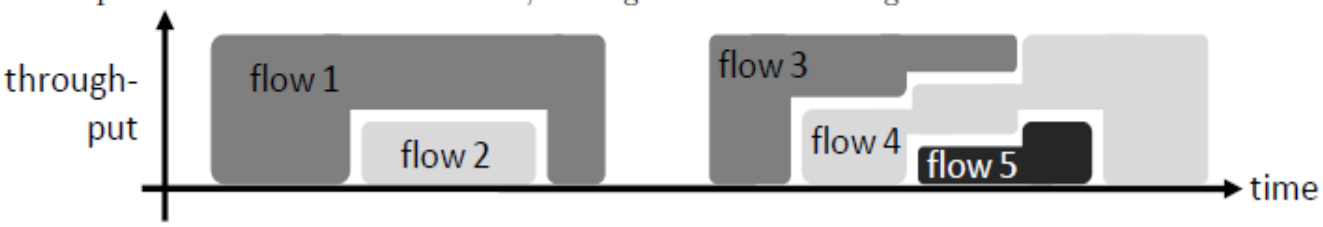
§2 Simulation

For this section, we will work with this running example:

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:



Let's suppose we're interested in two quantities:

- how many flows are active, typically?
- what is the typical completion time (time it takes from when the flow starts to when it finishes)?

How do these quantities depend on the system parameters:

- the link speed C
- the typical size of a flow, in Mb
- the flow arrival process (ie the times that flows arrive)?

§ 2.1 Types of simulation

DISCRETE-TIME SIMULATOR, ALSO CALLED SLOTTED-TIME

```
timestep = 0.1
activeflows = []
simtime = 0
while True:
    # generate a random number of new flows to start this timestep
    n = numnewflows(timestep)
    # for each new flow, add it to the list of active flows
    for i in range(n): activeflows.append( randomfilesize() )
    # do timstep worth of work to each flow
    throughputperflow = C/len(activeflows)
    activeflows = [sizeleft - throughputperflow*timestep for sizeleft in activeflows]
    # remove any flow that has finished
    activeflows = [sizeleft for sizeleft in activeflows if sizeleft<=0]
    # advance the clock
    simtime = simtime+timestep
```

Issue: how to choose timestep?

- If it's too small, then numnewflows(timestep) will typically be 0, so we waste lots of time iterating when we could just as well do lots of steps of the loop in one go.
- But if it's too big, then the simulation is unfaithful — it doesn't take proper account of flow departure time.

EVENT-DRIVEN SIMULATOR

This simulator only "wakes up" when something interesting changes. It doesn't need us to pick an arbitrary parameter like timestep above.

```
activeflows = []
simtime = 0
nextarrival = first scheduled arrival time
while True:
    # Assuming no new arrivals in the interim, when is the next departure?
    throughputperflow = C/len(activeflows)
    nextdeparture = simtime + min(activeflows)/throughputperflow
    # Advance the clock to the next 'interesting' event
    elapse = min(nextarrival,nextdeparture) - simtime
    activeflows = [sizeleft - throughputperflow*elapse for sizeleft in activeflows]
    simtime = simtime + elapse
    # Work out what the interesting event is: is it an arrival or a departure?
    if nextarrival<nextdeparture: # It's an arrival: generate a new flow
        activeflows.append( randomfilesize() )
    else: # it's a departure: remove the departing flow
        activeflows.removesmallest()
```

(For a non-buggy version of this pseudocode, see separate handout).

Issue:
This is a faithful simulator — no approximations, no nuisance parameters to pick. However, the loop runs once per event, ie once for every single arrival or departure, and this can get very slow.
If we were simulating water in a pipe, we wouldn't simulate every single molecule; indeed, even in this simulator, we're not modeling individual packets, only "flow rates" i.e. smoothed-out aggregates of packets.
Is there a corresponding way to model aggregates of flows?

FLUID SIMULATOR

Mathematical analysis tells us that "large aggregates" tend to behave predictably. In this case, if $n(t)$ = # active flows at time t, a reasonable approximation (when $n(t)$ is large) is

$$\frac{d}{dt} n(t) = \text{drift}(n(t)), \quad \text{for some suitable function drift()}^*.$$

* §3 is all about these drift functions.

In other words, for small δ, $\quad n(t+\delta) \approx n(t) + \delta \times \text{drift}(n(t))$.

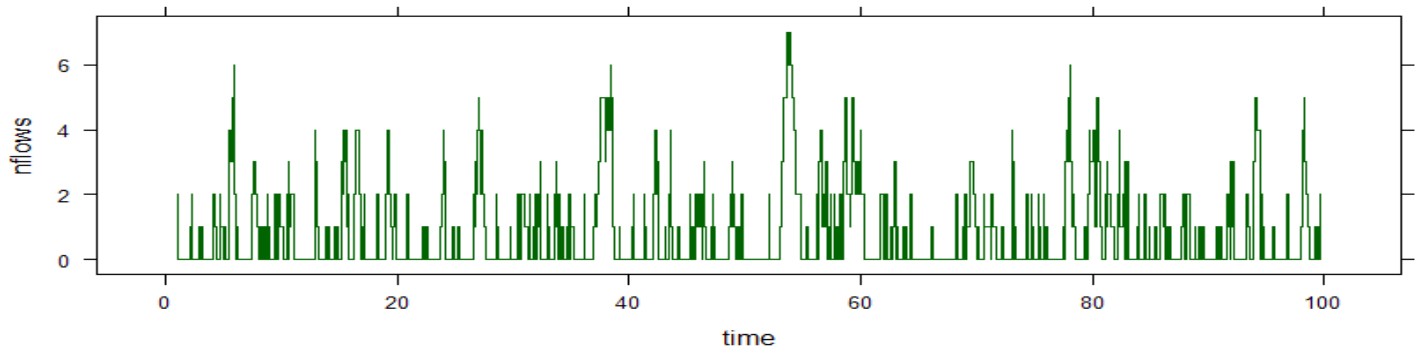
```
simtime = 0
delta = 0.01
n = initial number of flows
while True:
    currentdrift = drift(n)
    n = n + delta*currentdrift
    simtime = simtime + delta
```

Issues:

- How to choose δ? If we choose it too big then n jumps about erratically; if we choose it too small then we waste a lot of time iterating. Good maths libraries have routines for choosing δ adaptively — search for "ODE solver".
- This model ignores randomness and it doesn't bother keeping track of individual flows. It's very fast — but it's only as good as the underlying mathematical approximation.

§ 2.2 Where you measure from

Suppose I want to learn the "typical" number of active flows. This number varies — it's random. So, what is it's distribution?



Here are three different "vantage points" from which to measure the distribution of "number of active flows".

- Periodic Sampling.
  Every 1s, say, record the number of active flows. Define

$$\pi^P(x) = \frac{\#\text{ of samples that found } x \text{ active flows}}{\text{total \# of samples.}}$$

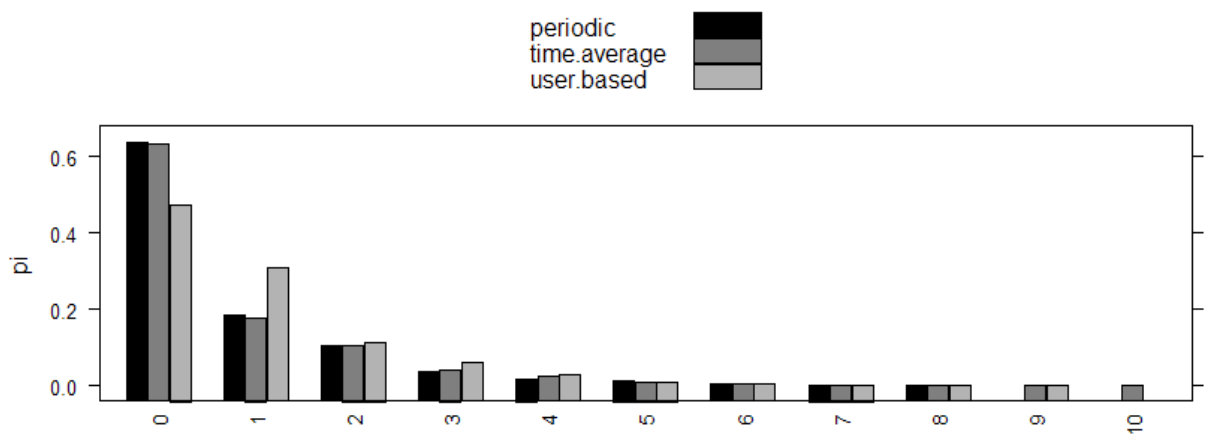
- Time-average.
  Record the full simulation trace. Define

$$\pi^T(x) = \frac{\text{amount of time for which there are } x \text{ active flows}}{\text{total simulation time}}$$

- User-based sampling.
  Every time a flow arrives, record the no. of active flows immediately beforehand. Let

$$\pi^U(x) = \frac{\#\text{ of flows that arrived to find } x \text{ active flows}}{\text{total \# of flow arrivals.}}$$

These three can give different answers:



The three are all legitimate metrics, they just measure different things.

e.g. You're sitting in the pub, watching the bar. You see that most of the time there's no-one there.
$\longrightarrow \pi^T(0)$ is large.
Or, you look up from your drink from time to time, and see the same.
$\longrightarrow \pi^P(0)$ is large.

The publican is probably concerned that the bartender is underworked.
The bartender is happy to have time free.

However, whenever you arrive at the pub, you always seem to find a big queue
$\longrightarrow \pi^u(0)$ is small.

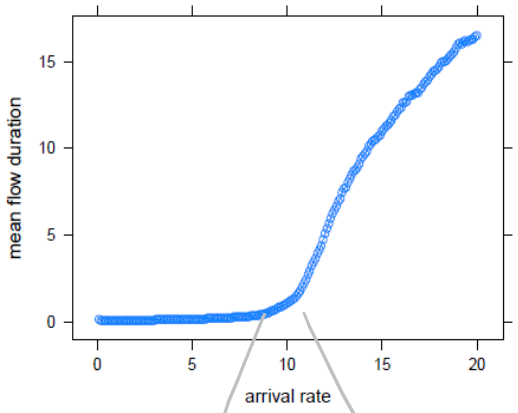
Customers are unhappy to be made to wait.

Exercise

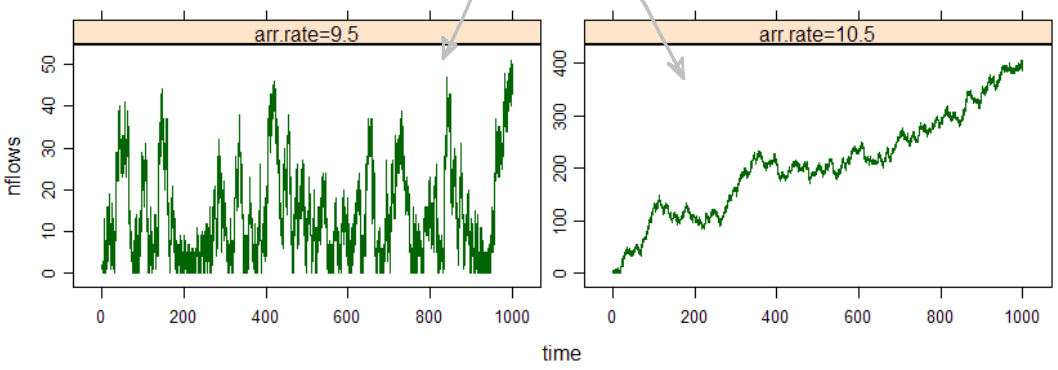
- Adapt the simulator to measure these three different distributions.
- Find an arrival process that makes $\pi^T \neq \pi^u$.
  Find an arrival process that makes $\pi^T \neq \pi^P$.
  In each case, explain why.

§2.3 Steady State

A student produced this plot, of average flow completion time as a function of the arrival rate of new flows:

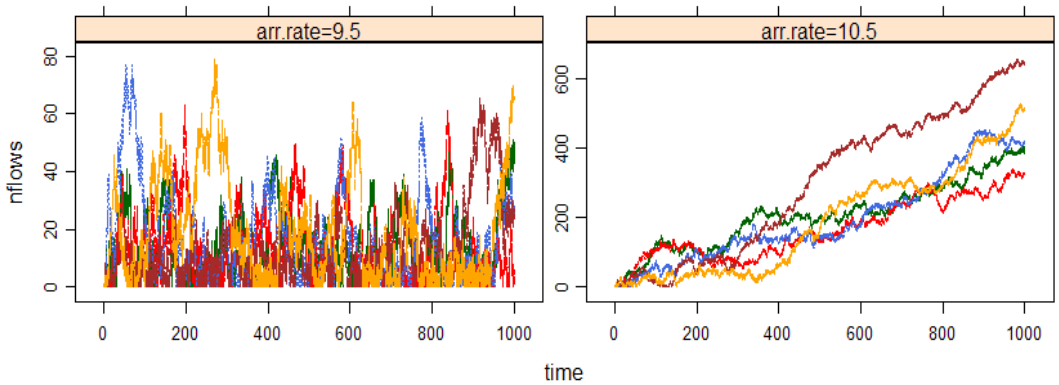


But this plot hides some very different behaviours ....

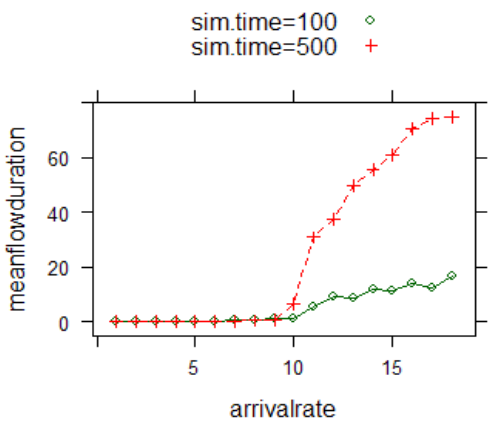


Note: very different y-scales.

Is this just a fluke? Let's repeat 5 times ...



Another way to spot the issue: run the experiment twice, once with small simtime, once with longer simtime. Do they differ?



EQUILIBRIUM

Am I simulating something that is stable or unstable?

To check:
- look at traces — do they grow steadily?
- Does your experiment give different answers when you run it longer?

"Unstable" means that the state keeps growing, the longer you run the system. Also called secular.

In such a situation, it's dumb to report a quantity like "average flow completion time"; all it tells you is how long you ran the simulation.

In an unstable system, you should report growth rates: how rapidly does #active flows increase?

"stable" means that the state settles down, and once it's settled down we can report its distribution. This is called the equilibrium distribution.

- The eqm. distribution is stationary, ie if at some time the system has reached equilibrium, then it stays in equilibrium thereafter.
- The eqm. distribution is limiting, ie no matter where you start, you approach the equilibrium distribution the longer you run the system.

Does my simulator show periodic behaviour?

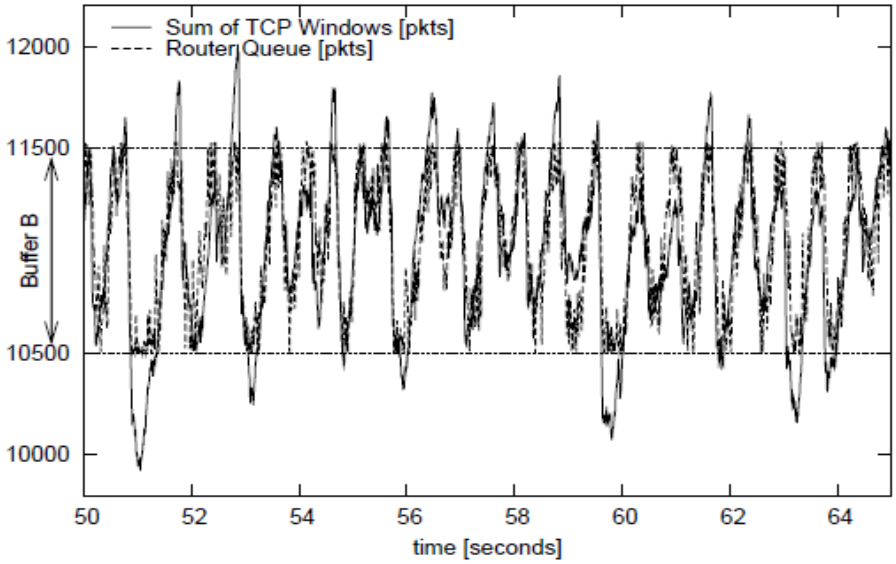
The system may settle down into a periodic cycle. If it does, you have to be careful about sampling artefacts: π^P and π^T may differ if the sampling period is badly chosen.

Has my simulator reached equilibrium yet?

To check: eyeball. There is no systematic rule for how long it takes, or how to tell if you've reached eqm.

The atypical initial readings you get when you start from some arbitrary initial state are called transients. You should discard them, i.e. you should run-in your simulator before you start measuring equilibrium values.

It's also interesting to study how long it takes to reach equilibrium, from different initial states



This is a plot of queue size at an Internet router, from a 2004 paper on why router buffers should be much smaller than was commonly thought.

Does it show natural (random fluctuations), or does it show periodic oscillations?

QUASI-EQUILIBRIUM

Can I trust your findings? Are they a good guide to future behaviour, or are they just a fluke?

First thought: I'll just run my simulation longer. This'll tell me if my first 100 / 1000 / 10^6 / whatever seconds of simulated time where a fluke. (And, as a bonus, I only need to waste run-in time once.) Or, I could split my simulation run into say 10 chunks, and compare them. If they're consistent, it's unlikely to be a fluke.

But here's a thought experiment which proves that just running your simulator longer isn't enough:

```
def sim2(λ, m, c):
    linkspeed = (2/3 + 2/3 * random.random) * c
    sim(rexp(λ), rexp(1/m), linkspeed).
```

This simulator converges to a limiting distribution the longer you run it — but each time you run it, you get a different limiting distribution!

In this case it's easy to see the problem. In practical/interesting simulators, it's far harder to spot....

You should usually run multiple independent simulations ("trials") and check that the answers are consistent.

There is one situation where multiple trials are unnecessary:
A system is said to be ergodic if time-averages (in a single trial) agree with run-averages (across multiple trials), in the limit as simulation run length tends to infinity.

> If a system is stable, and if it's possible to reach any state from any other state, then the system is ergodic.

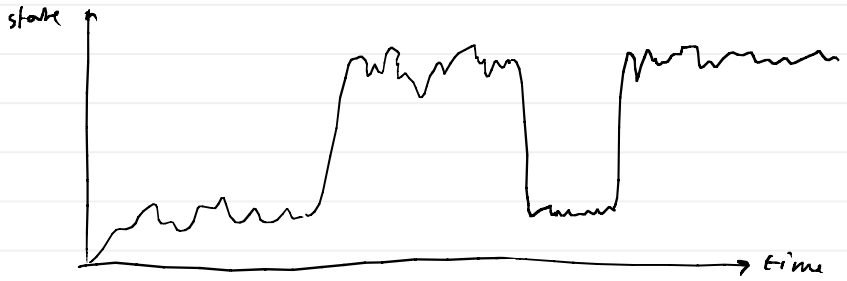
Well-behaved stable ergodic systems can still throw up surprises:

Congestion Avoidance and Control*

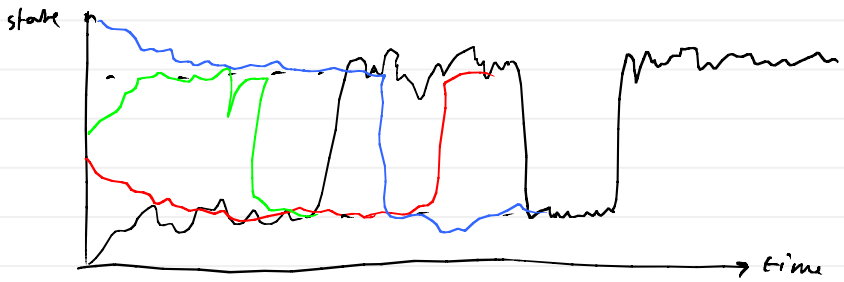
Van Jacobson†
Lawrence Berkeley Laboratory

In October of '86, the Internet had the first of what became a series of 'congestion collapses'. During this period, the data throughput from LBL to UC Berkeley (sites separated by 400 yards and two IMP hops) dropped from 32 Kbps to 40 bps. We were fascinated by this sudden factor-of-thousand drop in bandwidth and embarked on an investigation of why things had gotten so bad. In particular, we wondered if the 4.3BSD (Berkeley UNIX) TCP was mis-behaving or if it could be tuned to work better under abysmal network conditions. The answer to both of these questions was "yes".

Another system, which we'll analyse later, has two semi-stable states, and it flips between them at random times. If your simulation is too short, you may never see a flip — you'll probably be misled by the initial seemingly-nice equilibrium.



You should run a variety of simulations with different initial conditions, in the hope of catching this.



But it's extremely hard to anticipate this, without the mathematical analysis tools we'll develop in §3.

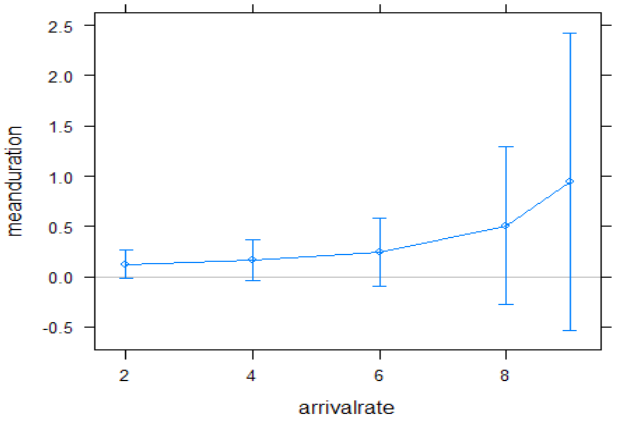
§ 2.4 Replication

Can I trust that your findings are a good guide to future performance, or are they just a fluke?

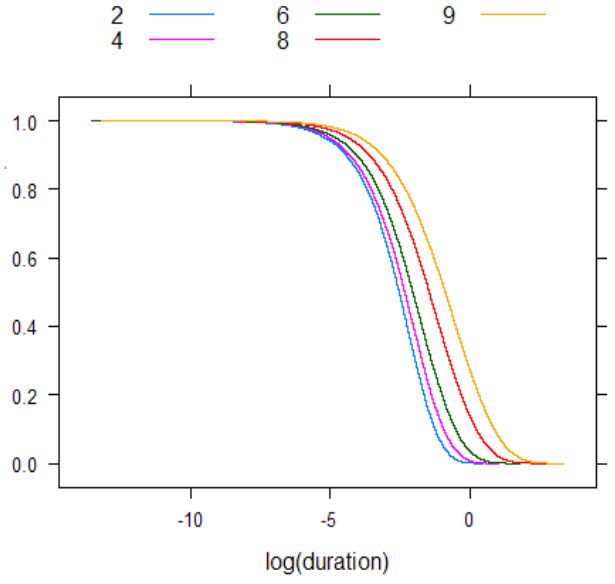
FROM A SINGLE RUN

Suppose I'm interested in typical flow completion time. Naturally, I'd measure lots of them, say $x_1, \cdots, x_n$, and report their aggregate statistics:

- The sample mean $\bar{X} = \frac{1}{n}(x_1 + \cdots + x_n)$
  $= \sum_{\substack{\text{all obs} \\ x}} x \, \hat{\mathbb{P}}(X = x)$, where $\hat{\mathbb{P}}(X = x) = \frac{\#\text{ of times I measured } x}{\text{total \# of measurements}}$
- The sample variance $s_{xx} = \frac{1}{n} \sum (x_i - \bar{x})^2 = \sum_{\text{all obs } x} (x - \bar{x})^2 \, \hat{\mathbb{P}}(X = x)$
- The empirical distribution function $\hat{F}(x) = \frac{\#\text{ occurrences} \le x}{\text{total \# samples}}$
- An empirical 95% confidence interval, ie a range $[x_1, x_2]$ such that 95% of my samples lie in this range (read it off the EDF plot)
- It's rarely worth reporting the sample min and max, because the more readings you take the more extreme min and max will likely be. In other words, min and max say more about your simulation experiment than about likely future behaviour. The other four quantities on the other hand just get more and more accurate the more samples you take.



mean flow completion time, as a function of arrival rate. The error bars show $\pm \hat{\sigma}$ where $\hat{\sigma} = \sqrt{s_{xx}}$ = sample standard deviation.

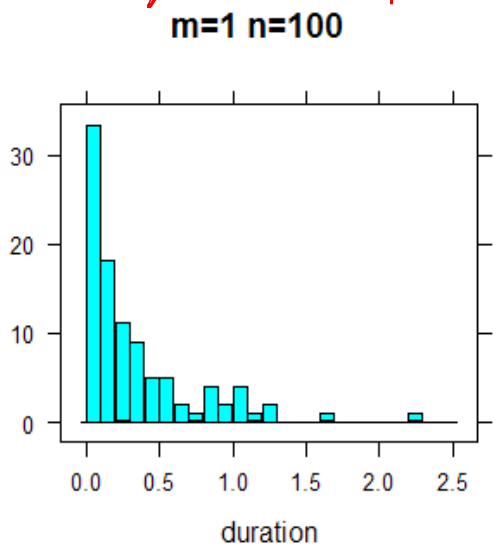


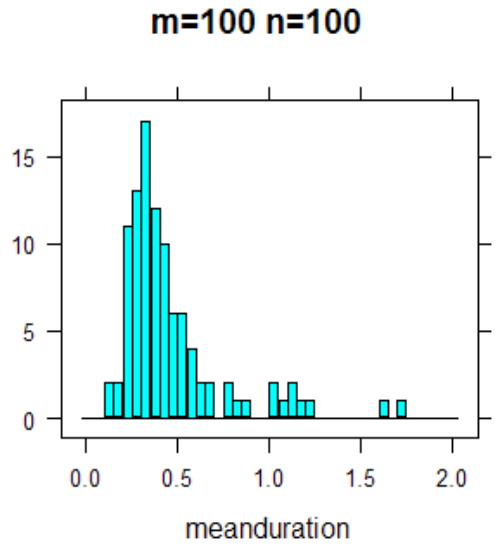
Empirical distribution curves, for a range of arrival rates. I've log-transformed the x axis, to see more detail.

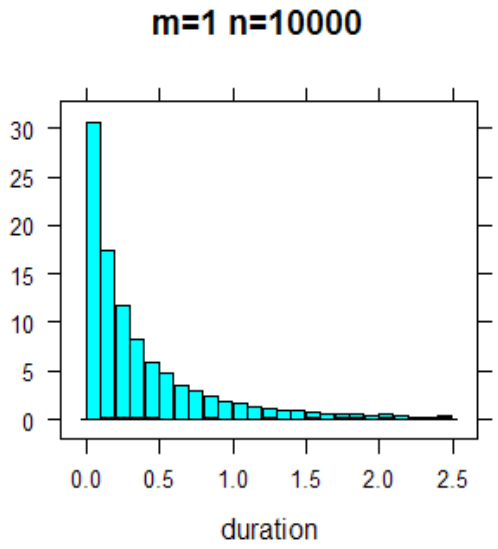
FROM MULTIPLE RUNS

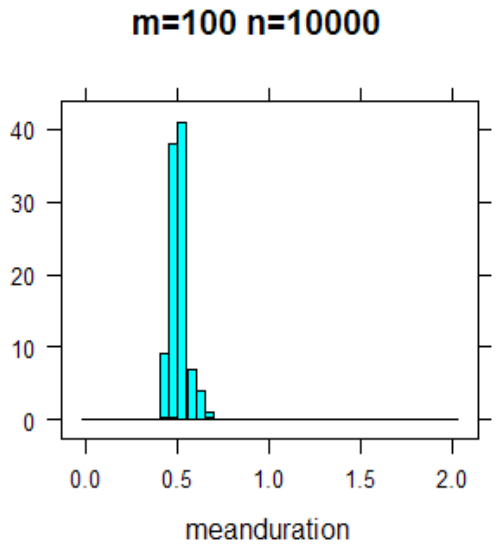
The average flow completion time $\bar{X}$ from a single run may be a fluke. To check this, do multiple runs, and measure the average flow completion time for each, say $\bar{X}_1, \bar{X}_2, \cdots, \bar{X}_m$. If they're all consistent, we can trust them.

⚠ Make sure you don't confuse sample values with sample means...









n samples taken from a single trial. The longer you run the simulation, the more samples you have, and the more accurately you know the distribution of flow completion time — e.g. you get a more and more accurate estimate of its standard deviation.

Run 100 trials, and for each of these 100 trials find the <u>average</u> flow completion time, $\bar{X}_1, \cdots, \bar{X}_{100}$. The longer the simulation run, the lower the variance of $\bar{X}_i$, ie the less "noisy" they are.

Think of $\bar{X}_i$ as Normal $(\mu, \sigma^2(n))$. They cluster around the true mean μ, and the variance $\sigma^2(n)$ decreases as simulation runtime increases.

Naturally, we'd report the average-of-averages, $\bar{\bar{X}} = \frac{1}{m}(\bar{X}_1 + \cdots + \bar{X}_m)$.
But how should we say how confident we are in this value?

- Report a 95% confidence interval, by discarding the top 2.5% of runs and the bottom 2.5%. Then, we're 95% confident that a future run of the same length will produce an average flow completion time in this range.

- If you only have a few runs, it's tricky to "discard 2.5%". Here's an alternative.

  The $\bar{X}_i$ are from different runs, so they're independent.
  Let $\bar{X}_i$ have mean μ and variance $\sigma^2(n)$.
  n is the simulation run length. The longer n, the smaller $\sigma^2(n)$.
  μ is the "true mean", and $\bar{X}_i$ is a noisy sample of it.
  Then $\bar{\bar{X}}$ has mean μ and variance $\frac{1}{m}\sigma^2(n)$, by the rules for mean & var.
  So approximate $\bar{\bar{X}}$ by Normal$(\mu, \frac{1}{m}\sigma^2(n))$.
  We then know

$$P\left(\mu - 1.96\sqrt{\frac{\sigma^2(n)}{m}} \le \bar{\bar{X}} \le \mu + 1.96\sqrt{\frac{\sigma^2(n)}{m}}\right) = 95\%.$$

  Rearranging,

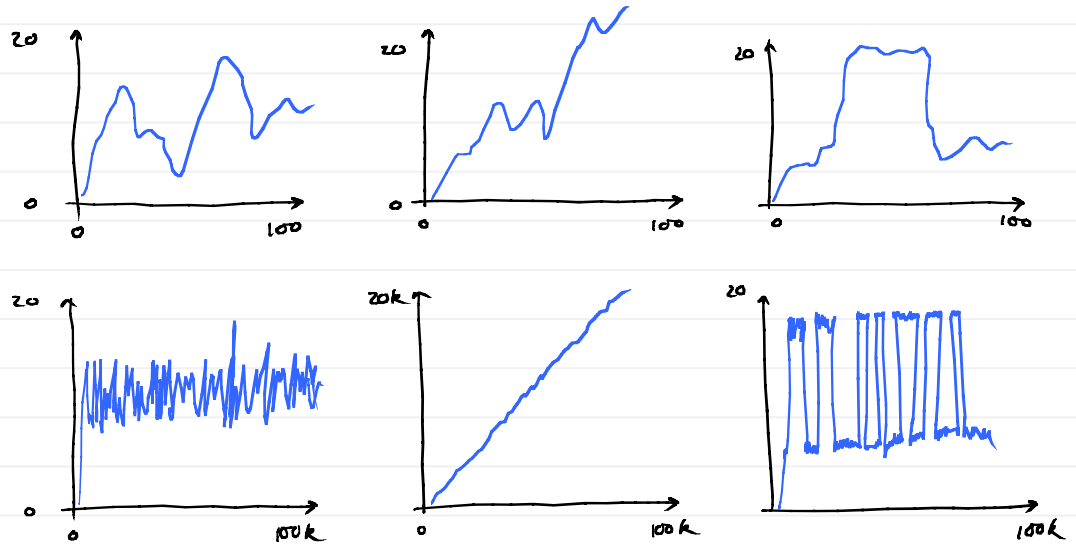
$$P\left(\bar{\bar{X}} - 1.96\sqrt{\frac{\sigma^2(n)}{m}} \le \mu \le \bar{\bar{X}} + 1.96\sqrt{\frac{\sigma^2(n)}{m}}\right) = 95\%$$

  In other words, we have a confidence interval for μ, the true mean.
  (We still need $\sigma^2(n)$. Estimate it by the sample variance, $\sigma^2(n) \approx \frac{1}{m}\sum(\bar{X}_i - \bar{\bar{X}})^2$.)

  This calculation relies on independence. It doesn't apply if the $\bar{X}_i$ come from a single trial split into chunks, since they won't be independent.)

§2.5 Mice and Elephants

We've discussed three types of behaviour we might see in a simulation run: stable, unstable, bistable. For each of these, the system looks more and more smooth/well-behaved over longer and longer intervals.

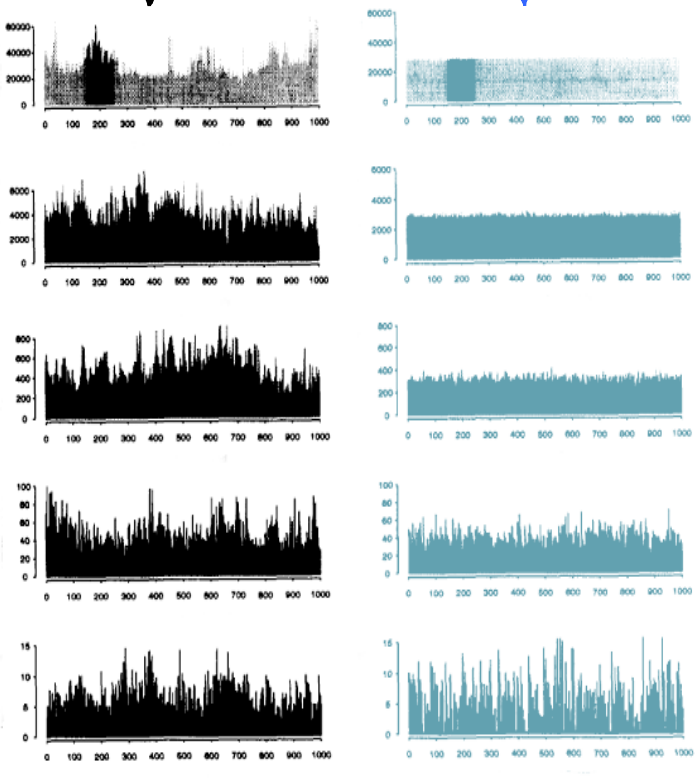


eqm. dist.

growth rate: $\frac{20k}{100k} = 0.2$

eqm dist.

Sometimes, life isn't this well-behaved: Here are plots of packets per time unit, from Ethernet traces and from a simple random traffic generator with the same mean.



time unit = 100 sec
28 hours total

time unit = 10 sec
3 hours total

time unit = 1 sec
17 min total

time unit = 0.1 sec
2 min total

time unit = 0.01 sec
10 sec total

On the Self-Similar Nature of Ethernet Traffic

W.E. Leland, M.S. Taqqu, W. Willinger, D.V. Wilson
(Originally Published in: Proc. SIGCOMM '93, Vol. 23, No. 4, October 1993)

The explanation for this seems to be "heavy-tailed" file sizes: the longer you watch, the more likely you are to see a gigantic file.

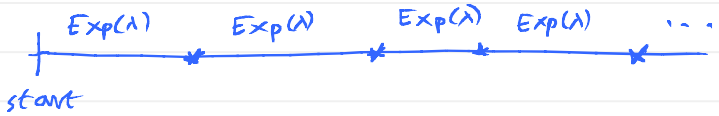
If your simulator has heavy-tailed distributions, you need to run for a very long time to get robust averages (i.e. $\sigma^2(n)$ decreases only very slowly).

If real life has this property, maybe it's meaningless to even ask about steady-state behaviour.

The canonical model for heavy tails is the Pareto distribution. See §1.8c for details.

§ 2.6 Standard Processes

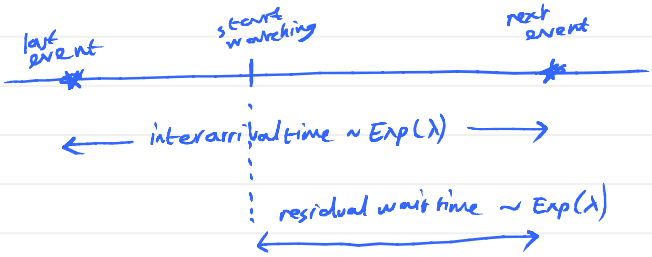
1. POISSON PROCESS



A sequence of events is called a *Poisson process of rate* λ if the times between events are independent Exp(λ) random variables, and so is the time until the first event.

It is appropriate to use a Poisson process to model event times in systems which have a large number of independent agents, each of which may trigger events — call start times on a telephone network — web-browsing sesssion start times (though not page requests) — deaths by mule-kick in Napoleon's army — packet arrivals at a core Internet router with many flows (though not at an edge router) — radioactive particle emissions from a chunk of radioactive matter — See Paxson and Floyd, The failure of Poisson modeling, *for the situations in which a Poisson arrival model is appropriate.*

(i) The Poisson process is memoryless. That is, if I turn up at some arbitrary point in time, the wait until the next event is Exp(λ).



Let $X \sim \text{Exp}(\lambda)$ be the interarrival time, let x_0 be the time after the last arrival when I start watching, and let $Y = X - x_0$ be the residual wait time.

What's the distribution of Y?

$$\mathbb{P}(Y \geq y) = \mathbb{P}(X - x_0 \geq y \mid X \geq x_0) = \mathbb{P}(X \geq x_0 + y \mid X \geq x_0) = \frac{\mathbb{P}(X \geq x_0 + y \text{ and } X \geq x_0)}{\mathbb{P}(X \geq x_0)}$$

$$= \frac{\mathbb{P}(X \geq x_0 + y)}{\mathbb{P}(X \geq x_0)} = \frac{e^{-\lambda(x_0+y)}}{e^{-\lambda x_0}} = e^{-\lambda y}. \quad \text{So } Y \sim \text{Exp}(\lambda).$$

This is called the memoryless property of the Exponential distribution.

(ii) There are on average λ events per time unit. This is why λ is called the *rate* of the process.

The expected time until the next arrival is $\mathbb{E}(\text{Exp}(\lambda)) = \frac{1}{\lambda}$.
So the expected time for n arrivals is n/λ.
The long-term average arrival rate is the big-timescale limit of

$$\frac{\#\text{ arrivals}}{\text{time for those arrivals}} = \frac{n}{n/\lambda} = \lambda.$$

(iii) The rate parameter λ has units sec^{-1}, min^{-1} etc. Rescaling time is equivalent to multiplying λ. For example, a Poisson process of rate 2/sec is the same as a Poisson process of rate 60/min.

This derives from a scaling property of the Exponential distribution:
if $X \sim \text{Exp}(\lambda)$ then $aX \sim \text{Exp}(\lambda/a)$.

(iv) Let N be the number of events in an interval of duration Δ. Then

$$\mathbb{P}(N = r) = \frac{(\lambda\Delta)^r e^{-\lambda\Delta}}{r!}, \qquad \mathbb{E}N = \lambda\Delta \qquad \text{Var}N = \lambda\Delta.$$

This is called the *Poisson distribution*. Do not confuse the Poisson process (the entire collection of all arrival times) with the Poisson distribution (the number of arrivals in a given interval).

This also tells us the arrival rate: $\frac{\mathbb{E}\,\#\text{arrivals over }\Delta}{\Delta} = \frac{\lambda\Delta}{\Delta} = \lambda$.

(v) If flies arrive as a Poisson process of rate λ, and each independently has probability p of landing in my soup, then the arrival process of flies to my soup is a Poisson process of rate λp. This is called *thinning*.

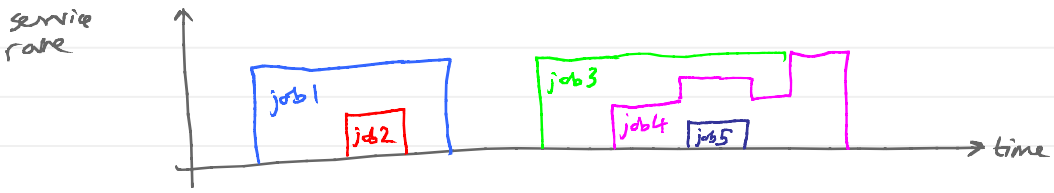
If flies arrive as a Poisson process of rate λ, and wasps arrive as a Poisson process of rate μ, then the arrival process of insects is a Poisson process of rate $\lambda + \mu$. This is called *superposition*.

Superposition relates to the following properties of the exponential distribution:
If $X \sim \text{Exp}(\lambda)$ and $Y \sim \text{Exp}(\mu)$ are independent, then

- $\min(X, Y) \sim \text{Exp}(\lambda+\mu)$
- $\mathbb{P}(X < Y) = \frac{\lambda}{\lambda+\mu}$.

(vi) *Poisson Arrivals See Time Averages* (PASTA). This means that, for a stable system with Poisson arrivals, the time-average state distribution π^T and the user-average state distribution π^U (described in §2.2) are equal.

2. PROCESSOR-SHARING QUEUE



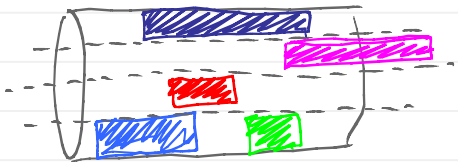
This is a simple model for a single bottleneck link shared by several TCP flows, or for a single CPU shared by several tasks.

Let new jobs arrive at average rate λ jobs/sec, and let the average size of each job be m units of work. The jobs share the resource fairly: when there are n jobs, they each get served at rate C/n, where C is the service rate of the resource measured in units/sec. Once a job's work has been completely served, it departs.

This system is stable if $\rho = \lambda m/C < 1$ and unstable if $\rho > 1$. If arrivals are a Poisson process of rate λ and job sizes are independent and $\rho < 1$, then the equilibrium distribution is

$$\mathbb{P}(\text{num active jobs} = r) = (1-\rho)\rho^r.$$

3. ERLANG LINK



This is a simple model for a link in a circuit-switched telephone network, i.e. a network in which each call takes up a fixed amount of capacity and lasts for a given duration regardless of the other calls in the system.

Let new jobs arrive at average rate λ jobs/sec, and let the average job duration be m sec. If a job arrives to find that C jobs are already present, then this job is *blocked* i.e. it is not admitted to the system. Otherwise it is admitted, and it remains in the system for its full duration.

This system is always stable. If the arrivals are a Poisson process of rate λ, then the equilibrium distribution is

$$\mathbb{P}(\text{num active jobs} = r) = \frac{\rho^r/r!}{\sum_{i=0}^{C}\rho^i/i!}, \qquad \text{where } \rho = \lambda m.$$

By the PASTA property, the probability that an incoming job is blocked is equal to the equilibrium probability that there are C jobs active, and this is

$$\text{blocking prob} = E(\rho, C) = \frac{\rho^C/C!}{\sum_{i=0}^{C}\rho^i/i!}.$$

If you try to compute $E(\rho,C)$ naively, you'll run into problems with numerical overflow. To avoid this, note that

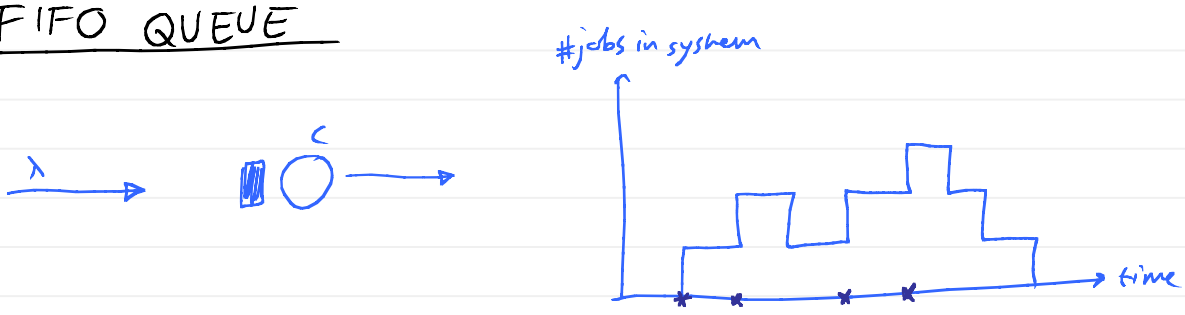
$$E(\rho,C) = \frac{\rho^C/C!}{\sum_{i=0}^{C}\rho^i/i!} = \frac{e^{-\rho}\rho^C/C!}{\sum_{i=0}^{C}e^{-\rho}\rho^i/i!} = \frac{\mathbb{P}(X=C)}{\mathbb{P}(X\le C)}, \quad \textit{where } X \sim \text{Poisson}(\rho).$$

Typo

You can probably find library functions for $\mathbb{P}(X=C)$ and $\mathbb{P}(X\le C)$. In Python,

```
import scipy.special
# pdtr(k,λ) gives Prob(X≤k) where X~Poisson(λ)
def dpois(ρ,C): return scipy.special.pdtr(C,ρ)-scipy.special.pdtr(C-1,ρ)
def erlang(ρ,C): return dpois(ρ,C)/scipy.special.pdtr(C,ρ)
```

4. FIFO QUEUE



Consider a classic first-in–first-out queue, with an infinite buffer. Suppose that jobs arrive as a Poisson process of rate λ and that job service times are Exponential random variables with mean m. This is stable if $\rho = \lambda m < 1$. If so, the equilibrium distribution of the number of jobs in the system (including the one being served at the head of the queue) is

$$\mathbb{P}(r \text{ jobs in system}) = (1-\rho)\rho^r.$$

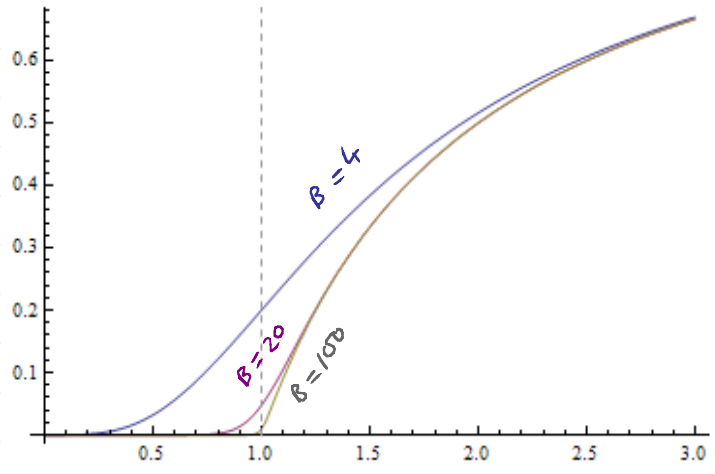
Now suppose instead the queue has a finite buffer of size B, and that when it is full then incoming jobs are dropped. The equilibrium distribution is

$$\mathbb{P}(r \text{ jobs in system}) = \frac{(1-\rho)\rho^r}{1-\rho^{B+1}}.$$

By the PASTA property, the drop probability is

$$\mathbb{P}(\text{job dropped}) = \frac{(1-\rho)\rho^B}{1-\rho^{B+1}}.$$

These equilibrium formulae only apply to Poisson arrivals and Exponential job sizes. In some other systems, the formulae apply to Poisson arrivals and any distribution of job sizes.

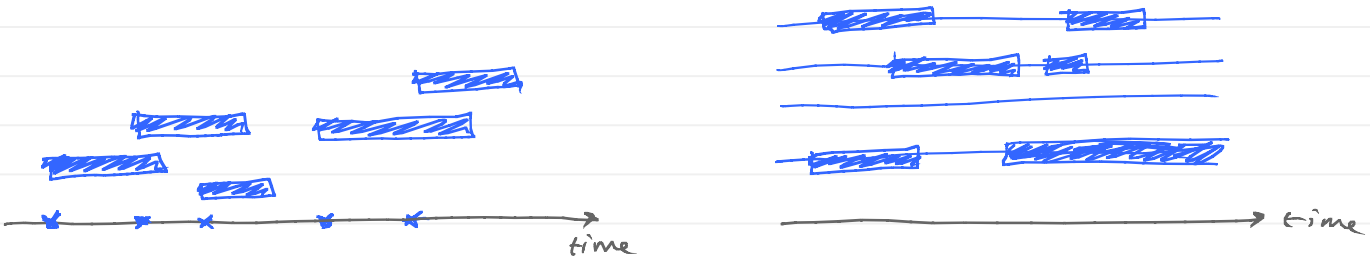


As $B \to \infty$,

$$\mathbb{P}(\text{job dropped}) \to \begin{cases} 0 & \text{if } \rho \le 1 \\ \frac{\rho-1}{\rho} & \text{if } \rho \ge 1 \end{cases}$$

This is written more concisely as $\left(\frac{\rho-1}{\rho}\right)^+$.

5. OPEN AND CLOSED WORKLOAD MODELS



Here are two simple models for demand, in a system with unlimited capacity. It's useful to have an idea of how much total demand there might be, so you can do a quick rule-of-thumb estimate of how much capacity you'd need to give near-perfect quality of service.

- *Open workload model.* Let jobs arrive as a Poisson process of rate λ jobs/sec and have mean duration m sec. The equilibrium distribution of the number of active jobs is Poisson with mean λm.
- *Closed workload model.* Let there be a fixed population of n users. Suppose each user alternates between idle and active, and that the mean active time is b and the mean idle time is i, and that users are independent. The equilibrium distribution of the number of active users if $\mathrm{Bin}\big(n, b/(i+b)\big)$.

More generally, an open model is one in which there is an infinite population of possible jobs and job arrivals are independent of the state of the system; all the models in the preceding sections have been open models. Open models are appropriate for e.g. telephone calls, or packets at a large Internet router.

A closed model is one in which there is a finite population of jobs, and they move between different states. We will analyse these in §3. Closed models are appropriate for e.g. a pool of worker threads at a web server, or a local network with a small number of hosts.