**C code:**

**#2:**

// arrival rate 0 < lamda < 1000, 100, 200, 300, 400, 500, 600, 700, 800, 900, 950

#define PACKET\_ARRIVAL\_RATE 600 /\* packets per second \*/

#define PACKET\_LENGTH 1e3 /\* bits \*/

#define LINK\_BIT\_RATE 1e6 /\* bits per second \*/

#define RUNLENGTH 10e6 /\* packets \*/

/\* Comma separated list of random seeds to run. \*/

#define RANDOM\_SEED\_LIST 123456,234567,345678,456789,5678901,6789123,7891234,8912345,1061556,1069639

#define PACKET\_XMT\_TIME ((double) PACKET\_LENGTH/LINK\_BIT\_RATE)

#define BLIPRATE (RUNLENGTH/1000)

**#3:**

**<-- main.c -->**

int counter;

double delay;

**<-- main.h -->**

extern int counter;

extern double delay;

**<-- cleanup\_memory.c -->**

void

cleanup\_memory (Simulation\_Run\_Ptr simulation\_run)

{

Simulation\_Run\_Data\_Ptr data;

Fifoqueue\_Ptr buffer;

Server\_Ptr link;

data = (Simulation\_Run\_Data\_Ptr) simulation\_run\_data(simulation\_run);

buffer = data->buffer;

link = data->link;

if(link->state == BUSY) /\* Clean out the server. \*/

xfree(server\_get(link));

xfree(link);

while (fifoqueue\_size(buffer) > 0) /\* Clean out the queue. \*/

xfree(fifoqueue\_get(buffer));

xfree(buffer);

xfree(data); /\* Clean up the simulation\_run data. \*/

simulation\_run\_free\_memory(simulation\_run); /\* Clean up the simulation\_run. \*/

}

**<-- packet\_transmission.c -->**

/\* Collect statistics. \*/

data->number\_of\_packets\_processed++;

delay = simulation\_run\_get\_time(simulation\_run) - this\_packet->arrive\_time;

data->accumulated\_delay += delay;

if (delay > 0.005) counter++;

**<-- output.c -->**

printf("Probability = %.2f%% \n", (double)counter/data->number\_of\_packets\_processed\*100.0);

// re-initialize counter value

counter = 0;

**#4:**

**<-- main.c -->**

data->blip\_counter = 0;

data->arrival\_count = 0;

data->number\_of\_packets\_processed = 0;

data->accumulated\_delay = 0.0;

data->random\_seed = random\_seed;

data->number\_in\_link = 0;

data->number\_in\_link2 = 0;

/\*

\* Create the packet buffer and transmission link, declared in main.h.

\*/

data->buffer = fifoqueue\_new();

data->link = server\_new();

data->link2 = server\_new();

**<-- main.h -->**

typedef struct \_simulation\_run\_data\_

{

Fifoqueue\_Ptr buffer;

Server\_Ptr link;

Server\_Ptr link2;

**<-- clearnup\_memory.c -->**

void

cleanup\_memory (Simulation\_Run\_Ptr simulation\_run)

{

Simulation\_Run\_Data\_Ptr data;

Fifoqueue\_Ptr buffer;

Server\_Ptr link;

Server\_Ptr link2;

data = (Simulation\_Run\_Data\_Ptr) simulation\_run\_data(simulation\_run);

buffer = data->buffer;

link = data->link;

link2 = data->link2;

if(link->state == BUSY) /\* Clean out the server. \*/

xfree(server\_get(link));

xfree(link);

if(link2->state == BUSY) /\* Clean out the server. \*/

xfree(server\_get(link2));

xfree(link2);

**<-- packet\_arrival.c -->**

/\*

\* Start transmission if the data link is free. Otherwise put the packet into

\* the buffer.

\*/

if(server\_state(data->link) == BUSY && server\_state(data->link2) == BUSY) {

fifoqueue\_put(data->buffer, (void\*) new\_packet);

} else {

start\_packet\_transmission(simulation\_run, new\_packet);

}

**<-- packet\_transmission.c -->**

/\*

\* Packet transmission is finished. Take the packet off the data link.

\*/

this\_packet = (Packet\_Ptr) ptr;

if(this\_packet->packet\_link == 1){

this\_packet = (Packet\_Ptr) server\_get(data->link);

}else{

this\_packet = (Packet\_Ptr) server\_get(data->link2);

}

**...**

void

start\_packet\_transmission(Simulation\_Run\_Ptr simulation\_run,

Packet\_Ptr this\_packet)

{

Simulation\_Run\_Data\_Ptr data;

data = (Simulation\_Run\_Data\_Ptr) simulation\_run\_data(simulation\_run);

TRACE(printf("Start Of Packet.\n");)

if(server\_state(data->link) == BUSY)

{

server\_put(data->link2, (void\*) this\_packet);

this\_packet->packet\_link = 2;

}else{

server\_put(data->link, (void\*) this\_packet);

this\_packet->packet\_link = 1;

}

this\_packet->status = XMTTING;

**#5:**

**<-- main.c -->**

/\*

\* Create a Simulation\_Run\_Data object. This will hold all of our user

\* defined data (declared in main.h). Set the simulation\_run data pointer

\* to our new object.

\*/

data = (Simulation\_Run\_Data\_Ptr) xmalloc(sizeof(Simulation\_Run\_Data));

simulation\_run\_set\_data(simulation\_run, (void\*) data);

/\*

\* Initialize the simulation\_run data variables, declared in main.h.

\*/

data->blip\_counter = 0;

data->arrival\_count = 0;

data->number\_of\_packets\_processed = 0;

data->accumulated\_delay = 0.0;

data->random\_seed = random\_seed;

data->number\_of\_data\_packets\_processed = 0;

data->number\_of\_voice\_packets\_processed = 0;

data->number\_of\_voice\_2\_packets\_processed = 0;

data->data\_packet\_accumulated\_delay = 0.0;

data->voice\_packet\_accumulated\_delay = 0.0;

data->voice\_2\_packet\_accumulated\_delay = 0.0;

/\*

\* Create the packet buffer and transmission link, declared in main.h.

\*/

data->buffer = fifoqueue\_new();

data->link = server\_new();

/\*

\* Set the random number generator seed for this run.

\*/

random\_generator\_initialize(random\_seed);

/\*

\* Schedule the initial packet arrival for the current clock time (= 0).

\*/

schedule\_packet\_arrival\_event(simulation\_run,

simulation\_run\_get\_time(simulation\_run));

schedule\_voice\_packet\_arrival\_event(simulation\_run,

simulation\_run\_get\_time(simulation\_run));

schedule\_voice\_2\_packet\_arrival\_event(simulation\_run,

simulation\_run\_get\_time(simulation\_run));

/\*

\* Execute events until we are finished.

**<-- main.h -->**

typedef struct \_simulation\_run\_data\_

{

Fifoqueue\_Ptr buffer;

Server\_Ptr link;

long int blip\_counter;

long int arrival\_count;

long int number\_of\_packets\_processed;

double accumulated\_delay;

unsigned random\_seed;

long int number\_of\_data\_packets\_processed;

long int number\_of\_voice\_packets\_processed;

long int number\_of\_voice\_2\_packets\_processed;

double data\_packet\_accumulated\_delay;

double voice\_packet\_accumulated\_delay;

double voice\_2\_packet\_accumulated\_delay;

} Simulation\_Run\_Data, \* Simulation\_Run\_Data\_Ptr;

typedef enum {XMTTING, WAITING} Packet\_Status;

typedef enum {DATA, VOICE, VOICE\_2} Packet\_Type;

typedef struct \_packet\_

{

double arrive\_time;

double service\_time;

int source\_id;

int destination\_id;

Packet\_Type packet\_type;

Packet\_Status status;

} Packet, \* Packet\_Ptr;

**<-- output.c -->**

output\_results(Simulation\_Run\_Ptr simulation\_run)

{

double xmtted\_fraction;

Simulation\_Run\_Data\_Ptr data;

data = (Simulation\_Run\_Data\_Ptr) simulation\_run\_data(simulation\_run);

printf("\n");

printf("Random Seed = %d \n", data->random\_seed);

printf("Packet arrival count = %ld \n", data->arrival\_count);

xmtted\_fraction = (double) data->number\_of\_packets\_processed /

data->arrival\_count;

printf("Transmitted packet count = %ld (Service Fraction = %.5f)\n",

data->number\_of\_packets\_processed, xmtted\_fraction);

printf("Arrival rate = %.3f packets/second \n", (double) PACKET\_ARRIVAL\_RATE);

printf("Overal Mean Delay (msec) = %.2f \n",

1e3\*data->accumulated\_delay/data->number\_of\_packets\_processed);

// output for #5

printf("Transmitted data packet count = %ld \n",

data->number\_of\_data\_packets\_processed);

printf("Transmitted voice 1 packet count = %ld \n",

data->number\_of\_voice\_packets\_processed);

printf("Transmitted voice 2 packet count = %ld \n",

data->number\_of\_voice\_2\_packets\_processed);

printf("Data Packet Mean Delay (msec) = %.2f \n",

1e3\*data->data\_packet\_accumulated\_delay/data->number\_of\_data\_packets\_processed);

printf("Voice Stream 1 Packet Mean Delay (msec) = %.2f \n",

1e3\*data->voice\_packet\_accumulated\_delay/data->number\_of\_voice\_packets\_processed);

printf("Voice Stream 2 Packet Mean Delay (msec) = %.2f \n",

1e3\*data->voice\_2\_packet\_accumulated\_delay/data->number\_of\_voice\_2\_packets\_processed);

printf("\n");

}

**<-- packet\_arrival.c -->**

/\*

\* This function will schedule a packet arrival at a time given by

\* event\_time. At that time the function "packet\_arrival" (located in

\* packet\_arrival.c) is executed. An object can be attached to the event and

\* can be recovered in packet\_arrival.c.

\*/

long int

schedule\_packet\_arrival\_event(Simulation\_Run\_Ptr simulation\_run,

double event\_time)

{

Event\_Type this\_event = {"Packet Arrival", packet\_arrival\_event};

long int event\_id;

event\_id =

simulation\_run\_schedule\_event(simulation\_run, this\_event, event\_time,

(void\*) NULL);

return event\_id;

}

long int

schedule\_voice\_packet\_arrival\_event(Simulation\_Run\_Ptr simulation\_run,

double event\_time)

{

Event\_Type this\_event = {"Voice Stream 1 Packet Arrival", voice\_packet\_arrival\_event};

long int event\_id;

event\_id =

simulation\_run\_schedule\_event(simulation\_run, this\_event, event\_time,

(void\*) NULL);

return event\_id;

}

long int

schedule\_voice\_2\_packet\_arrival\_event(Simulation\_Run\_Ptr simulation\_run,

double event\_time)

{

Event\_Type this\_event = {"Voice Stream 2 Packet Arrival", voice\_2\_packet\_arrival\_event};

long int event\_id;

event\_id =

simulation\_run\_schedule\_event(simulation\_run, this\_event, event\_time,

(void\*) NULL);

return event\_id;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* This is the event function which is executed when a packet arrival event

\* occurs. It creates a new packet object and places it in either the fifo

\* queue if the server is busy. Otherwise it starts the transmission of the

\* packet. It then schedules the next packet arrival event.

\*/

void

packet\_arrival\_event(Simulation\_Run\_Ptr simulation\_run, void\* ptr)

{

Simulation\_Run\_Data\_Ptr data;

Packet\_Ptr new\_packet;

data = (Simulation\_Run\_Data\_Ptr) simulation\_run\_data(simulation\_run);

data->arrival\_count++;

new\_packet = (Packet\_Ptr) xmalloc(sizeof(Packet));

new\_packet->arrive\_time = simulation\_run\_get\_time(simulation\_run);

new\_packet->service\_time = exponential\_generator((double) MEAN\_SERVICE\_TIME);

new\_packet->status = WAITING;

new\_packet->packet\_type = DATA;

/\*

\* Start transmission if the data link is free. Otherwise put the packet into

\* the buffer.

\*/

if(server\_state(data->link) == BUSY) {

fifoqueue\_put(data->buffer, (void\*) new\_packet);

} else {

start\_packet\_transmission(simulation\_run, new\_packet);

}

/\*

\* Schedule the next packet arrival. Independent, exponentially distributed

\* interarrival times gives us Poisson process arrivals.

\*/

schedule\_packet\_arrival\_event(simulation\_run,

simulation\_run\_get\_time(simulation\_run) +

exponential\_generator((double) 1/PACKET\_ARRIVAL\_RATE));

}

void

voice\_packet\_arrival\_event(Simulation\_Run\_Ptr simulation\_run, void\* ptr)

{

Simulation\_Run\_Data\_Ptr data;

Packet\_Ptr new\_packet;

data = (Simulation\_Run\_Data\_Ptr) simulation\_run\_data(simulation\_run);

data->arrival\_count++;

new\_packet = (Packet\_Ptr) xmalloc(sizeof(Packet));

new\_packet->arrive\_time = simulation\_run\_get\_time(simulation\_run);

new\_packet->service\_time = get\_voice\_packet\_transmission\_time();

new\_packet->status = WAITING;

new\_packet->packet\_type = VOICE;

/\*

\* Start transmission if the data link is free. Otherwise put the packet into

\* the buffer.

\*/

if(server\_state(data->link) == BUSY) {

fifoqueue\_put(data->buffer, (void\*) new\_packet);

} else {

start\_packet\_transmission(simulation\_run, new\_packet);

}

/\*

\* Schedule the next packet arrival. Independent, exponentially distributed

\* interarrival times gives us Poisson process arrivals.

\*/

schedule\_voice\_packet\_arrival\_event(simulation\_run,

simulation\_run\_get\_time(simulation\_run) + VOICE\_ARRIVAL\_TIME);

}

void

voice\_2\_packet\_arrival\_event(Simulation\_Run\_Ptr simulation\_run, void\* ptr)

{

Simulation\_Run\_Data\_Ptr data;

Packet\_Ptr new\_packet;

data = (Simulation\_Run\_Data\_Ptr) simulation\_run\_data(simulation\_run);

data->arrival\_count++;

new\_packet = (Packet\_Ptr) xmalloc(sizeof(Packet));

new\_packet->arrive\_time = simulation\_run\_get\_time(simulation\_run);

new\_packet->service\_time = get\_voice\_2\_packet\_transmission\_time();

new\_packet->status = WAITING;

new\_packet->packet\_type = VOICE\_2;

/\*

\* Start transmission if the data link is free. Otherwise put the packet into

\* the buffer.

\*/

if(server\_state(data->link) == BUSY) {

fifoqueue\_put(data->buffer, (void\*) new\_packet);

} else {

start\_packet\_transmission(simulation\_run, new\_packet);

}

/\*

\* Schedule the next packet arrival. Independent, exponentially distributed

\* interarrival times gives us Poisson process arrivals.

\*/

schedule\_voice\_2\_packet\_arrival\_event(simulation\_run,

simulation\_run\_get\_time(simulation\_run) + VOICE\_2\_ARRIVAL\_TIME);

}

**<-- packet\_arrival.h -->**

void

packet\_arrival\_event(Simulation\_Run\_Ptr, void\*);

void

voice\_packet\_arrival\_event(Simulation\_Run\_Ptr simulation\_run, void\* ptr);

void

voice\_2\_packet\_arrival\_event(Simulation\_Run\_Ptr simulation\_run, void\* ptr);

long

schedule\_packet\_arrival\_event(Simulation\_Run\_Ptr, double);

long

schedule\_voice\_packet\_arrival\_event(Simulation\_Run\_Ptr, double);

long

schedule\_voice\_2\_packet\_arrival\_event(Simulation\_Run\_Ptr, double);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**<-- packet\_transmission.c -->**

void

end\_packet\_transmission\_event(Simulation\_Run\_Ptr simulation\_run, void\* ptr)

{

Simulation\_Run\_Data\_Ptr data;

Packet\_Ptr this\_packet, next\_packet;

TRACE(printf("End Of Packet.\n"););

data = (Simulation\_Run\_Data\_Ptr) simulation\_run\_data(simulation\_run);

/\*

\* Packet transmission is finished. Take the packet off the data link.

\*/

this\_packet = (Packet\_Ptr) server\_get(data->link);

/\* Collect statistics. \*/

data->number\_of\_packets\_processed++;// total packets transmitted

// different packet type has different delay and accumulated delay

if(this\_packet->packet\_type == DATA){

data->number\_of\_data\_packets\_processed++;

data->data\_packet\_accumulated\_delay += simulation\_run\_get\_time(simulation\_run) - this\_packet->arrive\_time;

}

else if (this\_packet->packet\_type == VOICE){

data->number\_of\_voice\_packets\_processed++;

data->voice\_packet\_accumulated\_delay += simulation\_run\_get\_time(simulation\_run) - this\_packet->arrive\_time;

}

else{//VOICE\_2

data->number\_of\_voice\_2\_packets\_processed++;

data->voice\_2\_packet\_accumulated\_delay += simulation\_run\_get\_time(simulation\_run) - this\_packet->arrive\_time;

}

data->accumulated\_delay += simulation\_run\_get\_time(simulation\_run) -

this\_packet->arrive\_time;

/\* Output activity blip every so often. \*/

output\_progress\_msg\_to\_screen(simulation\_run);

/\* This packet is done ... give the memory back. \*/

xfree((void\*) this\_packet);

/\*

\* See if there is are packets waiting in the buffer. If so, take the next one

\* out and transmit it immediately.

\*/

if(fifoqueue\_size(data->buffer) > 0) {

next\_packet = (Packet\_Ptr) fifoqueue\_get(data->buffer);

start\_packet\_transmission(simulation\_run, next\_packet);

}

}

/\*

\* This function ititiates the transmission of the packet passed to the

\* function. This is done by placing the packet in the server. The packet

\* transmission end event for this packet is then scheduled.

\*/

void

start\_packet\_transmission(Simulation\_Run\_Ptr simulation\_run,

Packet\_Ptr this\_packet)

{

Simulation\_Run\_Data\_Ptr data;

data = (Simulation\_Run\_Data\_Ptr) simulation\_run\_data(simulation\_run);

TRACE(printf("Start Of Packet.\n");)

server\_put(data->link, (void\*) this\_packet);

this\_packet->status = XMTTING;

/\* Schedule the end of packet transmission event. \*/

schedule\_end\_packet\_transmission\_event(simulation\_run,

simulation\_run\_get\_time(simulation\_run) + this\_packet->service\_time,

(void\*) this\_packet);

}

/\*

\* Get a packet transmission time. For now it is a fixed value defined in

\* simparameters.h

\*/

double

get\_packet\_transmission\_time(void)

{

return ((double) PACKET\_XMT\_TIME);

}

double

get\_voice\_packet\_transmission\_time(void)

{

return ((double) VOICE\_XMT\_TIME);

}

double

get\_voice\_2\_packet\_transmission\_time(void)

{

return ((double) VOICE\_2\_XMT\_TIME);

**<-- packet\_transmission.h -->**

void

start\_packet\_transmission(Simulation\_Run\_Ptr, Packet\_Ptr);

void

end\_packet\_transmission\_event(Simulation\_Run\_Ptr, void\*);

double

get\_packet\_transmission\_time(void);

double

get\_voice\_packet\_transmission\_time(void);

double

get\_voice\_2\_packet\_transmission\_time(void);

#endif /\* packet\_transmission.h \*/

**<-- simparameters.h -->**

//arrival rate 0 < lamda < 50: 5, 10, 20, 30, 40, 45

#define PACKET\_ARRIVAL\_RATE 45 /\* packets per second \*/

#define PACKET\_LENGTH 1e3 /\* bits \*/

#define LINK\_BIT\_RATE 1e6 /\* bits per second \*/

#define RUNLENGTH 10e6 /\* packets \*/

#define VOICE\_ARRIVAL\_TIME 20e-3

#define VOICE\_2\_ARRIVAL\_TIME 40e-3

#define VOICE\_PACKET\_LENGTH 1.776e3

#define VOICE\_2\_PACKET\_LENGTH 3.056e3

#define MEAN\_SERVICE\_TIME 20e-3

/\* Comma separated list of random seeds to run. \*/

#define RANDOM\_SEED\_LIST 123456,234567,345678,456789,5678901,6789123,7891234,8912345,1061556,1069639

#define PACKET\_XMT\_TIME ((double) PACKET\_LENGTH/LINK\_BIT\_RATE)

#define VOICE\_XMT\_TIME ((double) VOICE\_PACKET\_LENGTH/LINK\_BIT\_RATE)

#define VOICE\_2\_XMT\_TIME ((double) VOICE\_2\_PACKET\_LENGTH/LINK\_BIT\_RATE)

#define BLIPRATE (RUNLENGTH/1000)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#endif /\* simparameters.h \*/

**#6:**

**<-- clearnup\_memory.c -->**

void

cleanup\_memory (Simulation\_Run\_Ptr simulation\_run)

{

Simulation\_Run\_Data\_Ptr data;

Fifoqueue\_Ptr buffer, voice\_buffer;

Server\_Ptr link;

data = (Simulation\_Run\_Data\_Ptr) simulation\_run\_data(simulation\_run);

buffer = data->buffer;

voice\_buffer = data->voice\_buffer;

link = data->link;

if(link->state == BUSY) /\* Clean out the server. \*/

xfree(server\_get(link));

xfree(link);

while (fifoqueue\_size(buffer) > 0) /\* Clean out the queue. \*/

xfree(fifoqueue\_get(buffer));

xfree(buffer);

while (fifoqueue\_size(voice\_buffer) > 0) /\* Clean out the voice queue. \*/

xfree(fifoqueue\_get(voice\_buffer));

xfree(voice\_buffer);

xfree(data); /\* Clean up the simulation\_run data. \*/

simulation\_run\_free\_memory(simulation\_run); /\* Clean up the simulation\_run. \*/

}

**<-- main.c -->**

/\*

\* Create the packet buffer and transmission link, declared in main.h.

\*/

data->buffer = fifoqueue\_new();

data->voice\_buffer = fifoqueue\_new();

data->link = server\_new();

**.....**

/\*

\* Schedule the initial packet arrival for the current clock time (= 0).

\*/

schedule\_voice\_packet\_arrival\_event(simulation\_run,

simulation\_run\_get\_time(simulation\_run));

schedule\_voice\_2\_packet\_arrival\_event(simulation\_run,

simulation\_run\_get\_time(simulation\_run));

schedule\_packet\_arrival\_event(simulation\_run,

simulation\_run\_get\_time(simulation\_run));

**......**

// accumulated number of packets < RUNLENGTH

while(data->number\_of\_data\_packets\_processed + data->number\_of\_voice\_packets\_processed + data->number\_of\_voice\_2\_packets\_processed < RUNLENGTH) {

simulation\_run\_execute\_event(simulation\_run);

}

**<-- main.h -->**

Fifoqueue\_Ptr buffer, voice\_buffer;

**<-- packet\_arrival.c -->**

void

packet\_arrival\_event(Simulation\_Run\_Ptr simulation\_run, void\* ptr)

{

Simulation\_Run\_Data\_Ptr data;

Packet\_Ptr new\_packet;

data = (Simulation\_Run\_Data\_Ptr) simulation\_run\_data(simulation\_run);

data->arrival\_count++;

new\_packet = (Packet\_Ptr) xmalloc(sizeof(Packet));

new\_packet->arrive\_time = simulation\_run\_get\_time(simulation\_run);

new\_packet->service\_time = exponential\_generator((double) MEAN\_SERVICE\_TIME);

new\_packet->status = WAITING;

new\_packet->packet\_type = DATA;

/\*

\* Start transmission if the data link is free. Otherwise put the packet into

\* the buffer.

\*/

if(server\_state(data->link) == BUSY) {

fifoqueue\_put(data->buffer, (void\*) new\_packet);

} else {

// only start transmitting the packet data

// if there is no voice packets remain in the voice\_buffer

if(fifoqueue\_size(data->voice\_buffer) == 0)

start\_packet\_transmission(simulation\_run, new\_packet);

}

/\*

\* Schedule the next packet arrival. Independent, exponentially distributed

\* interarrival times gives us Poisson process arrivals.

\*/

schedule\_packet\_arrival\_event(simulation\_run,

simulation\_run\_get\_time(simulation\_run) +

exponential\_generator((double) 1/PACKET\_ARRIVAL\_RATE));

}

void

voice\_packet\_arrival\_event(Simulation\_Run\_Ptr simulation\_run, void\* ptr)

{

Simulation\_Run\_Data\_Ptr data;

Packet\_Ptr new\_packet;

data = (Simulation\_Run\_Data\_Ptr) simulation\_run\_data(simulation\_run);

data->arrival\_count++;

new\_packet = (Packet\_Ptr) xmalloc(sizeof(Packet));

new\_packet->arrive\_time = simulation\_run\_get\_time(simulation\_run);

new\_packet->service\_time = get\_voice\_packet\_transmission\_time();

new\_packet->status = WAITING;

new\_packet->packet\_type = VOICE;

/\*

\* Start transmission if the data link is free. Otherwise put the packet into

\* the buffer.

\*/

if(server\_state(data->link) == BUSY) {

fifoqueue\_put(data->voice\_buffer, (void\*) new\_packet);

} else {

start\_packet\_transmission(simulation\_run, new\_packet);

}

/\*

\* Schedule the next packet arrival. Independent, exponentially distributed

\* interarrival times gives us Poisson process arrivals.

\*/

schedule\_voice\_packet\_arrival\_event(simulation\_run,

simulation\_run\_get\_time(simulation\_run) +

VOICE\_ARRIVAL\_TIME);

}

void

voice\_2\_packet\_arrival\_event(Simulation\_Run\_Ptr simulation\_run, void\* ptr)

{

Simulation\_Run\_Data\_Ptr data;

Packet\_Ptr new\_packet;

data = (Simulation\_Run\_Data\_Ptr) simulation\_run\_data(simulation\_run);

data->arrival\_count++;

new\_packet = (Packet\_Ptr) xmalloc(sizeof(Packet));

new\_packet->arrive\_time = simulation\_run\_get\_time(simulation\_run);

new\_packet->service\_time = get\_voice\_2\_packet\_transmission\_time();

new\_packet->status = WAITING;

new\_packet->packet\_type = VOICE\_2;

/\*

\* Start transmission if the data link is free. Otherwise put the packet into

\* the buffer.

\*/

if(server\_state(data->link) == BUSY) {

fifoqueue\_put(data->voice\_buffer, (void\*) new\_packet);

} else {

start\_packet\_transmission(simulation\_run, new\_packet);

/\*

\* Schedule the next packet arrival. Independent, exponentially distributed

\* interarrival times gives us Poisson process arrivals.

\*/

schedule\_voice\_2\_packet\_arrival\_event(simulation\_run,

simulation\_run\_get\_time(simulation\_run) +

VOICE\_2\_ARRIVAL\_TIME);

}

**<-- packet\_transmission.c -->**

/\*

\* See if there is are packets waiting in the buffer. If so, take the next one

\* out and transmit it immediately.

\*/

if(fifoqueue\_size(data->voice\_buffer) > 0) {

next\_packet = (Packet\_Ptr) fifoqueue\_get(data->voice\_buffer);

start\_packet\_transmission(simulation\_run, next\_packet);

}else if(fifoqueue\_size(data->buffer) > 0){

next\_packet = (Packet\_Ptr) fifoqueue\_get(data->buffer);

start\_packet\_transmission(simulation\_run, next\_packet);

}