Queueing Simulation with OMNet++

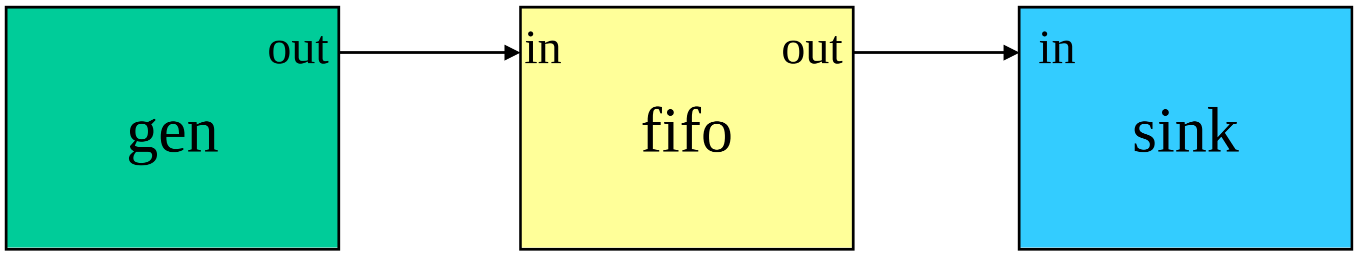
Chang-Gun Lee (cglee@snu.ac.kr)

Assistant Professor

The School of Computer Science and Engineering

Seoul National University

Single Queue System
- Modules and Connections -



-schedule next arrival

-send job

-service or queueing

-schedule completion

-send job completion

-collect stat.

Gen Module

- initialize
  - schedule the first job arrival
  - use “scheduleAt”
- handleMessage
  - only one message type – self message (job arrival)
  - send the job to the server: send(msg, “out”)
  - schedule the next arrival: scheduleAt

Fifo Module

- handleMessage
  - What events?
    - two message types
      - job arrival from Gen
      - job completion (self message)
  - for job arrival
    - if busy, insert queue
    - if idle, start service and schedule “job completion” (self message)
  - for job completion
    - send job completion to Sink
    - if queue is empty, make the server idle
    - else, get the next job from the queue, start service, and schedule “job completion”

Sink Module

- handleMessage
  - only one message types - job completion message from Fifo
  - collect “service delay” of completed jobs

.ned files

gen.ned

```
simple FFGenerator
    parameters:
        sendIaTime : numeric,
        serviceTime : numeric;
    gates:
        out: out;
endsimple
```

fifo.ned

```
simple FFJobFifo
    gates:
        in: in;
        out: out;
endsimple
```

sink.ned

```
simple FFSink
    gates:
      in: in;
endsimple
```

fifonet.ned

```
import
    "gen",
    "fifo",
    "sink";

module FifoNet
    submodules:
        gen: FFGenerator;
           display: "p=89,100;i=block/source";
        fifo: FFJobFifo;
           display: "p=209,100;…..
        sink: FFSink;
            display: "p=329,100;i=block/sink";
    connections:
        gen.out --> fifo.in;
        fifo.out --> sink.in;
endmodule

network fifonet : FifoNet
endnetwork
```

gen.cc

```
class FFGenerator : public cSimpleModule
{
  private:
    cMessage *sendMessageEvent;

  public:
     FFGenerator();
     virtual ~FFGenerator();

  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

Define_Module(FFGenerator);

FFGenerator::FFGenerator()
{    sendMessageEvent = NULL;}

FFGenerator::~FFGenerator()
{    cancelAndDelete(sendMessageEvent);}
```

```
void FFGenerator::initialize()
{
    sendMessageEvent = new
              cMessage("sendMessageEvent");
    scheduleAt(0.0, sendMessageEvent);
}

void FFGenerator::handleMessage(cMessage
*msg)
{
    ASSERT(msg==sendMessageEvent);

    cMessage *m = new cMessage("job");
    m->setTimestamp(par("serviceTime")); //
API ref to check available member functions
    send(m, "out");

    scheduleAt(simTime()+
          (double)par("sendIaTime"),
          sendMessageEvent);
}
```

fifo.cc

```
class FFJobFifo : public cSimpleModule
{
  protected:
    cMessage *msgServiced;
    cMessage *endServiceMsg;
    cQueue queue; // container class
       ……..
  };

void FFJobFifo::handleMessage(cMessage *msg)
{
    if (msg==endServiceMsg)  { // completion
        endService( msgServiced );
        if (queue.empty()){
            msgServiced = NULL;
        }
        else{
            msgServiced =  queue.getTail();
            simtime_t serviceTime =
startService( msgServiced );
            scheduleAt( simTime()+serviceTime,
endServiceMsg );
        }
    }
    else if (!msgServiced) { // job arrival when idle
        arrival( msg );
        msgServiced = msg;
        simtime_t serviceTime =
                    startService( msgServiced );
        scheduleAt( simTime()+serviceTime,
                     endServiceMsg );
    }
    else  { // job arrival when busy
        arrival( msg );
        queue.insert( msg );
    }
}

simtime_t FFJobFifo::startService(cMessage *msg)
{
        return msg->timestamp();
}

void FFJobFifo::endService(cMessage *msg)
{
        send( msg, "out" );
}
```

sink.cc

```
class FFSink : public cSimpleModule
{
  private:
    cStdDev qstats;
    cOutVector qtime;

  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    virtual void finish();
};

Define_Module( FFSink );

void FFSink::initialize()
{
    qstats.setName("queueing time stats");
    qtime.setName("queueing time vector");
}
```

```
  void FFSink::handleMessage(cMessage *msg)
{
    double d = simTime()-msg->creationTime();
    ev << "Received " << msg->name() << ",
queueing time: " << d << "sec" << endl;
    qstats.collect( d );
    qtime.record( d );
    delete msg;
}

void FFSink::finish()
{
    ev << "Total jobs processed: " << qstats.samples()
<< endl;
    ev << "Avg queueing time:    " << qstats.mean()
<< endl;
    ev << "Max queueing time:    " << qstats.max()
<< endl;
    ev << "Standard deviation:   " << qstats.stddev()
<< endl;
}
```

Demo 1 (myFifo)

How to run without GUI

- Change USERIF_LIBS in Makefile
- Remake
- Let’s see demo

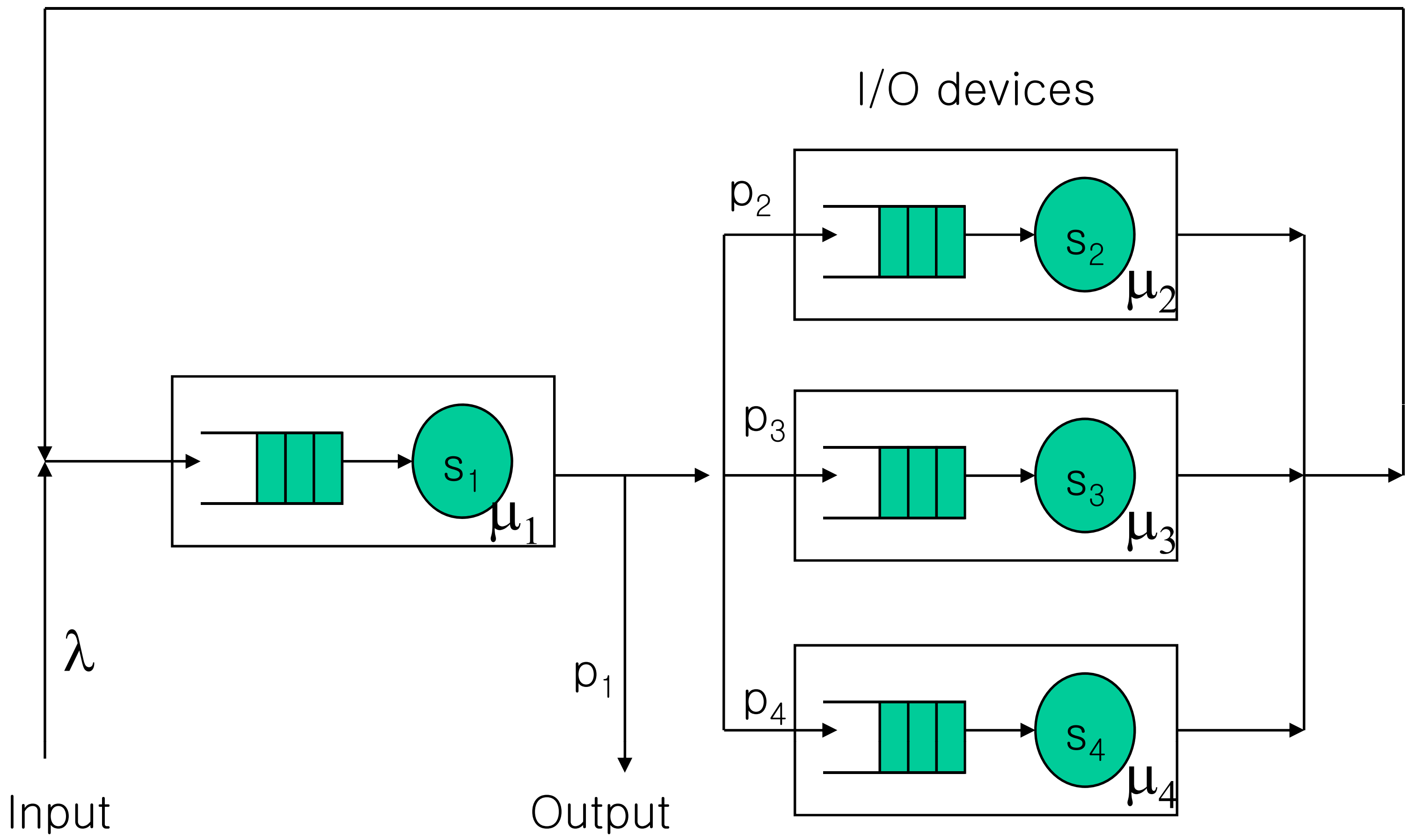
How to run without crazy text outputs

- Change “module-message” and “event-banners” of omnetpp.ini to “no”
- Don’t need Remake
- Let’s see demo

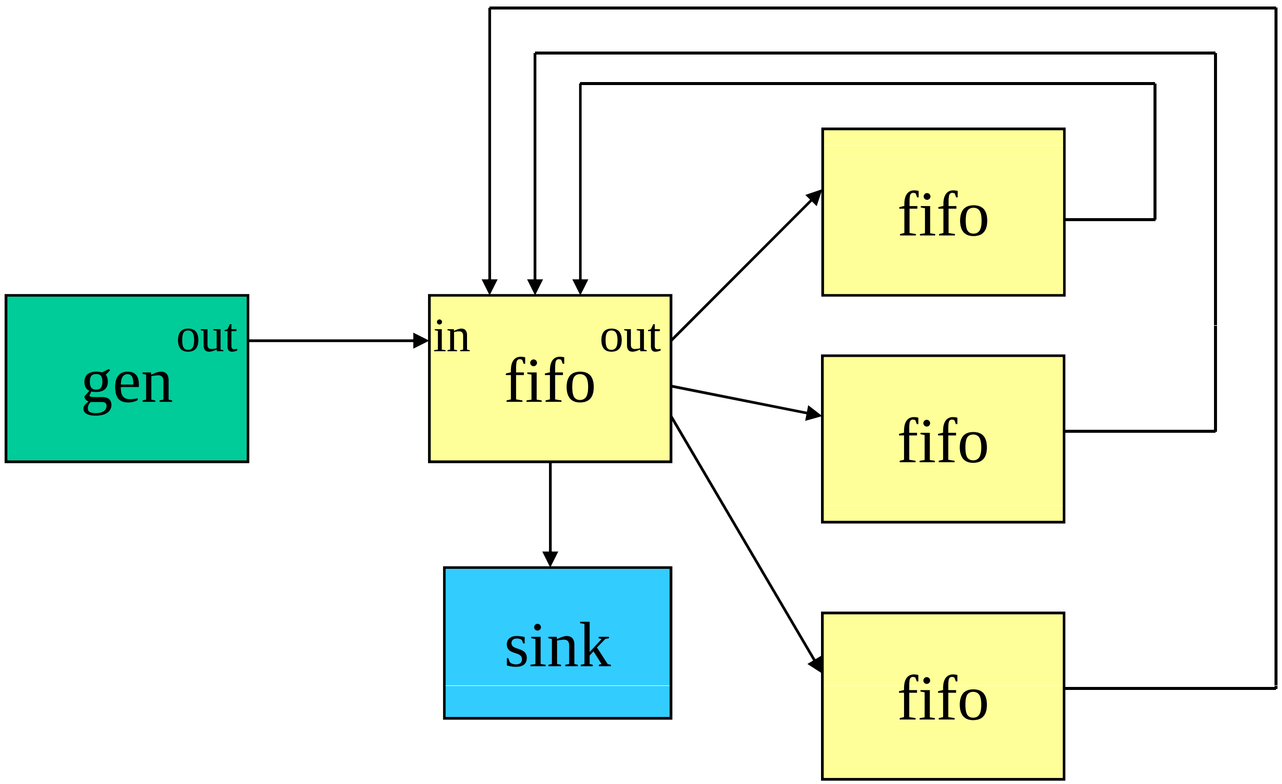
Homework 4

- Draw the following three graphs by OMNet++ simulation
  - traffic load (0-0.95) vs. average # of jobs in the system
  - traffic load (0-0.95) vs. average queue length
  - traffic load (0-0.95) vs. average queueing delay for each job
- Compare the three graphs with the graphs of Homework 1
- When simulating the system with a traffic load 1.0 and 1.5,
  - Observe what happens in your simulation
  - Discuss how to handle such situation

Queueing Network

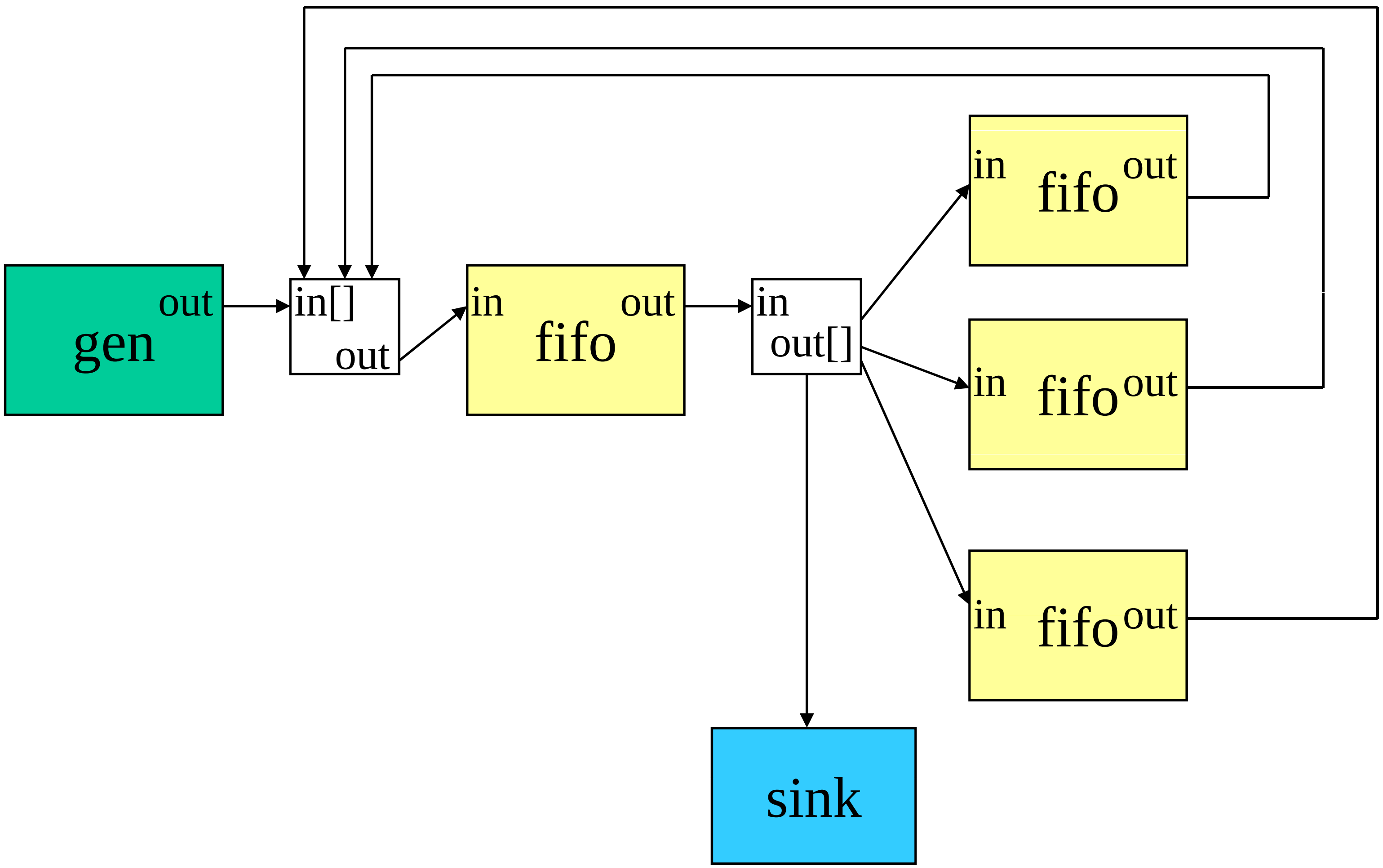


Queueing Network
- Modules and Connections -



Any Problem?

Better Approach



Homework 5

- Let
  - Service rates: μ_1=1/3, μ_2=1/1, μ_3 = 1/2, μ_4 =1/4
  - Job transferring probs: p_1=p_2=p_3=p_4=0.25
- As increasing λ (x-axis) from 1/20 to 1/1, draw the followings by Simulation
  - L1, L2, L3, L4, W1, W2, W3, W4, L, W
  - Utilization of each server
- Draw the same curves by analysis
- Answer the followings
  - Which server is the bottleneck that first makes the system unstable
  - How to reassign server rates so that λ can be maximized while keeping the system stable