfrac{\rho^2}{1-\rho} = \dfrac{0.8^2}{0.2} = 3.2$ packets

$\rho = \dfrac{100}{125} = 0.8$

j) $T_w = \dfrac{\rho}{\mu(1-\rho)} = \dfrac{0.8}{125 \cdot 0.2} = 0.032\,s$

k) slowdown $= \dfrac{\text{turnaround time}}{\text{service time}} = \dfrac{0.04}{0.008} = 5$

$T_s = \dfrac{1}{\mu} = 0.008\,s$

l) The system can handle 125 packets/sec and it is currently processing 100 packets/sec on average. If 26 packets/sec of adversarial traffic is injected, the server will crash.

```java
import java.util.*;

/**
 * @author Matthew Huynh
 * @description This program has 2 methods which can generate
 * random values from the standard normal distribution and
 * from a normal distribution with a parameterized mean and
 * standard deviation.
 */
public class RandGen {

 public static void main(String[] args) {
   System.out.println("random value from standard normal dist: " + Zrand(100));
   System.out.println("random value from norm dist, mean=0, stdev=1: " + Grand(0, 2));
 }

 /**
  * returns a random value that is distributed according to
  * a standard normal distribution
  */
 static double Zrand(int N)
 {
   Random uniRand = new Random();
   double[] samples = new double[N];
   for (int i = 0; i < samples.length; i++)
   {
     samples[i] = 4*uniRand.nextDouble()-2; // -2 to 2
   }

   double mean = mean(samples);
     double sd = stdev(samples, mean);
     System.out.println("mean: " + mean + ", stdev: " + sd);

     // return a random sample from this sample
   return samples[uniRand.nextInt(N)];
 }

 /**
  * returns a random value that is distributed according to
  * a normal distribution with mean U and standard deviation S
  */
 static double Grand(double U, double S)
 {
   Random uniRand = new Random();
   double shift = 0.5 - U; // if shift is negative, add it to random value, else subtract
   double variance = S*S;
   double expansion = variance;
   //System.out.println("shift: " + shift + ", variance: " + variance + ", expansion: " + expansion);

   double[] samples = new double[100];
   for (int i = 0; i < samples.length; i++)
   {
     double rv = uniRand.nextDouble();
     if (shift < 0)
     {
       rv += shift;
     }
     else
     {
       rv -= shift;
     }
     rv *= expansion;
     samples[i] = rv;
   }

   double mean = mean(samples);
     double sd = stdev(samples, mean);
     System.out.println("mean: " + mean + ", stdev: " + sd);

   // return a random sample from this sample
   return samples[uniRand.nextInt(samples.length)];
 }

static double mean(double[] samples)
{
  double sum = 0;
  for (double s : samples)
  {
    sum += s;
  }
  return sum/samples.length;
}

static double stdev(double[] samples, double mean)
{
  double variance = 0;
  for (double s : samples)
  {
    variance += Math.pow(s, 2);
  }
  variance /= samples.length;
  return Math.sqrt(variance);
}

}
```