Project	**IEEE 802.16 Broadband Wireless Access Working Group <http://ieee802.org/16>**
Title	**C-Program for 2IRP Streaming Video Model**
Date Submitted	**2000-03-05**
Source(s)	Tingjun Wen, Nalin Molligoda and Jun Huang, Dept. of S.C.E. Carleton University Ottawa ON K1S 5B6 Canada — Voice: 613 271 6889 Fax: 613 599 4225 mailto: nmolligo@opentext.com
Re:	This is an informational contribution for promoting IEEE802.16.3 traffic ad-hoc models, specifically 2IRP part.
Abstract	In this contribution, an example software program written in C language is offered for the 2IRP model proposed in the contribution IEEE802.16.3c-01/30r1 and IEEE802.16.3c-00/58, the flow chart and comments are included to ease the reading of the program. It may help MAC/PHY proponents to shorten their development cycle for the traffic simulations. This is just a preliminary reference for understanding the basic structure of the model; no one is obligated to employ it.
Purpose	Post on the web server for public access.
Notice	
Release	
Patent Policy and Procedures	

C-Program for 2IRP Streaming Video Model

T. Wen, N. Molligoda and J. Huang
Carleton University

I. Introduction

In this contribution, an example C program is offered for simulating the traffic generated from a 2IRP model proposed in the contribution [3] IEEE802.16.3c-01/30r1 and [4] IEEE802.16.3c-00/58. Details on the models are not repeated here, we assume you have read both contributions.

To simulate the exponentially distributed packet inter arrival time, a subroutine is written to generate random numbers that follows the exponential distribution [1]. The initial random number generation is done using the `rand()` function in C language [2]. A similar subroutine handles the generation of Pareto distributed sojourn times.

We have verified the simulation result with theoretical calculation of the Hurst parameters against real video trace. The average packets per unit-of-time is also very close to theoretical value after normalization for compensating Pareto s thick tail effect.

This program has been developed as a part of the graduate course 94.581Y at Systems and Computer Engineering Department of the Carleton University, Ottawa, Canada. The Word format of this file is downloadable from the site [5].

II. Overall Flowchart

Here is the flow diagram for the whole program.

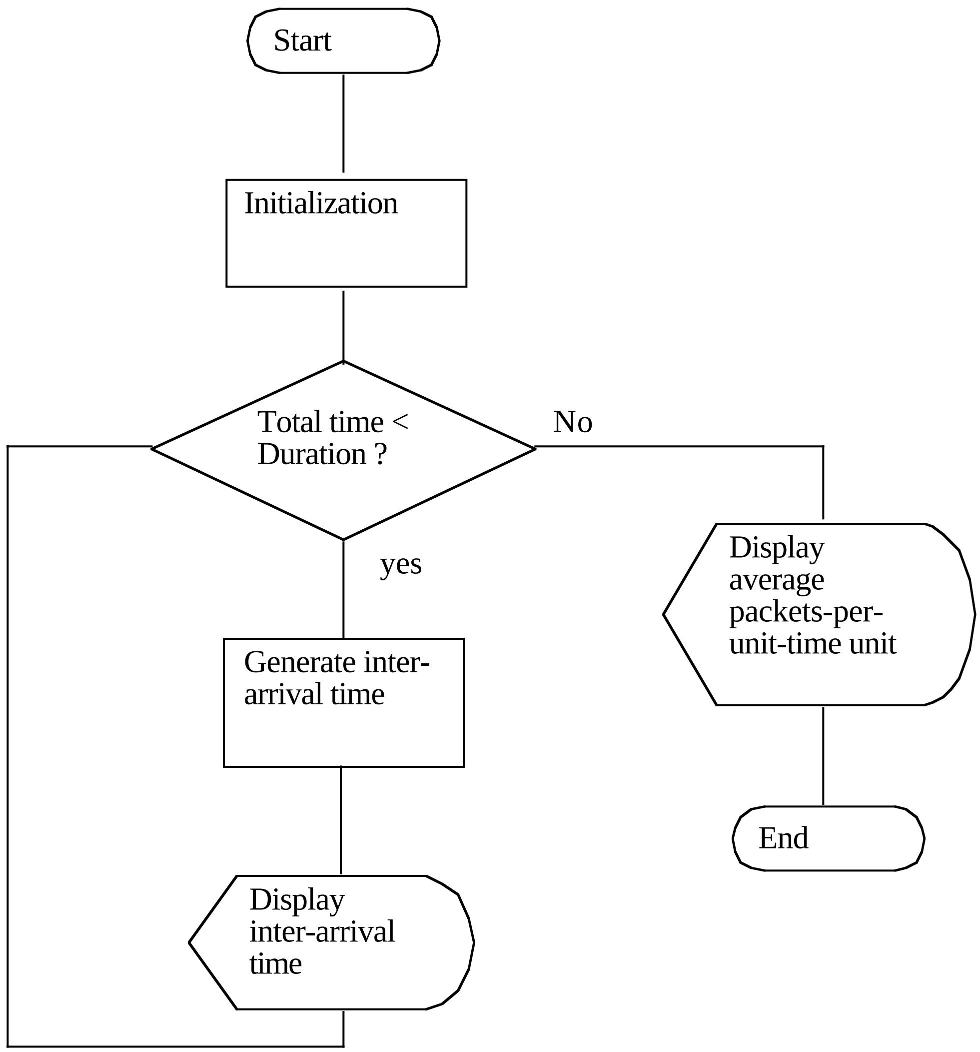

Figure 1. Flow Diagram

III. Program Break Down

The program composites of following sections:
Section 1: Notice.
Section 2: Defines the functions used in the program.
Section 3: Main section of the program. This contains
the initialization,
main loop and
displaying of
final results.
Main loop iterates until the simulation time is equal to the duration required. This duration is defined in the initialization part.
Section 4: This section generates 2IRP packets using the IRP function defined in the section 5.
Section 5: This function generates IRP packets.
Section 6: Exponentially distributed random number generator.
Section 7: Pareto distributed random number generator.
Section 8: Draws a line graph to depict the number of packets generated.

IV. Usage Guide

This program can be compiled using any C/C++ compilers and can be run on most of the operating systems available.

To compile in Unix:

- gcc 2IRP.c -o 2IRP.exe -lm

To compile in Windows:

- Create a new project using the compilers (e.g. Visual C++) interface.
- Add the 2IRP.c file to the newly created project.
- Compile 2IRP.c and create a new exe (2IRP.exe) using the interface.

To execute:

- Run the 2IRP.exe.
- After the message `"We will display the number of packets being   "` press enter to continue.
- If plot() in Main is NOT commented out before compilation, a line graph will be flashed in the screen depicting the number of packet per unit-of-time.
- Average packets-per-unit-time time will be displayed at the end of the execution.

- The program writes all inter-arrival times to a file named “`iat.txt` .
- All packets-per-unit-time is written to a file named "`ppu.txt`"

With some trivial modifications, the core part of this program can also be embedded in other simulators:

- Use section 5, 6 and 7 if child processing is available.
- Use section 4, 5, 6 and 7 if child process structure is not used.
- Section 6 and 7 may not need if your simulator already has them.
- Normalization in Section 3 can be embedded in section 4 or 5.
- You don t need Normalization if your implementation is perfect.
- By simply scale down input lambdas you can also skip the Normalization section.
- If you already have 4IPP model, all you need is section 7.
- By calling section 4 twice and skip section 7, it can generate 4IPP for Internet.
- By modified section 5 and skip section 7, it generates IDP for voice.

V. Further Study

Since Pareto distribution has large variance, we observe that in our implementation, the observed average packet arriving rate is above the theoretical value. The way, we solve this implementation issue is to introduce a simple brute force normalization, rather than making the program more delicate and complicate.

We achieved results very close to the theoretical average of 50.52 packets per unit time after we applied Normalization for the Pareto thick tail compensation. Due to time limitation, the program is not fully tested. More studies on these Pareto characteristics will be reported later, for improving the simulation precision.

All parameters used in the program (mean arrival rates, ON time Pareto parameters, OFF time Pareto parameters and the duration to run the simulation) are hard coded in this version of code, but can be easily changed to user inputs manner in a future version.

Pareto generator has only one parameter (shape parameter), the scaling parameter is assumed as 1, the next release will address that, if there is a need.

VI. Reference

[1] Devroye, L, “Non-Uniform Random Variate Generation”, Springer-Verlag New York Inc. 1986

[2] Kernighan, B, “The C programming language”, Englewood Cliffs, N.J., Prentice Hall. 1988

[3] C.R. Baugh, J. Huang, "Traffic Model for MAC/PHY Simulations, IEEE802.16.3c-01/30r1, March, 2001.

[4] J. Huang, "Extending 4IPP Model for IEEE802.16.3", IEEE802.16.3c-00/58, Dec. 2000.

[5] http://www.sce.carleton.ca/courses/94581/index.html

VII. Appendix: The Program

```
/*
SECTION 1. NOTICE. $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
*/

/* Copyright 2001, Tingjun Wen, Nalin Molligoda and Jun Huang,
Carleton University.

This software is a result of a project assignment of a graduate
course 94.581Y taught by Professor Jun Huang et al at Department
of Computer and System Engineering of Carleton University in
Capital region of Ottawa of the country of Canada.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

   1.Redistributions of source code must retain the above copyright
     notice, this list of conditions and the following disclaimer.
   2.Redistributions in binary form must reproduce the above
     copyright notice, this list of conditions and the following
     disclaimer in the documentation and/or other materials provided
     with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS ''AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

/*
SECTION 2, DECORATIONS. ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <time.h>

double irp2(double * lambda, double * alpha1, double * alpha2);
double irp(double lambda, double alpha1, double alpha2);
double exp_rand(double lamda);       /* Exponential random variable generator.*/
double pareto_rand(double alpha);    /*Pareto random variable generator.*/
void plot(long packets_per_unit_of_time, int max);

/*
SECTION 3, MAIN. !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
*/

void main(int argc, char * argv)
{
      double   duration = 170990.0;       /* Time period for generating "2-hour
                                             simulated Starwar movie" packets,
                                             in unit-of-time (or MPEG frame) */

      /**************************************************************************/
      /* Default parameters for 2IRP model
            */
      /*                  {IRP1,   IRP2}
            */
      /**************************************************************************/
      double   lambda[2] = {44.95, 61.90};      /* Mean arrival rates
      */
      double   alpha1[2] = { 1.14,  1.54};      /* ON time Pareto parameters        */
      double   alpha2[2] = { 1.22,  1.28};      /* OFF time Pareto parameters       */
      /**************************************************************************/

      /* iat: inter arrival time
       * ppu: packets per unit-of-time */
      FILE     * fp_iat, * fp_ppu;
      char     * file_name_iat = "iat.txt";
      char     * file_name_ppu = "ppu.txt";
      double   current_time = 0.0;
      double   inter_arrival_time = 0.0;
      double   next_frame_time = 1.0;
      long     packets_per_unit_of_time = 0;
      long     total_packets = 0;
      long     tmp_count;

      /* duration = 20.0; /* test this program using small number of frames*/

      /* Random number seed derived from current simulating time*/
      srand( (unsigned)time( NULL ) );

      /* open a file to write the inter-arrival-time */
      fp_iat = fopen(file_name_iat, "w");
      if (fp_iat == NULL)
      {
            fprintf(stderr, "Error in creating file %s\n", file_name_iat);
            exit(1);
      }
      /* open a file to write the packets per unit-of-time */
      fp_ppu = fopen(file_name_ppu, "w");
      if (fp_ppu == NULL)
      {
            fprintf(stderr, "Error in creating file %s\n", file_name_ppu);
            fclose(fp_iat);
            exit(1);
      }

      /* pop up messages to screen for "busy guys like you!" */

      printf("We will display the number of packets being generated if you want, \n");
      printf("press Enter key to continue, it may take a few minute...\n");
```

```
getchar();
printf("Writing inter-arrival-time data to iat file: %s\n", file_name_iat);
printf("Writing packets-per-unit-of-time data to file: %s\n\n", file_name_ppu);


/* keep on our iterations */
while (current_time <= duration && next_frame_time <= duration) {
      inter_arrival_time = irp2(lambda, alpha1, alpha2);

      /*************************************************************
      Due to Pareto's "thick tail effect", we merge the 29th
      and the 30th packets. This thins the point process and closes the gap
      between  continuum  theory and  discrete  realities, it is proposed by
      Professor Huang. After this compensation, the packets-per-unit-of-time
      in average is very close to the theoretical calculation.
      *************************************************************/
      tmp_count++;

      /* merge 29th and 30th packets to compensate Pareto's thick tail effect */
      if ((tmp_count % 29) == 0) {
            inter_arrival_time += irp2(lambda, alpha1, alpha2);
            tmp_count++;
      }

      current_time += inter_arrival_time;
      packets_per_unit_of_time++;
      total_packets++;

      /* output inter arrival times into a file */
      fprintf(fp_iat, "%lf\n", inter_arrival_time);

      /* output packets_per_unit_of_time */
      while (current_time >= next_frame_time) {
            fprintf(fp_ppu, "%ld\n", packets_per_unit_of_time);

            /* :-) :-) :-) :-) :-) :-) :-) :-) @ (-: (-: (-: (-: (-: (-: (-: (-: */

            /* To see the packets-per-unit-of-time graph,
             * uncomment the following line */
            /* plot(packets_per_unit_of_time, (int)(lambda[0] + lambda[1])); */

            packets_per_unit_of_time = 0;
            next_frame_time += 1.0;
      }
}

fclose(fp_ppu);
fclose(fp_iat);

/* pop up more messages to screen */

printf("Writing inter-arrival-time data to iat file: %s done\n", file_name_iat);
printf("Writing packets-per-unit-of-time data to file: %s done\n\n", file_name_ppu);
printf("Press Enter key to continue.\n");
getchar();
printf("\n\nAverage packets-per-unit-of-time is: %lf\n",
      (double)total_packets / (double)next_frame_time);
```

```
      printf("Press any key to exit.\n");
      getchar();
      return;
}

/*
SECTION 4, SUBROUTINE "MARRY". ++++++++++++++++++++++++++++++++++++++
*/

/* Superposition of 2 irp sources.
*
* Input : lambda[2], alpha1[2], alpha2[2]
* Output: inter_arrival_time
*/
double irp2(double * lambda, double * alpha1, double * alpha2)
{
      static double current_time[2] = {0.0, 0.0};
      static double last_packet_time = 0.0;
      double inter_arrival_time;

/**************************************************************
                      +---------> current_time[0]
                      |
  |__________|________|_______|_____|_____________|___

  |_______|________|_______|_____|__________|_______|_
                   |       |
                   |       +---> current_time[1]
                   +-----------> last_packet_time
***************************************************************/

      /* if this is the first time this function is executed,
       * initialize current_time[2] = { 0.0, 0.0}.
       * Otherwise, collision happens, for simplicity, no packet is generated,
       * the introduced error if any will be compensated in section 3 */
      while (fabs(current_time[0] - current_time[1]) <= 0.0) {
            current_time[0] += irp(lambda[0], alpha1[0], alpha2[0]);
            current_time[1] += irp(lambda[1], alpha1[1], alpha2[1]);
      }

      /* (1) Select the slow source.
       * (2) Inter arrival time is the difference between
       *     last_packet_time and the slow source.
       * (3) Generate a new packet for the slow source */
      if (current_time[0] < current_time[1]) {
            inter_arrival_time = current_time[0] - last_packet_time;
            last_packet_time = current_time[0];
            current_time[0] += irp(lambda[0], alpha1[0], alpha2[0]);
      } else if (current_time[0] > current_time[1]) {
            inter_arrival_time = current_time[1] - last_packet_time;
            last_packet_time = current_time[1];
            current_time[1] += irp(lambda[1], alpha1[1], alpha2[1]);
      }

      return inter_arrival_time;
}
```

```
/*
SECTION 5, SUBROUTINE  SINGLE . ||||||||||||||||||||||||||||||||||
*/

/* Single IRP source.
*
* Input : lambda, alpha1, alpha2
* Output: inter_arrival_time
*/
double irp(double lambda, double alpha1, double alpha2)
{
      /* sojourn time left is initialized as 0.0, which means
       * initially it's on the start edge of the OFF state */
      static double sojourn_time_left = 0.0;
      double inter_arrival_time;
      double exp_rv;

/*************************************************************
        ON              OFF             ON          OFF

  __________________         _________________
  |__|__|____|_____|_________|____|____|_____|______|
     |  |          |         |               |
     \_/|\________/ \_______/ \_____________/
      | |    |         |            |
      | |    |         |            +--> pareto_rand(alpha1)
      | |    |         +---------------> pareto_rand(alpha2)
      | |    +-------------------------> sojourn_time_left
      | +------------------------------> current_packet
      +--------------------------------> exp_rand(lambda)
**************************************************************/

      inter_arrival_time =  exp_rand(lambda);
      /* if it's in ON state, inter arrival time obeys
       * exponential distribution */
      if (sojourn_time_left - inter_arrival_time >= 0.0) {
            /* deduct the sojourn time by Poisson interval */
            sojourn_time_left -= inter_arrival_time;
      }
      /* if it's in OFF state, inter arrival time is
       * roughly equal to OFF time. To simplify the process
       * We assume Expectation(Poisson(lambda)
       * << {Expectation(Pareto(alpha1), Expectation(Pareto(alpha2)}
       * which is 97% true in practice. The compensation for the
       * rest 3% is made at Section 3 */
      else {
            inter_arrival_time += pareto_rand(alpha2);
            /* no packet on rising edge of OFF state*/
            exp_rv = exp_rand(lambda);
            inter_arrival_time += exp_rv;
            sojourn_time_left = pareto_rand(alpha1) - exp_rv;
      }

      return inter_arrival_time;
}

/*
```

```
SECTION 6, SUBROUTINE EXP. eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
*/

/* Exponential distribution random number generator.
* f(x) = lambda * exp(-lambda * x)
* inversion: -log(z) / lambda
*/
double exp_rand(double lambda)
{
      /* uniformly distributed random variable */
      int U;

      /* get uniformly distributed random variable
       * in (0.0, RAND_MAX) */
      do {
            U = rand();
      } while ((U == 0) || (U == RAND_MAX));

      /* convert it to exponentially distributed random
       * variable using inversion method and return it. */
      return (-(log((double)U) - log((double)RAND_MAX)) / lambda);
}


/*
SECTION 7, SUBROUTINE PAR. pppppppppppppppppppppppppppppppp
*/

/* Pareto distribution random number generator.
* f(x) = a * b / x ** (a + 1)
* inversion: b / z ** (1 / a)
*
* Note, in this example program, unit of time (b) is assumed as 1.
* Because of this, only one parameter is used in the Pareto random number generation.
*/
double pareto_rand(double alpha)
{
      /* uniformly distributed random variable */
      int U;

      /* get uniformly distributed random variable
       * in (0.0, RAND_MAX) */
      do {
            U = rand();
      } while ((U == 0) || (U == RAND_MAX));

      /* convert it to Pareto distributed random
       * variable using inversion method and return it. */
      return(1.0 / pow((double)U / (double)RAND_MAX, (1.0 / alpha)));
}


/*
SECTION 8, SUBROUTINE "SHOW-TIME". :-) :-) :-) :-) :-) :-) :-) :-)
*/

void plot(long packets_per_unit_of_time, int max)
```

```
{
	int i;
	/* assume your screen is 80x25 */
#define SCREEN_WIDTH 79
	int delta = (int)((float)max / (float)SCREEN_WIDTH + 1.0);

	putchar('.');
	for (i= 0; i < packets_per_unit_of_time; i += delta) {
		putchar('_');
		if (i >= delta * SCREEN_WIDTH) {
			putchar('*');
			break;
		}
	}
	putchar(0X0A);
}
```