대기행렬 이론 개론

VIII. Computer Simulation

학습에 앞서

- 학습 목표
  - 컴퓨터 시뮬레이션을 통한 통신네트워크 성능 분석 방법을 학습한다.
  - 대기행렬이론을 이용한 수학적인 성능 분석 방법과 컴퓨터 시뮬레이션을 이용한 성능 분석 방법을 비교한다.

- 목차
  - 1. Random Number Generation
  - 2. The Inversion Transform Method
  - 3. Traffic Generation
  - 4. Computer Simulation

1. Random Number Generation (1/2)

A random number generator is a computational or physical device designed to generate a sequence of numbers or symbols that lack any pattern, i.e. appear random.

Most C implementations have, lurking within, a pair of library routines for initializing, and then generating, random numbers. In ANSI C, the synopsis is:

source 7.1.0 rand()

```
#include <stdlib.h>
#define RAND_MAX 32767

void srand(unsigned seed);
int rand(void);
```

Random Number Generation (2/2)

- The random number has the uniformity and the independence.

You initialize the random number generator by invoking `srand(seed)` with some arbitrary seed. You obtain successive random numbers in the sequence by successive calls to `rand()`. That function returns an integer that is typically in the range 0 to the largest representable positive value of type int (inclusive). Usually, as in ANSI C, this largest value is available as `RAND_MAX`, but sometimes you have to figure it out for yourself. If you want a random float value between 0.0 (inclusive) and 1.0 (exclusive), you get it by an expression like

```
x = (float)rand()/(RAND_MAX+1.0);
```

Example

```
#define A    16807L                /* multiplier (7**5) for 'ranf' */
#define M    2147483647L           /* modulus (2**31-1) for 'ranf' */

static long SeedTable[16]= {0L, /* seeds for streams 1 thru 15  */
  1973272912L,  747177549L,    20464843L,  640830765L, 1098742207L,
    78126602L,   84743774L,   831312807L,  124667236L, 1172177002L,
  1124933064L, 1223960546L,  1878892440L, 1449793615L,  553303732L};

static int strm=1;                /* index of current stream */

double ranf(void)
{
    short *p,*q,k; long Hi,Lo;
    p=(short *)&SeedTable[strm]; Hi=*(p+1)*A;    /* 16807*H->Hi */
    *(p+1)=0; Lo=SeedTable[strm]*A;              /* 16807*L->Lo */
    p=(short *)&Lo; Hi+=*(p+1); /* add high-order bits of Lo to Hi */
    q=(short *)&Hi;                          /* low-order bits of Hi->Lo */
    *(p+1)=*q&0x7fff;                        /* clear sign bit */
    k=*(q+1)<<1; if (*q&0x8000) k++;         /* Hi bits 31-45->K */
    /* form Z + K [- M] (where Z=Lo) : presubtract M to avoid overflow */
    Lo-=M; Lo+=k; if (Lo<0)  Lo+=M;
    SeedTable[strm]=Lo;
    return((double)Lo*4.656612875E-10); /* Lo x 1/(2**31-1) */
}
```

Random Integer Generator

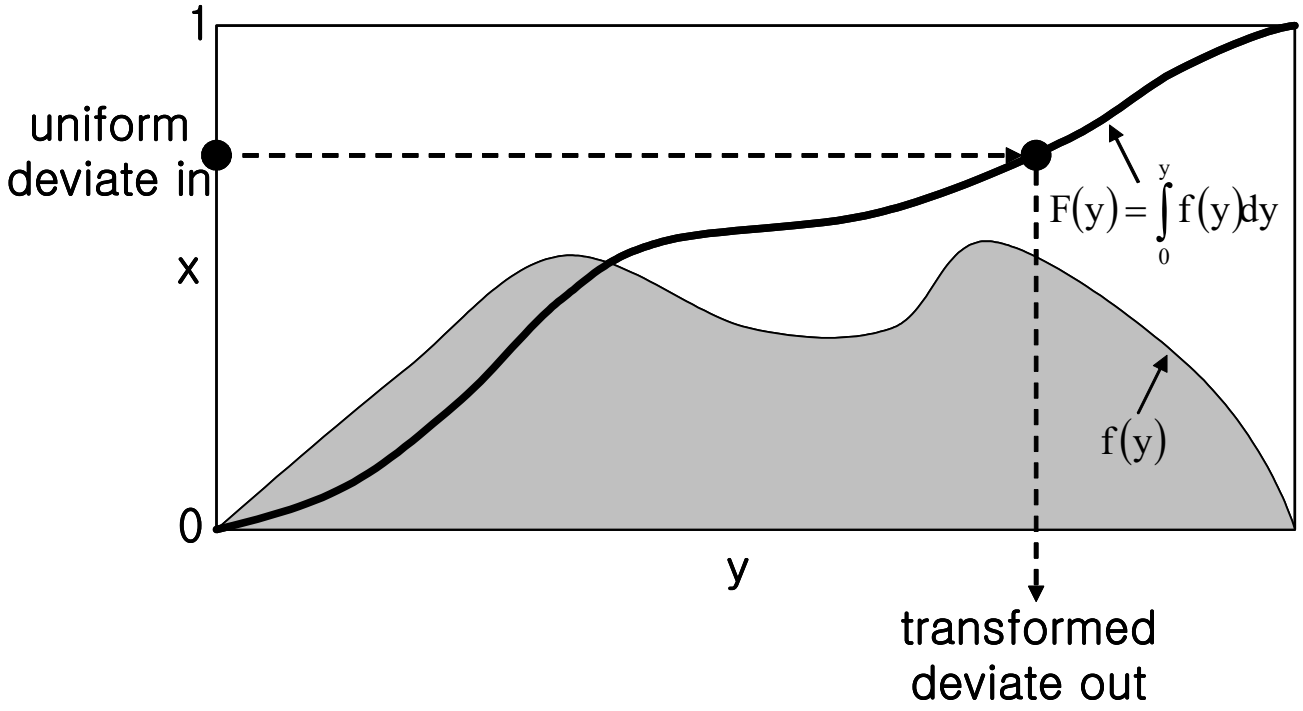
- Random Integer Generator
  - 'rani' returns an integer equiprobably selected from the set of integers i, i+1, i+2, …, n.

```
int rani(int i, int n)
{
    if (i==n)   return i;
    if (i>n) { printf("random Argument Error: i > n\n"); exit(1); }
    n-=i; n=(int)( (n+1.0)*ranf() );
    return(i+n);
}
```

2. The Inversion Transform Method

How to generate sample values from the exponential distribution? This section gives you answers.

We consider the cumulative distribution function $y = F(x)$, where y is uniformly distributed in $[0, 1]$. The sample value x can then be generated from the inverse function $x = F^{-1}(y)$. In some cases, it is possible to determine directly the inverse of the distribution function of a theoretical distribution.



Example – Exponential Distribution

- Exponential distribution

For example, suppose interarrival times are exponentially distributed with mean λ. The cumulative distribution is given by

$$F(x) = 1 - e^{-\lambda x}$$

$$x = -\frac{1}{\lambda}\ln(1-y)$$

where y is a random variate uniformly distributed in $[0, 1]$, $(1-y)$ is identically distributed, therefore, we can rewrite as

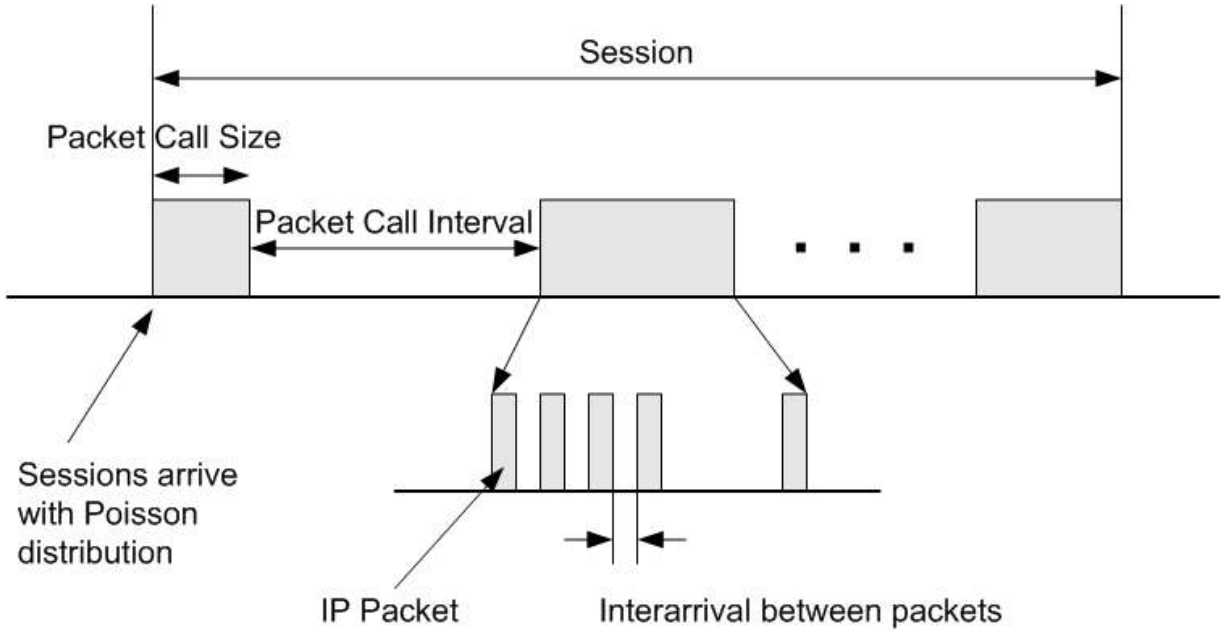
$$x = -\frac{1}{\lambda}\ln y$$

- C codes

```
double expntl(double mean)
{
    return(-mean*log(ranf()));
}
```

Example – Pareto Distribution (1/2)

- Pareto distribution
  - A pareto distribution has a heavy-tail. It is useful to model internet traffic using a pareto distribution.



  - One session in web traffic model is composed of packet calls and one call consists of packets. The packet calls per session are geometrically distributed and inter-arrival time between packet calls is exponentially distributed. The number of packets per packet call is Pareto distributed with parameter $\beta = 1.1$ and $k = 2.27$ (mean number = 25).

Example – Pareto Distribution (2/2)

- Pareto distribution

$$F(x) = \begin{cases} 1 - \left(\frac{k}{x}\right)^{\alpha} & \text{if } x \geq k > 0 \\ 0 & \text{if } x < k \end{cases}$$

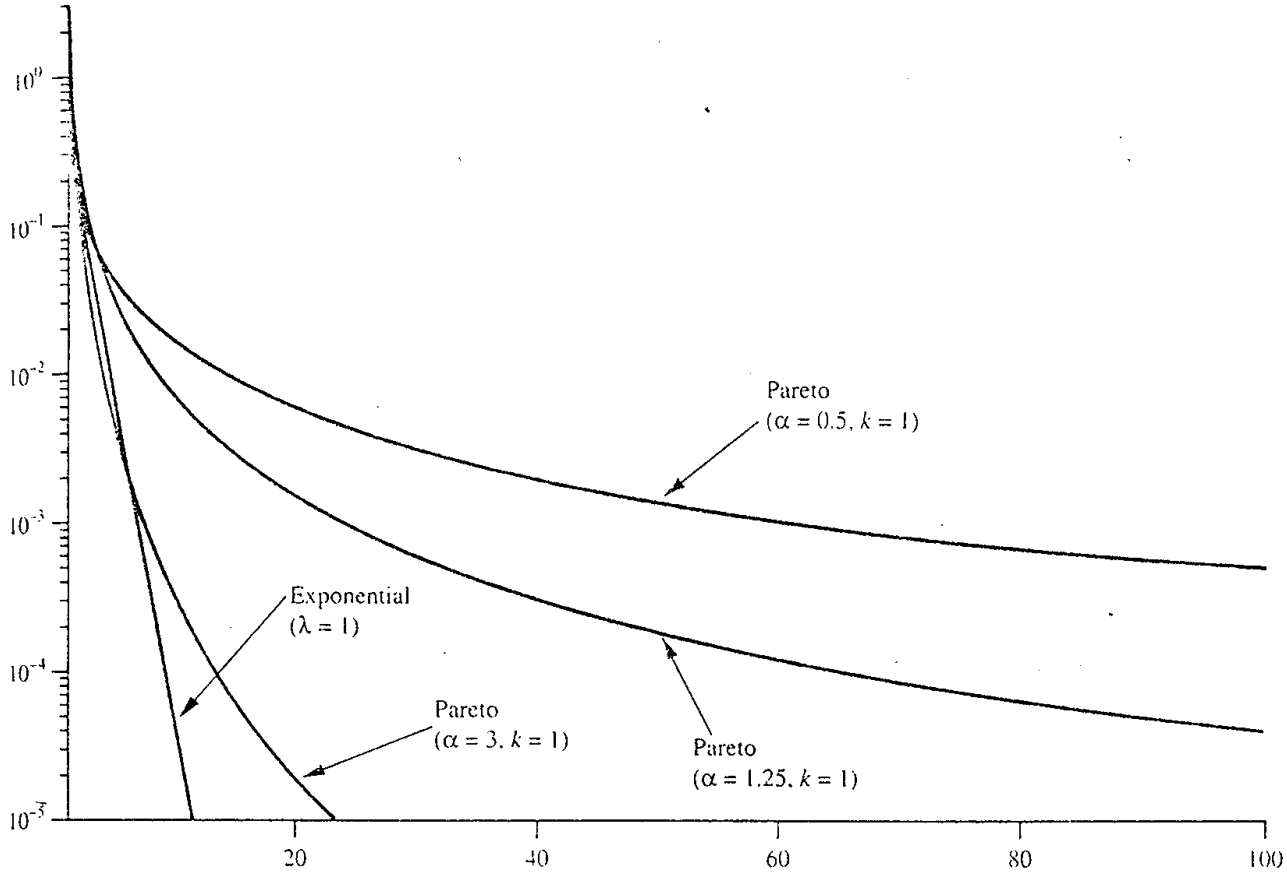
- C codes

```
double pareto(double alpha, double k)
{
    double  u;

    u = ranf();
    if (u==0.0) return  k;
    else        return  k / pow(u, 1.0/alpha);
}
```

Exponential Distribution vs. Pareto Distribution

- The comparison of an exponential distribution and pareto distributions
  - The heavy-tail property of pareto distribution means that probability of pareto distribution is greater than that of exponential distribution as x increases. It is considered that the heavy-tail property is suitable for modeling an Internet traffic which has data transmissions **for a long time.)**



3. Traffic Generation

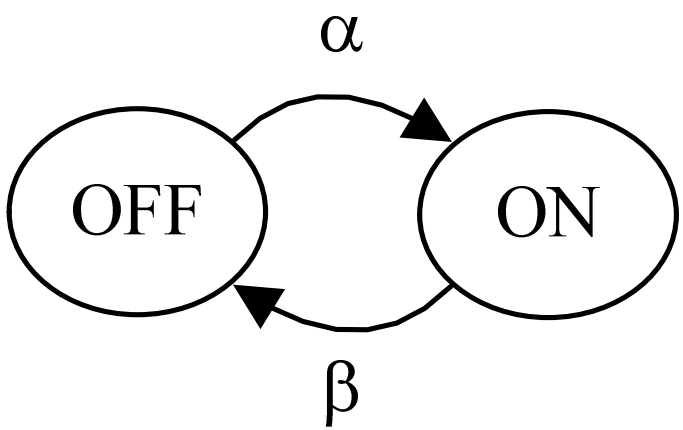
- Data traffic
  - FTP traffic can be modeled as a Poisson process.
  - The interarrival time between packets is exponentially distributed.

```
#define END_SIMTIME 36000  /* simulation time */
int poisson_traffic(int queueNum, double lambda, double avgPktLength)
{
    int numArrival=0;
    double arrivalTime, departureTime;

    for (arrivalTime=0.0; arrivalTime<END_SIMTIME; arrivalTime+=expntl(1./lambda))
    {
        departureTime = arrivalTime + expntl(avgPktLength);
        insert_queue(arrivalTime, departureTime);
                        /* insert the packet into a traffic_queue */
        numArrival++;
    }
    return numArrival;
}
```

Voice Traffic Model

- Voice traffic
  - In VoIP services, the ON–OFF source model has been used to characterize traffic from a single voice source. A voice packet arrives at fixed intervals of *T* ms during talk spurts or in the ON state, and no packet arrives during silences or in the OFF state.



Voice Traffic Generation (1/2)

- There are *K* voice users. Each user generates ON-OFF voice traffic with the avgTalkTime and the avgSilentTime.

```
#define END_SIMTIME 36000  /* simulation time */
int voice_traffic(int K, double avgTalkTime, double avgSilentTime)
{
    int     userIndex;
    int     numArrival=0;
    double  arrivalTime, departureTime;
    int     state, nextState;
    double  onPeriod, offPeriod;

    for (userIndex=0; userIndex<K; userIndex++)
    {
        state = rani(0, 1);
        ...

    }
    return numArrival;
}
```

Voice Traffic Generation (2/2)

```
for (arrivalTime=0.0; arrivalTime<END_SIMTIME; \
        arrivalTime=departureTime, state=nextState) {
    // ON state
    if (state==1)
    {
        onPeriod        = expntl(avgTalkTime);
        departureTime   = arrivalTime + onPeriod;
        nextState       = 0;
        insert_queue(arrivalTime, departureTime, userIndex);
                /* insert the packet into a traffic_queue */
    }
    // OFF state
    else
    {
        offPeriod       = expntl(avgSilentTime);
        departureTime   = arrivalTime + offPeriod;
        nextState       = 1;
    }
}
```

4. Computer Simulation

- We consider an uplink for a wireless communication system with *N* users and a base station. The *N* users generate data packets at a Poission rate λ and the service times of the packets are exponentially distributed with a mean of 1 second. Plot the average delay as the value of ρ increases.

- There are two kinds of simulation methods: time-based simulation and event-based simulation.

Definitions

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// Simulation environment
#define SIM_TIME   1.0e6       // Simulation time
#define SERV_TIME  1.00        // Mean service time

double ranf();
double expntl(double mean);
double mm1_ana(double lambda);         // Analysis
double mm1_sim(double lambda);         // Event-based simulation
double mm1_sim2(double lambda);        // Time-based simulation
```

Main Functions

```
int main()
{
   double       lambda;
   double       mu = 1.0 / SERV_TIME;
   FILE         *fp;

   fp = fopen("f_result.txt", "w");
   if (fp==NULL)
   {
        printf("File open error\n");
        exit(1);
   }

   for (lambda = 0.01; lambda <= mu; lambda += mu/50.0)
   {
        fprintf(fp, "%lf\t%lf\t%lf\t%lf\n", \
                lambda, mm1_ana(lambda), \
                mm1_sim(lambda), mm1_sim2(lambda));
        fflush(fp);
   }

   fclose(fp);
   return 0;
}
```

Analysis

- Average sojourn time

$$W = E(T) = \frac{L}{\lambda} = \frac{1}{\mu - \lambda} = \frac{1}{\mu(1-\rho)}$$

```
// M/M/1 Analysis
double mm1_ana(double lambda)
{
   double mu = 1.0 / SERV_TIME;

   return (1.0 / (mu - lambda));
}
```

Event-based Simulation (1/2)

```
// M/M/1 Event-based simulation in terms of arrivals and departures

double mm1_sim(double lambda)
{
   double   Ta = 1.0/lambda;  // Mean time between arrivals
   double   Ts = SERV_TIME;   // Mean service time

   double   time = 0.0;       // Simulation time
   double   t1 = 0.0;         // Time for next event #1 (arrival)
   double   t2 = SIM_TIME;    // Time for next event #2 (departure)
   unsigned int n = 0;        // Number of customers in the system

   unsigned int c = 0;        // Number of service completions
   double   b = 0.0;          // Total busy time
   double   s = 0.0;          // Area of number of customers in system
   double   tn = time;        // Variable for "last event time"
   double   tb;               // Variable for "last start of busy time"

   double   throughput;       // throughput = c / time
   double   utilization;      // utilization = b / time
   double   mean_number;      // mean_number = s / time;
   double   delay;            // delay = mean_number / throughput;
```

Event-based Simulation (2/2)

```
// Main simulation loop
while (time < SIM_TIME)
{
    if (t1 < t2)                          // *** Event #1 (arrival) ***
    {
        time = t1;
        s = s + n * (time - tn);          // Update area under "s" curve
        n++;
        tn = time;                        // tn = "last event time" for next event
        t1 = time + expntl(Ta);
        if (n == 1)
        {
            tb = time;                    // Set "last start of busy time"
            t2 = time + expntl(Ts);
        }
    }
    else                                  // *** Event #2 (departure) ***
    {
        time = t2;
        s = s + n * (time - tn);          // Update area under "s" curve
        n--;
        tn = time;                        // tn = "last event time" for next event
        c++;                              // Increment number of completions
        if (n > 0)
            t2 = time + expntl(Ts);
        else {
            t2 = SIM_TIME;
            b = b + time - tb;            // Update busy time sum if empty
        }
    }
}
```

Time-based Simulation (1/3)

```
// M/M/1 Time-based simulation
#define MAX_PKTS       1000000

typedef struct {
   double arrivalTime;
   double departureTime;
} pktType;

double mm1_sim2(double lambda)
{
   pktType     *pktList;
   int         i = 0;
   int         numArrivals = 0;
   double      sumDelay = 0.0;
   double      waitingTime;
   double      arrivalTime;

   pktList = (pktType *)malloc(sizeof(pktType)*MAX_PKTS);
   if (!pktList)
   {
        printf("allocation error\n");
        return -1;
   }
   ...
```

Time-based Simulation (2/3)

```
// traffic generation
for (arrivalTime = 0.0; arrivalTime <= SIM_TIME; \
                              arrivalTime += expntl(1.0/lambda))
{
     if (i >= MAX_PKTS)
     {
          printf("Out of MAX_PKTS\n");
          return -1;
     }
     else
     {
          pktList[i].arrivalTime = arrivalTime;
          pktList[i].departureTime =
                              arrivalTime + expntl(SERV_TIME);
          i++;
          numArrivals++;
     }
}
```

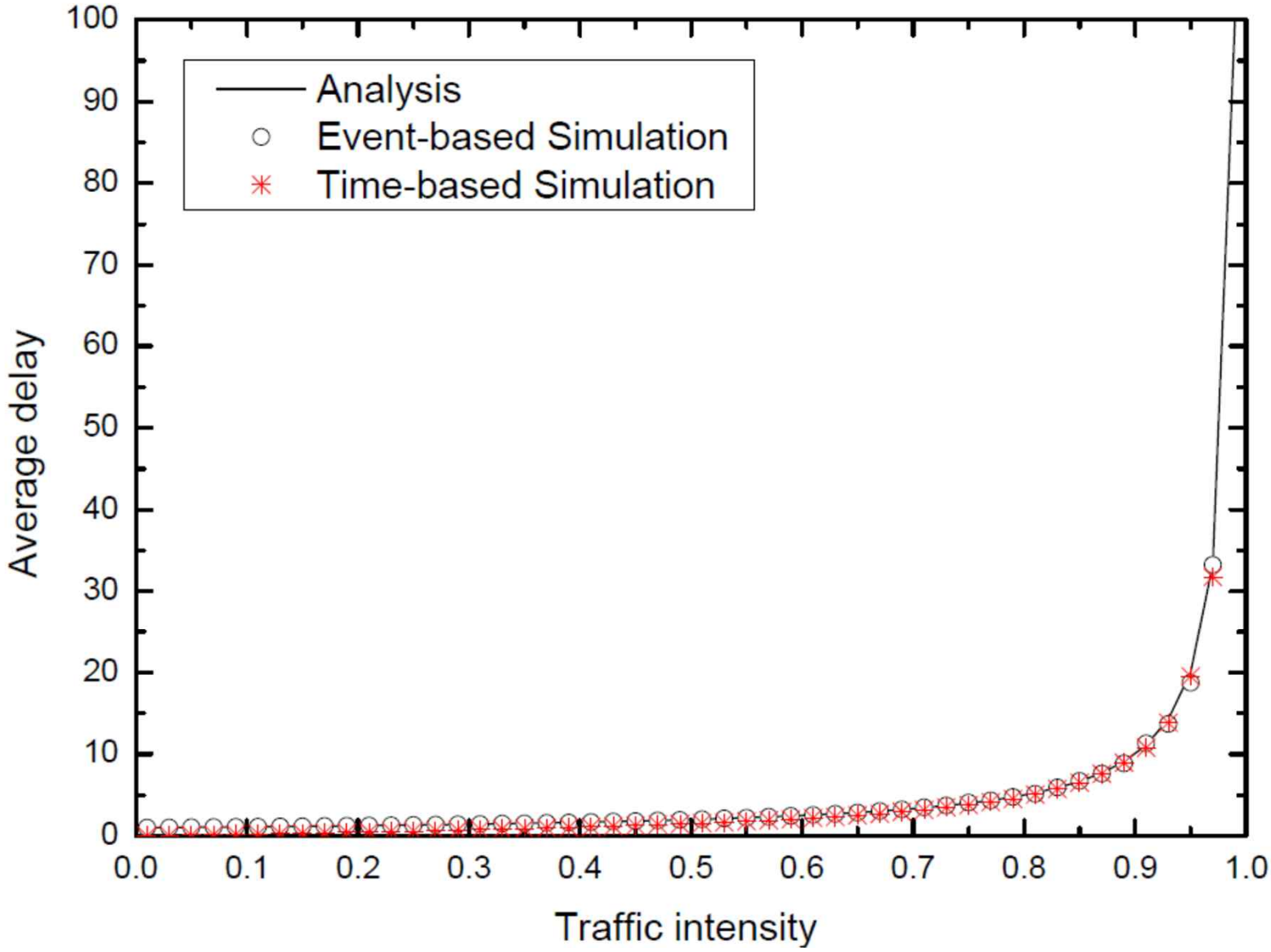
Time-based Simulation (3/3)

```
    // scheduling
    for (i=1; i<numArrivals; i++)
    {
         if (pktList[i].arrivalTime < pktList[i-1].departureTime)
         {
              waitingTime = \
                  pktList[i-1].departureTime - pktList[i].arrivalTime;
              pktList[i].departureTime += waitingTime;
              sumDelay +=
                  pktList[i].departureTime - pktList[i].arrivalTime;
         }
    }

    free(pktList);

    return sumDelay / (double)numArrivals;
}
```

Average Delay



Summary

- Random Number Generation
  - Generate a sequence of numbers that lack any pattern, i.e., appear random
  - The random number has the uniformity and the independence
- The Inversion Transform Method
  - Generate sample values from the exponential distribution
  - Examples: exponential distribution, pareto distribution
- Traffic Generation
  - Data traffic: Poisson process model
  - Voice traffic: ON-OFF source model
- Computer Simulation
  - Time-based simulation and event-based simulation