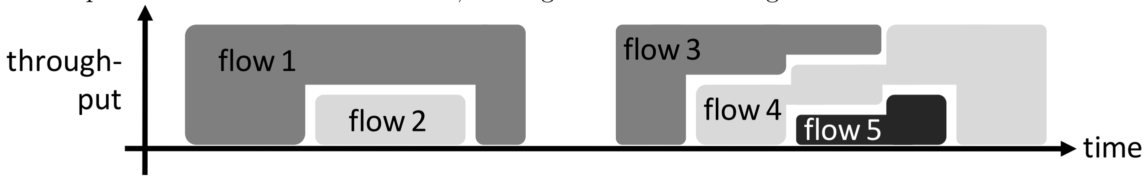
Simulation of a processor sharing link

Network Performance—DJW—2011/12

Coursework from 2010/2011

Here is a simple model for a single bottleneck link shared by several TCP flows. New flows arrive at random times, and each flow has a random amount of data to send, measured in Mb. The flows share the link fairly: when there are n flows, they each get throughput C/n where C is the link speed, measured in Mb/s. Once a flow's data has all been sent, the flow departs. For example, over a period in which five flows arrive, we might see the following:



Question 1. Program a simulator of this system. Your written report should include the source code. It should also include a brief description of how your simulator works—imagine you are giving instructions to a colleague who knows how to program but who knows nothing about simulation or about networks.

The obvious performance measure that matters to users is the *average flow completion time*, i.e. the average length of time that each flow is active. To investigate this, we will use a random number generator for interarrival times and for flow sizes. The following Python code generates a random number called $\mathsf{Exp}(\lambda)$:

```
import math, random
def rexp(λ): return -1.0/λ * math.log(random.random())
```

If the times between flow arrivals are $\mathsf{Exp}(\lambda)$ seconds, then the average arrival rate is λ arrivals per second. If the flow sizes are $\mathsf{Exp}(1/m)$ Mb, then the average flow size is m Mb.

Question 2. Run simulations with link speed 10Mb/s, flow sizes $\mathsf{Exp}(1)$Mb, and interarrival times $\mathsf{Exp}(\lambda)$ seconds for a range of values of λ in the range 0 to 20. Measure the average flow completion time, and plot it as a function of arrival rate.

Internet measurements suggest that a better distribution to use for flow sizes is given by the following type of random number, called $\mathsf{Pareto}(\alpha,m)$:

```
def rpareto(α,m): return m*(1-1.0/α)*math.pow(random.random(),-1.0/α)
```

If the flow sizes are $\mathsf{Pareto}(\alpha,m)$ Mb and $\alpha > 1$ then the mean flow size is m Mb.

Question 3. Repeat the experiment, but with $\mathsf{Pareto}(\alpha,1)$ flow sizes, for a selection of values of $\alpha > 1$. Plot a graph with average flow duration on the vertical axis, arrival rate on the horizontal axis, and several lines, one for each value of α.

Bad solution, by A. Student

Simulator

The simulator code is given below. The main `sim` function takes four arguments: a generator for interarrival times (`arrivals`) in seconds, a generator for flow sizes (`flowsizes`) in Mb, the link speed in Mb/s (`linkspeed`), and a logging function `logfunc`. The logging function takes one argument, a pair (departuretime,starttime), and is called whenever a flow terminates. I made the interface to `sim` generic in this way, to make it easy to modify the simulator to use different interarrival times and flow sizes, and also so that the logging code is kept separate from the simulator logic.

The simulator keeps track of the currently active flows in a list `activeflows`. For each active flow there is an entry in this list, consisting of a pair `(sizeleft,starttime)` where `sizeleft` is the amount of data left to transmit and `starttime` is when the flow started. As the simulator runs `sizeleft` decreases (line 12), but `starttime` is left unchanged and is only there for logging purposes. The `activeflows` list is kept sorted using the `heapq` Python package: this ensures that the first entry in the list is always the flow with the least left to transmit, i.e. the flow that will finish first.

The operation of the simulator is as follows. It maintains a variable `simtime`, the current (simulated) time. It also keeps track of the next scheduled flow arrival, and it calculates the time of the next departure. It works out which of these two is scheduled to happen next, and advances `simtime` correspondingly (line 11). It works out the throughput given to each of the currently active flows in the meantime, and subtracts the appropriate amount from their `sizeleft` record (lines 12–13). If the next scheduled event is a departure then the flow to depart is removed and logged (lines 17–19). Otherwise the next scheduled event is an arrival, so we generate a new flow and add it to `activeflows`, and work out the next scheduled arrival (lines 22–24). If there are no active flows, then the next scheduled event must be an arrival (line 15). It repeats this for a fixed length of simulated time (line 6). I chose the simulation time so that my program runs at a reasonable speed.

```python
 1 import heapq
 2 def sim(arrivals,flowsizes,linkspeed,logfunc):
 3     simtime = 0.0
 4     activeflows = []
 5     nextarr = arrivals.next()
 6     while simtime<100:
 7         if activeflows:
 8             throughputperflow = linkspeed/float(len(activeflows))
 9             smallestflowsize = activeflows[0][0]
10             nextdep = simtime + smallestflowsize/throughputperflow
11             elapsedtime,simtime = min(nextdep,nextarr)-simtime, min(nextdep,nextarr)
12             activeflows = [(sizeleft-throughputperflow*elapsedtime,starttime)
13                            for sizeleft,starttime in activeflows]
14         else:
15             nextdep,simtime = float('inf'),nextarr
16         if nextdep<=nextarr:
17             departingflow = heapq.heappop(activeflows)
18             departingflowstarttime = departingflow[1]
19             logfunc( (simtime,departingflowstarttime) )
20         else:
21             try:
22                 newflowsize = flowsizes.next()
23                 heapq.heappush(activeflows, (newflowsize,simtime))
24                 nextarr = simtime + arrivals.next()
25             except StopIteration:
26                 break
```

The simulator harness uses the utility functions below. The `rexp(λ)` function is a generator which generates Exp(λ) random numbers, as specified in the assignment. I have specified the random seed to be used for random numbers in order to make my simulation results reproducible (line 29); without this it is very hard to track down bugs in a simulator based on random numbers. The `LogDuration` class provides a function `logdeparture` (line 31), which can be passed to `sim`; `sim` will then call `logdeparture((departuretime,starttime))` for each flow when it departs, and the logging class will keep a running total.

```python
27 import math, random
28 def rexp(rate):
29     random.seed('geranium')
30     while True:
31         yield -1.0/rate * math.log(random.random())
32
33 class LogDuration:
```

```
34      def __init__(self): self.tot, self.n = 0,0
35      def logdeparture(self,record):
36          deptime,starttime = record
37          self.tot = self.tot+(deptime-starttime)
38          self.n = self.n+1
```

Finally, the actual simulator is run as follows. This runs simulations for a range of arrival rates. It stores a list consisting of records which are pairs, `(arrivalrate, average flow completion time)` (line 43). When it has finished, it writes all the results to a plain text file (lines 44–47), which can be loaded into a graphing package.

```
39  results = []
40  for arrivalrate in [(x+1)/10.0 for x in range(200)]:
41      logger = LogDuration()
42      sim(rexp(arrivalrate),rexp(1),10, logger.logdeparture)
43      results.append( (arrivalrate,logger.tot/logger.n) )
44  with open('results.csv','wt') as f:
45      f.write('arrivalrate,meanflowduration\n')
46      for res in results:
47          f.write( ','.join([str(s) for s in res]) +'\n')
```

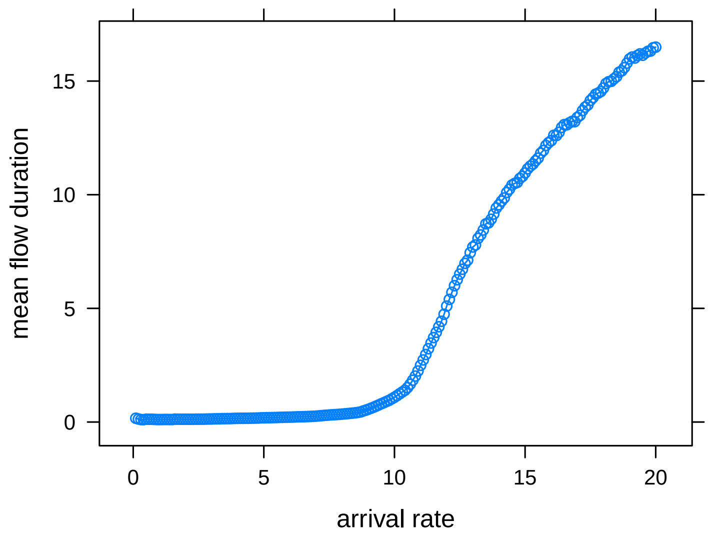
Results

I used R to plot the simulator output. Here is some of the output:

```
> res <- read.csv("results.csv")
> res[1:5, ]
```

```
  arrivalrate meanflowduration
1         0.1        0.1653807
2         0.2        0.1331617
3         0.3        0.1085765
4         0.4        0.1057340
5         0.5        0.1254007
```

```
> library(lattice)
> print(xyplot(meanflowduration ~ arrivalrate, data = res, xlab = "arrival rate",
+     ylab = "mean flow duration", type = "b"))
```


mean flow duration
arrival rate