§0 Introduction

This course studies the question:
"How can a distributed collection of entities cooperate to achieve some goal?"

The most interesting application of this question, for us, is
"How can a collection of devices co-ordinate themselves, so as to achieve a fair sharing of network resources?" We are particularly interested in algorithms for achieving coordination with scant communication between devices.

Example. In a wifi network, we want one device to send and all the others to be quiet, so the transmission gets through. But we don't want to waste time or bandwidth trying to work out whose turn it is to send.

Example. In a wired network, suppose a link has capacity 10 Mb/s, and it is shared between 5 devices. It seems reasonable for each to send at 2 Mb/s: any more and the excess will simply be dropped, any less and the capacity is wasted. How can they collectively achieve this division? And what if it is a network of links we are trying to share?

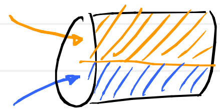
Illustration. The Coordination Game

The game has 3 rounds.
In each round you may raise your hand
- if exactly one player raises his/her hand, he/she gets £10, everyone else gets £1, and the game ends
- otherwise, I say how many hands were raised, and we move on to the next round.

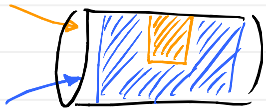
Players must keep quiet and keep their eyes shut.

If there was one person in charge, it would be easy to ensure the money is won, and then to share it out fairly. But in a distributed setting, where each player is autonomous, it is much harder to achieve co-ordination.

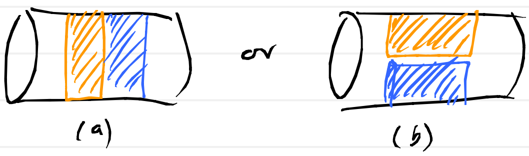
Even if the entities had perfect knowledge of what each other was doing, and they could co-operate to achieve the desired sharing of network resources, we are left with the question: what is the right way to share? Here are some examples to think about.

(1) 

A single link with two flows. It seems reasonable to share the bandwidth fairly.

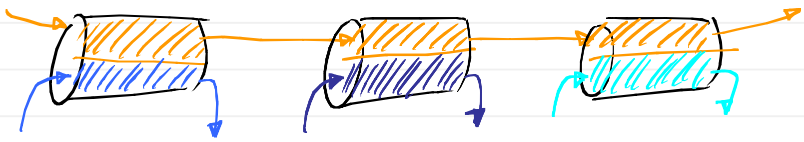
(2) 

A single link with one short-lived flow and one long-lived flow. While they are both active, would it be fairer to prioritize the shorter flow?

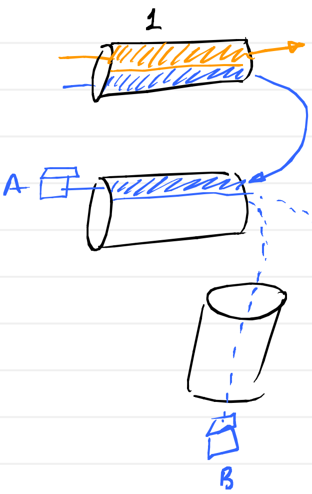
(3) 

Two equal-sized flows which arrive at the same time.

In (a) and (b), the blue flow finishes at the same time. In (a), the orange flow finishes sooner. Surely, on balance, (a) is better? We could randomize priority, to make it fair.

(4) 

Is it still fair to split the bandwidth on each link equally? In a sense, the orange flow is consuming 3× more resources than any of the other flows.

(5) 

Is it still fair to split the capacity of link 1 evenly, if the blue flow is for a P2P user A, who could just as well have chosen a less congested peer B or C?

Comcast is a large cable ISP in the US. Over the past two years it has been fighting a battle with the regulator, the FCC.

October 2007. Comcast (a large US cable ISP) was found to be blocking BitTorrent traffic

http://arstechnica.com/news.ars/post/20071019-evidence-mounts-that-comcast-is-targeting-bittorrent-

January 2008. FCC (the US telecoms regulator) opens proceedings. Comcast protests it is merely applying reasonable network management, for the good of its customers.

http://arstechnica.com/news.ars/post/20080114-fcc-officially-opens-proceeding-on-comcasts-p2p-

August 2008. FCC sanctions Comcast for "secretly degrading peer-to-peer applications". It upholds its 2005 "Internet Policy Statement":

* Consumers are entitled to access the lawful Internet content of their choice

* Consumers are entitled to run applications and use services of their choice, subject to the needs of law enforcement

* Consumers are entitled to connect their choice of legal devices that do not harm the network

* Consumers are entitled to competition among network providers, application and service providers, and content providers

http://arstechnica.com/news.ars/post/20080725-hammer-drops-at-last-fcc-opposes-comcast-p2p-throttling.html

September 2008. Comcast says it will sue FCC. In the meantime, it is developing "protocol-agnostic" throttling systems.

http://arstechnica.com/news.ars/post/20080904-comcast-sues-fcc-wants-p2p-throttling-order-

http://arstechnica.com/news.ars/post/20080919-comcast-loses-p2p-religion-goes-agnostic-on-throttling.html

August 2009. Comcast tells Appeals court: "The FCC had no authority to sanction us"

September 2009. FCC tells Appeals court: "We have authority from Congress"

http://arstechnica.com/tech-policy/news/2009/08/fcc-enforcing-imaginary-laws-in-p2p-ruling-says-comcast.ars

http://arstechnica.com/tech-policy/news/2009/09/fcc-congress-said-we-could-spank-comcast-for-p2p-blocking.ars

COMCAST'S CONUNDRUM

Why is this fight happening?

- In Comcast's network, there isn't much bandwidth available for upstream communication. This is because, at the time the hardware was put in place, no one envisaged P2P etc. It was assumed that users would only upload short objects like http requests.

- This means that simple profitable occassional-surfers get squeezed out by P2P uploaders (see Fairness example 2 above).

- Indeed, the MAC layer (called DOCSIS) tends to give priority to existing flows (see Fairness example 3b above).

- And anyway, given that cable networks tend to have congested upstream pipes, whereas ADSL networks tend to have congested downstream pipes, isn't it reasonable for the ADSL users to do all the P2P uploading? (see Fairness example 5 above).
  — (except that TCP has a bias against long-distance connections — see later in the course.)

AIMS OF THIS COURSE

We cannot have a reasoned debate about e.g. Comcast, the FCC, and network neutrality, unless we understand how the performance of the network depends on the algorithms/mechanisms that are put in place.

Remember, there are very hard questions to answer about:
- what outcome should we be aiming for?
- how can we get a distributed collection of devices to co-ordinate among themselves, so as to achieve the desired outcome?

- You will learn the mathematical models and methods for describing the behaviour of network systems
  - MAC level (packets)
  - TCP (rates)
  - Job level (flows that come and go)

- You will see different styles of modelling
  - rules of thumb & simple formulae
  - simplified models, where you can solve the algebra by hand
  - complex models, which a computer can solve rapidly
  - simple simulations, with details cut out
  - complex simulations } not covered in this course.
  - experiments on test-beds }
  - measurement of real-world data

  The "best" model is the one that gives you the most useful answers with the least amount of work. This is generally <u>not</u> the most detailed model. The best analyses draw on all the levels of model listed above — use the more detailed model to check the assumptions of the simpler model; use the simpler model to guide you to interesting scenarios.

- You will also learn some fundamental general-purpose tools for modelling systems and reasoning about data.

§1 Random numbers

§1.1 Introduction

Example use: I've developed a new load-balancing algorithm for a web server. I want to test my algorithm, by means of simulation. My simulator needs a random number generator, to generate file sizes, request times etc. The performance of my algorithm will probably depend on the random number generator I use. How should I program this random number generator?

We would use random number generators in situations like this because the world is too complicated for us to model it in a Newtonian cause-and-effect system. Even though there might be deterministic explanations for every little variation in file size etc., it's too hard to take account of them all. Instead, we use random numbers to say "There is variability, and I can quantify the degree of variability, but I'm not going to look in excruciating detail for causes for every little variation."

Typically, we take real-world measurements, we look at the data, and we try to program a random number generator that produces output consistent with the data.

Perhaps the real-world measurements show a range of behaviours, e.g. web requests arrive close together at peak times, far apart at off-peak. How should I make my random number generator tunable, to capture this range?

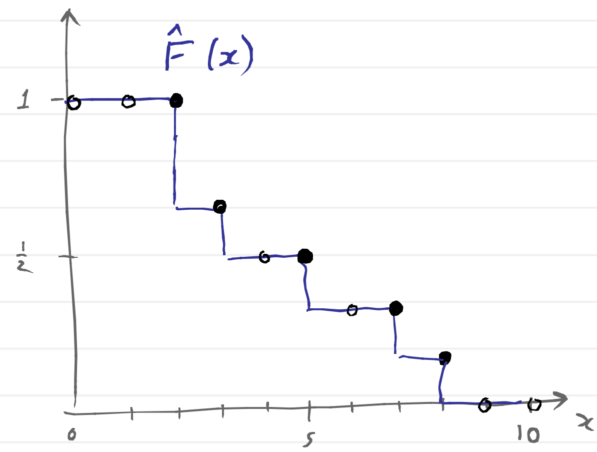
Many standard random number generators come with tunable parameters. Typically, we look at the real-world measurements and try to estimate what values of the tuning parameters give the best fit.

Then, we can use the simulator to ask: how does the performance of my algorithm depend on these parameters? In some cases we don't even need to run the simulator — we can use maths to calculate the performance.

<u>Example</u> I take real-world measurements of some quantity, and get readings 3, 8, 7, 2, 2, 5. A good way to illustrate this is by plotting the <u>empirical tail distribution function</u>

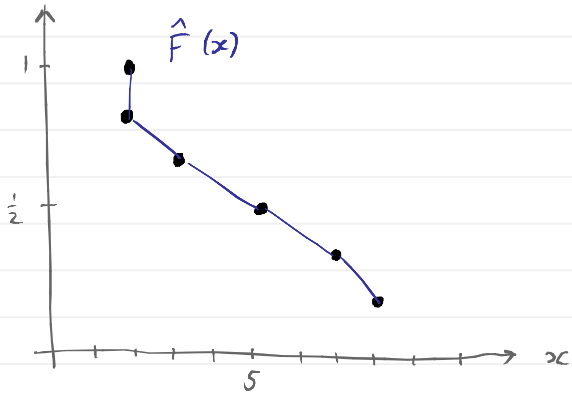
$$\hat{F}(x) = \frac{\text{number of readings that are} \geq x}{\text{total \# of readings}}.$$

x	# of readings that are $\geq x$	$\hat{F}(x)$
0	6	1
1	6	1
2	6	1
3	4	4/6
4	3	3/6
5	3	3/6
6	2	2/6
7	2	2/6
8	1	1/6
9	0	0



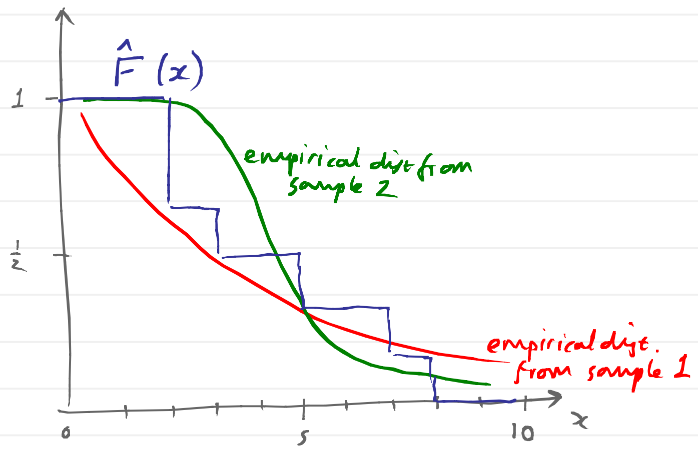
The only "interesting" points on this graph are the points at which it steps down. In practice, it's more convenient to only tabulate & plot those points. Also, if your plotting system doesn't do "step-style" curves, it's fine to simply connect the points. If you have a large enough sample (many more than 6 values) the difference will be negligible.

x	# of readings that are $\geq x$	$\hat{F}(x)$
2	6	1
2	5	5/6
3	4	4/6
5	3	3/6
7	2	2/6
8	1	1/6



This row is a bit of a cheat. Really, it should be $\hat{F}(2) = 1$. But by using these values, (a) it's easier to generate the values in this table, and (b) I get a "step down" at 2, like the graph at the top of the page.

Suppose I have two candidate random number generators. I could generate samples from each, e.g. by calling each 1000 times, and I could plot the resulting empirical distributions.



I'd then pick whichever of the two seems to be a better fit. In this case it's hard to tell — I need to do more measurements to get a more detailed picture for $\hat{F}(x)$.

Later in the course, we will learn about "true" theoretical distribution functions. Typically you know exactly what the distribution function for a given random number generator should look like, so you can plot it directly rather than generating 1000 or so samples.

§1.2 Describing random variables

We use the term <u>random variable</u> to refer to the output of a random number generator. There may be two different pieces of code, that produce indistinguishable output. (Of course the output is random, so the two may be indistinguishable but not identical.) In this case we'd say that there are two different random number generators, but that the random variables that they output are the same.

Do not confuse random variables with non-random. It's a good habit to read through your equations and ask: which values will be different each time I run the simulation/experiment? These are the random variables.

The most basic way to describe a random variable X is by its <u>tail distribution function</u>

$$F(x) = \mathbb{P}(X \geq x) =$$ probability that when you run the random number generator, you get an output X that is $\geq x$.

Suppose we call the random number generator n times, and get outputs $X_1, \dots, X_n$ (also called <u>samples</u>), and let N be the number of these that are $\geq x$. Of course N itself is random. But we expect

$$\frac{N}{n} \approx F(x) \quad , \quad \text{or equivalently} \quad \hat{F}(x) \approx F(x).$$

Note: here, X represents the output of a random number generator, ie a value which is different each time you run your simulator or experiment. And x is an arbitrary non-random value. I will generally use uppercase for random variables, lower-case for non-random values — but not always, e.g. F is just a function, not random.

We expect that the approximation $\hat{F}(x) \approx F(x)$ should get more accurate as the number of samples $n \to \infty$. In a way, random variables are an "idealized version" of the output of a random number generator

FURTHER DESCRIPTIONS OF RANDOM VARIABLES

There are two important classes of real-valued random variables:

discrete random vars, and **continuous** random vars

the set of possible outcomes is finite or countable, e.g. the number of 134 buses to pass Gower St today (possible outcomes = integers)

the set of possible outcomes is an interval of real numbers, e.g. time until the next 134 bus (possible outcomes = positive real numbers)

Defn If X is continuous, define the **density** to be $f(x) = -\frac{d}{dx} F(x)$, which means $F(x) = \mathbb{P}(X \geq x) = \int_x^\infty f(y)\,dy$, and $\mathbb{P}(X \in [a,b]) = \int_a^b f(y)\,dy$.

Note that $\mathbb{P}(X \in \mathbb{R}) = 1$, so $\int_{-\infty}^{\infty} f(y)\,dy = 1$ necessarily.

If X is discrete, define the **density** to be $\pi_x = \mathbb{P}(X = x)$, which means $F(x) = \mathbb{P}(X \geq x) = \sum_{y \in \Omega : y \geq x} \pi_y$ where Ω is the set of possible outcomes.

Note that $\mathbb{P}(X \in \Omega) = 1$, so $\sum_{y \in \Omega} \pi_y = 1$ necessarily.

Sometimes, people use the words density, distribution, distribution function, interchangeably. You have to read the context carefully to work out which is meant. Also, "probability mass function" is a synonym for density. Another common term is "cumulative distribution function" which refers to $\mathbb{P}(X < x)$ i.e. to $1 - F(x)$.

§ 1.3 Common Distributions

There are a few distributions for random variables that crop up again and again in nature. See handout for details of these.

CONTINUOUS

Exponential distribution — used to model the time until an event happens for many natural processes
(Telnet session initiations, telephone call initiations, light bulb blows, radioactive nucleus decays)

Pareto distribution — time until an event happens in certain "cascade" processes, where one event can trigger others
(FTP transfer starts, landslide)
— size of a "cascade" event
(TCP flow size, insurance claim, terrorist attack)

Normal distribution — the size of a "natural" observation which doesn't deviate too much, (ie doesn't vary by many orders of magnitude) especially observations of the accumulation of many small factors
(height, weight, IQ)

DISCRETE

Geometric distribution — Like Exponential but in discrete time, i.e. the number of clock ticks until an event happens
(weeks until I win the lottery, packets sent until one is dropped)

Binomial/multinomial — The outcome of classifying n observations into categories
(e.g. number of heads in 100 tosses of a coin)
(e.g. survey 85 people, classify them by

	male	female
cute		
ugly		

)

Poisson — The number of events that occur in a fixed observation window, for well-behaved "natural" systems
(e.g. number of deaths by mule-kick each year in Napoleon's army, number of telnet sessions per hour)

Zipf — If e.g. city sizes have a Pareto distribution, and we pick a person at random, the rank of his/her city (1st, 2nd biggest, 3rd ...) has a Zipf distribution.

Example: The Exponential distribution

For network applications, the Exponential distribution is the most important random variable.

Let $X \sim \mathrm{Exp}(\lambda)$, ie let X have an exponential distribution with parameter λ. The handout tells us that the set of possible outcomes (the range) is $\mathbb{R}^+$, ie any positive real number is possible — so it's a continuous rand. var.

The density function is $f(x) = \lambda e^{-\lambda x}$, according to the handout.

The distribution function is $F(x) = \mathbb{P}(X \geq x) = \int_x^\infty \lambda e^{-\lambda y}\,dy = \left[-e^{-\lambda y}\right]_x^\infty = e^{-\lambda x}$.

Example: The Geometric distribution

Let $X \sim \mathrm{Geom}(p)$, ie let X have a geometric distribution with parameter p. The handout tells us that the set of possible outcomes is $\{1, 2, \cdots\}$ which is countable — so it's a discrete random variable.

The density is $\mathbb{P}(X=r) = (1-p)^{r-1} p$.

The distribution function is $F(r) = \mathbb{P}(X \geq r) = \sum_{s=r}^{\infty} (1-p)^{s-1} p = (1-p)^{r-1}$.

Interpretation. I toss a biased coin (probability p of getting Heads) repeatedly. Let X be the number of tosses until my first Heads.

$\mathbb{P}(X=r) = \mathbb{P}($first $r-1$ tosses were tails, then next is heads$) = (1-p)^{r-1} p$.

$\mathbb{P}(X \geq r) = \mathbb{P}($first $r-1$ tosses were tails$) = (1-p)^{r-1}$.

Thus, $X \sim \mathrm{Geom}(p)$.

§1.4 Generating random variables

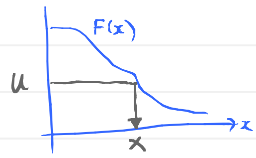
Most languages come with a built-in function for generating a Uniform [0,1] random variable, eg random.random() in Python. We can use this to generate other random variables.

Exercise Look up the "mother of all RNGs" by Marsaglia, sci.stat.consult, 1994. Implement it.

THE INVERSION METHOD

Suppose we want to generate a r.v. X with $P(X \geq x) = F(x)$.

1. Generate $U \sim$ Uniform $[0,1]$
2. Find X such that $F(X) = U$.
3. Use this value X as the r.v. we want.

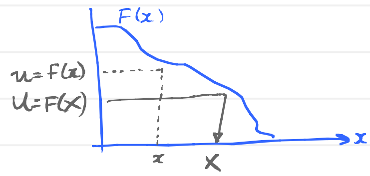


Example Suppose we want to generate $X \sim \mathrm{Exp}(\lambda)$, with $P(X \geq x) = e^{-\lambda x}$.

1. Generate $U \sim$ Uniform $[0,1]$
2. Solve $e^{-\lambda X} = U \Rightarrow -\lambda X = \log U \Rightarrow X = -\frac{1}{\lambda} \log U$.
3. Use $X = -\frac{1}{\lambda} \log U$ as our $\mathrm{Exp}(\lambda)$ random variable.

THEORY BEHIND THE METHOD:

$$P(X \geq x) = P(U \leq F(x)) = F(x).$$



THE BOX-MULLER ALGORITHM [Not examinable]

There are many cases where step 2 of the inversion method is too hard. One such case is the Normal distribution. Luckily, there is a simple alternative.

Let U, V be independent Uniform $[0,1]$ random variables.

Let
$$X = \sqrt{-2 \log U} \cos(2\pi V)$$
$$Y = \sqrt{-2 \log U} \sin(2\pi V)$$

Then X, Y are independent Normal(0,1) random variables.

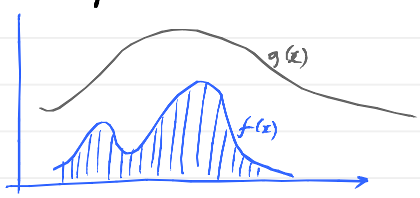
THE ACCEPTANCE-REJECTION METHOD [Not examinable]

This method always works, but it can be inefficient.

Suppose we want to generate a r.v. X with density $f(x)$.

Suppose we are able to generate a r.v. Y with density $g(x)$, where $f(x) \leq \alpha g(x)$ for all x,

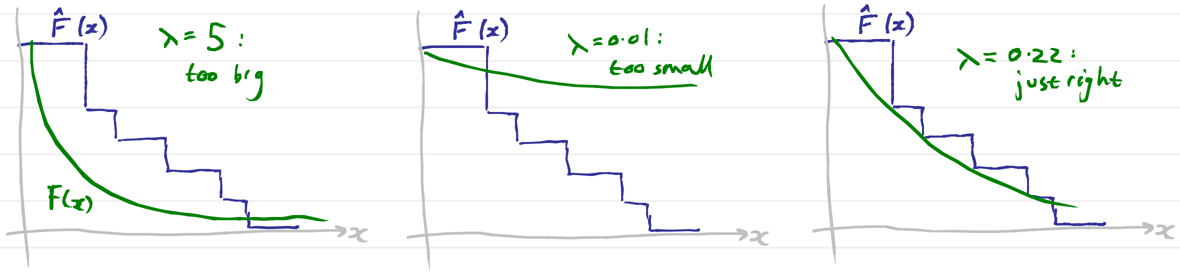
1. Generate a r.v. Y with density g.
2. Generate a r.v. $U \sim$ Uniform $[0,1]$
3. If $U \leq \dfrac{f(Y)}{\alpha g(Y)}$ then output Y; otherwise go back to step 1.



§1.5 Fitting Distributions

Suppose we have a collection of real-world measurements 3, 8, 7, 2, 2, 5.
Suppose I want a random number generator to mimic these outputs,
and suppose I've settled on using an Exponential random variable.
This distribution depends on a parameter λ, and I need to pick a suitable λ.

I could plot the empirical distribution function $\hat{F}(x) = \frac{1}{6}(\#\text{ of measurements that are} \geq x)$ and the theoretical distribution function $F(x) = \mathbb{P}(X \geq x) = e^{-\lambda x}$ for the Exponential distribution, and I could tweak λ until the two curves match up.



There is a systematic procedure for doing this:

1. Write down the density function $f_p(x)$ of the random variable whose parameters you want to fit. The density function depends on those parameters (there may be more than one). Here I've written p to denote "parameters to fit".

2. Write out $lik(p) = f_p(x_1) \times f_p(x_2) \times \cdots \times f_p(x_n) = \prod_{i=1}^{n} f_p(x_i)$.

3. Find the value of p that maximizes $lik(p)$, call it $\hat{p}$.
Often it's easier to maximize $\log(lik(p))$. This must give the same value of $\hat{p}$.

4. This value $\hat{p}$ is the maximum likelihood estimator of p.
It gives the best-fitting distribution.

Example I observe bus inter-arrival times of 2 min, 10 min, 3 min, 8 min, 7 min.
I suspect these are independent $\sim Exp(\lambda)$ random variables, and I want to estimate λ.

1. The density function of $Exp(\lambda)$ is $f_\lambda(x) = \lambda e^{-\lambda x}$.

2. The likelihood function is

$$lik(\lambda) = (\lambda e^{-\lambda \times 2})(\lambda e^{-\lambda \times 10})(\lambda e^{-\lambda \times 3})(\lambda e^{-\lambda \times 8})(\lambda e^{-\lambda \times 7})$$

$$= \lambda^5 e^{-30\lambda}$$

3. We'll choose λ to maximise $\log lik(\lambda) = 5\log\lambda - 30\lambda$:

$$\frac{d}{d\lambda} \log lik(\lambda) = 0 \quad \Rightarrow \quad \frac{5}{\lambda} - 30 = 0 \quad \Rightarrow \quad \lambda = \frac{5}{30} = \frac{1}{6}.$$

4. Our maximum likelihood estimator is $\hat{\lambda} = \frac{1}{6}$
Note the units: λ must have units $[\frac{1}{min}]$ since the density function is $\lambda e^{-\lambda x}$ and x is measured in [min] and you're only allowed to take exponentials of a unitless quantity. So really $\hat{\lambda} = \frac{1}{6}$ / min.

Exercise Read "Empirical study of OS errors".
This is a very good paper about fitting distributions to data.
- What quantities are being described by means of random variables?
- How are the distributions illustrated?

§1.6 Working with distribution functions

For some calculations, we have to work directly from the distribution function. or density function. For others, there are short-cuts — see §1.8

Here are some useful summaries of the distribution function: describing the average and variability of a random variable.

Defn The "average"

The *expected* or *mean* value of X is $\mathbb{E}X = \begin{cases} \sum_{x\in\Omega} x\,\mathbb{P}(X=x) & \text{when } X \text{ is discrete} \\ \int_\Omega x f(x)\,dx & \text{when } X \text{ is continuous.} \end{cases}$

The "typical value"

The *median* of X is x such that $\mathbb{P}(X \geq x) = \mathbb{P}(X < x) = \frac{1}{2}$.

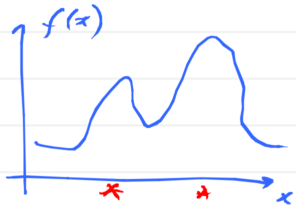
NOTE. For discrete random variables, it may not be possible to find such an x exactly. In that case, use the closest x you can.

The "most likely value"

The *mode* of X is a local maximum on the density graph

There may be more than one mode

a unimodal rand. var.



a bimodal rand. var.

The "variability"

The *variance* of X is $\mathrm{Var}X = \mathbb{E}\left[(X-\mu)^2\right] = \begin{cases} \sum_{x\in\Omega} (x-\mu)^2\,\mathbb{P}(X=x) & \text{when } X \text{ is discrete} \\ \int_\Omega (x-\mu)^2 f(x)\,dx & \text{when } X \text{ is continuous} \end{cases}$

where $\mu = \mathbb{E}X$.

The *standard deviation* of X is $\mathrm{sd}(X) = \sqrt{\mathrm{Var}X}$.

Likely ranges

The *first quartile* is a number x such that $\mathbb{P}(X \leq x) = 25\%$

median —— $\mathbb{P}(X \leq x) = 50\%$

third quartile —— $\mathbb{P}(X \leq x) = 75\%$

p-percentile —— $\mathbb{P}(X \leq x) = p$

The range $[x_1, x_2]$ is a 95% confidence interval if $\mathbb{P}(x_1 \leq X \leq x_2) = 95\%$.
Often we choose a two-sided confidence interval with $\mathbb{P}(X < x_1) = \mathbb{P}(X > x_2) = 2.5\%$.
In other contexts it may be useful to report a one-sided confidence interval — either an upper confidence interval $[x_1, \infty)$ or a lower confidence interval $(\infty, x_2]$
Confidence intervals are a way to express the variability of a random variable, rather like the standard deviation — but std. dev is often easier to calculate with.

§ 1·6a : The Exponential Distribution

The random variable reference sheet tells us that the Exponential distribution is for a real-valued random variable taking values in $[0,\infty)$. It has one parameter, $\lambda > 0$. Its density is

$$f(x) = \lambda e^{-\lambda x}.$$

Distribution function: $\mathbb{P}(X \geq x) = \int_x^\infty \lambda e^{-\lambda y}\, dy = e^{-\lambda x}$

Mean: $\mathbb{E}X = \int_0^\infty x \cdot \lambda e^{-\lambda x}\, dx = \left[-xe^{-\lambda x}\right]_0^\infty - \int_0^\infty -e^{-\lambda x}\, dx = \int_0^\infty e^{-\lambda x}\, dx = \left[-\frac{1}{\lambda}e^{-\lambda x}\right]_0^\infty = \frac{1}{\lambda}.$

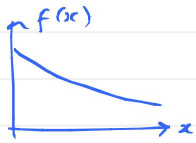
Median: $\mathbb{P}(X \leq x) = \frac{1}{2} \Rightarrow 1 - e^{-\lambda x} = \frac{1}{2}$

$\Rightarrow e^{-\lambda x} = \frac{1}{2}$

$\Rightarrow -\lambda x = \log \frac{1}{2}$

$\Rightarrow x = \frac{1}{\lambda}\log 2.$

Mode:



mode is 0.

Variance: $\mathrm{Var}\, X = \mathbb{E}\left(X - \frac{1}{\lambda}\right)^2 = \int_0^\infty \left(x - \frac{1}{\lambda}\right)^2 \cdot \lambda e^{-\lambda x}\, dx = \cdots = \frac{1}{\lambda^2}$

Std. dev: $\mathrm{sd}(X) = \sqrt{\mathrm{Var}\, X} = \frac{1}{\lambda}.$

Two-sided 95% confidence interval:

$\mathbb{P}(X < x) = 0.025 \Rightarrow 1 - e^{-\lambda x} = 0.025 \Rightarrow e^{-\lambda x} = 0.975 \Rightarrow x = -\frac{1}{\lambda}\log 0.975$

$\mathbb{P}(X \geq x) = 0.025 \Rightarrow x = -\frac{1}{\lambda}\log 0.025.$

So $\left[-\frac{1}{\lambda}\log 0.975, -\frac{1}{\lambda}\log 0.025\right]$ is a 95% confidence interval

Example: let $X \sim \mathrm{Exp}(\lambda)$, and let $Y = aX$ for some constant $a > 0$. What is the distribution of Y?

We will work out the distribution function.

$\mathbb{P}(Y \geq y) = \mathbb{P}(aX \geq y)$

$= \mathbb{P}(X \geq y/a)$ — try to turn it into a statement about sthg we know

$= e^{-\lambda(y/a)}$ — we know the distribution function of X

$= e^{-(\lambda/a)y}$ — emphasize that y is the variable we're interested in

So $Y \sim \mathrm{Exp}(\lambda/a)$ — recognize the distribution function, and name it.

Exercise See examplesheet 2 questions 3, 4, 5 for further exercises in the same vein.

§1.6b The Normal Distribution

The Normal distribution is a very popular choice for data analysis because

- it's often a good fit for the aggregate of small quantities (eg individual flowrates)
- it is very simple to do algebra with it

ALGEBRA OF THE NORMAL DISTRIBUTION

If $X \sim \text{Normal}(\mu, \sigma^2)$ then

- density is $f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$. There is no formula for the distribution function.
- the mean is $\mathbb{E}X = \mu$
- the variance is $\operatorname{Var} X = \sigma^2$.
- $aX + b \sim \text{Normal}(a\mu + b, a^2\sigma^2)$ $(*)$
- $(X-\mu)/\sigma \sim \text{Normal}(0,1)$

If $X \sim \text{Normal}(\mu, \sigma^2)$ and $Y \sim \text{Normal}(\nu, \rho^2)$ and X and Y are independent, then

- $X + Y \sim \text{Normal}(\mu + \nu, \sigma^2 + \rho^2)$

To generate a $\text{Normal}(0,1)$ random variable, use the Box-Muller method (§1.4).
To generate a $\text{Normal}(\mu, \sigma^2)$ random variable, first generate $X \sim \text{Normal}(0,1)$ then let $Y = \mu + \sigma X$: by $(*)$, $Y \sim \text{Normal}(\mu, \sigma^2)$.

THE NORMAL DISTRIBUTION AS AN APPROXIMATION

Also referred to as "The central limit theorem".

If $X_1, X_2, \cdots, X_n$ are independent random variables with the same distribution, from (nearly)* any distribution at all, and we let

$$Y = X_1 + \cdots + X_n$$

$$\mu = \mathbb{E}Y = n\,\mathbb{E}X_1, \quad \sigma^2 = \operatorname{Var} Y = n \operatorname{Var} X_1$$

then a good approximation is

$$Y \sim \text{Normal}(\mu, \sigma^2).$$

* assuming that μ and σ^2 are not infinite, plus some minor technical conditions

The more conventional way to write this is

$$\frac{Y - n\,\mathbb{E}X_1}{\sqrt{n}\ \operatorname{sd}(X_1)} \sim \text{Normal}(0,1).$$

In particular, one can prove that the distribution function of $\frac{Y - n\mathbb{E}X_1}{\sqrt{n}\ \operatorname{sd}(X_1)}$ approaches that of $\text{Normal}(0,1)$ as n increases.

APPROXIMATE CONFIDENCE INTERVALS

A standard fact is

$$\mathbb{P}(-1.96 \le \text{Normal}(0,1) \le 1.96) \approx 0.95$$

ie when we generate a Normal(0,1) random variable, we are 95% certain that the generated value lies in the range $[-1.96, 1.96]$

(You can use a computer to find the appropriate ranges for other levels of certainty).

Here is an example of how to use this to find an approximate confidence interval for another random variable.

e.g. I throw a dice 100 times and compute the total score, Y. What is the typical range of values I get for Y?

Let X be the outcome of a single throw.

$$\mathbb{E}X = \tfrac{1}{6}\times 1 + \tfrac{1}{6}\times 2 + \cdots + \tfrac{1}{6}\times 6 = 7/2$$

$$\text{Var}\,X = \tfrac{1}{6}\times(1-\tfrac{7}{2})^2 + \tfrac{1}{6}\times(2-\tfrac{7}{2})^2 + \cdots + \tfrac{1}{6}\times(6-\tfrac{7}{2})^2 = \tfrac{35}{12}$$

By the Normal approximation, Y is approx. $\text{Normal}(100\times\tfrac{7}{2},\ 100\times\tfrac{35}{12})$.

$\Rightarrow$ $Y - 100\times\tfrac{7}{2}$ is approx. $\text{Normal}(0,\ 100\times\tfrac{35}{12})$

$\Rightarrow$ $\dfrac{Y-100\times\frac{7}{2}}{\sqrt{100\times\frac{35}{12}}}$ is approx. $\text{Normal}(0,1)$

$$\Rightarrow \quad \mathbb{P}\left(-1.96 \le \frac{Y-100\times\frac{7}{2}}{\sqrt{100\times\frac{35}{12}}} \le 1.96\right) \approx 0.95$$

$$\Rightarrow \quad \mathbb{P}\left(-1.96\sqrt{100\times\tfrac{35}{12}} \le Y - 100\times\tfrac{7}{2} \le 1.96\times\sqrt{100\times\tfrac{35}{12}}\right) \approx 0.95$$

$$\Rightarrow \quad \mathbb{P}\left(100\times\tfrac{7}{2} - 1.96\sqrt{100\times\tfrac{35}{12}} \le Y \le 100\times\tfrac{7}{2} + 1.96\sqrt{100\times\tfrac{35}{12}}\right) \approx 0.95$$

In other words, I am 95% confident that Y lies in the range $[317, 383]$

Or, more generally, the 95% confidence interval is $[\mathbb{E}Y - 1.96\,\text{sd}(Y),\ \mathbb{E}Y + 1.96\,\text{sd}(Y)]$

or equivalently $[n\mathbb{E}X - 1.96\sqrt{n}\,\text{sd}(X),\ n\mathbb{E}X + 1.96\sqrt{n}\,\text{sd}(X)]$.

§1.6c The Pareto Distribution

The Pareto distribution has been found to arise in Internet traffic measurements. It causes particular problems for simulation and measurement.

The Random Variable Reference Sheet tells us that, if $X \sim \text{Pareto}(\alpha)$, then

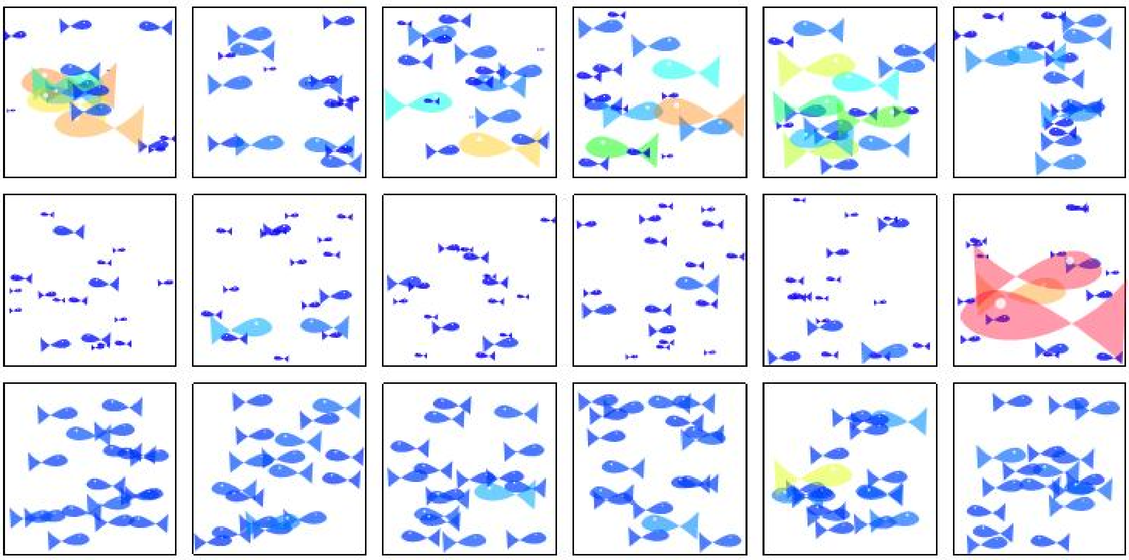
- X takes values in $[1, \infty)$
- X has density $f(x) = \alpha x^{-(\alpha+1)}$.

From this we can work out

- X has distribution function $\mathbb{P}(X \geq x) = x^{-\alpha}$
- $\mathbb{E} X = \begin{cases} \infty & \text{if } \alpha \leq 1 \\ \frac{\alpha}{\alpha-1} & \text{if } \alpha > 1 \end{cases}$
- $\text{Var}\, X = \begin{cases} \infty & \text{if } \alpha \leq 2 \\ \frac{\alpha}{(\alpha-1)^2(\alpha-2)} & \text{if } \alpha > 2. \end{cases}$
- To generate X using the inversion method (§1.4), generate $U \sim \text{Uniform}[0,1]$ and let $X = U^{-1/\alpha}$.

You may also come across more general versions of the Pareto distribution. For example, with density $f(x) = \alpha m^{\alpha} x^{-(\alpha+1)}$ and range $[m, \infty)$.

The Pareto distribution with $\alpha \leq 2$ tends to produce many small values ("mice") and very occassional huge values ("elephants"), and the elephants are so big that they make a significant contribution to the mean.



6 trials, each with 25 fish with size $\sim \text{Exp}(1)$
$\mathbb{E}\,\text{size} = 1$

size $\sim \frac{\alpha-1}{\alpha}\,\text{Pareto}(\alpha)$ with $\alpha = 1.1$
$\mathbb{E}\,\text{size} = 1$

size $\sim \frac{\alpha-1}{\alpha}\,\text{Pareto}(\alpha)$ with $\alpha = 5$
$\mathbb{E}\,\text{size} = 1$

This makes it hard to simulate — you need enough trials, and you need to run them long enough, to have a decent chance of catching the elephants.

§1.7 Independence

The concept of independent random variables is absolutely fundamental in modeling. Random variables X and Y are independent if knowing the value of one of them gives us no information about the value of the other. (Typically we assume that different users make independent requests, but that requests from a single user are not independent.)

Defn Two random variables X and Y are *independent* if

$$\mathbb{P}(X \geq x \text{ and } Y \geq y) = \mathbb{P}(X \geq x)\,\mathbb{P}(Y \geq y) \quad \text{for all } x \text{ and } y$$

Or, equivalently,

$$\mathbb{P}(X \geq x \mid Y \geq y) = \mathbb{P}(X \geq x). \quad \text{for all } x, y$$

Or, for discrete random variables, all we need is

$$\mathbb{P}(X = x \text{ and } Y = y) = \mathbb{P}(X = x)\,\mathbb{P}(Y = y). \quad \text{for all } x, y$$

or $\mathbb{P}(X = x \mid Y = y) = \mathbb{P}(X = x)$ for all x, y. $(*)$

e.g. I throw a fair die. Let Z be the result.

Let $X = Z \bmod 2$ (ie $X = 0$ if $Z = 2, 4$ or 6; $X = 1$ if $Z = 1, 3$ or 5).

Let $Y = Z \operatorname{div} 3$ (ie $Y = 0$ if $Z = 1$ or 2; $Y = 1$ if $Z = 3, 4$ or 5; $Y = 2$ if $Z = 6$).

Are X and Y independent?

Let's use the discrete equation $(*)$.

We need to run through all values of x and y, and check that $(*)$ holds.

- Try $x = 0$, $y = 0$.

  $\mathbb{P}(X=0 \text{ and } Y=0) = \mathbb{P}(Z = 2, 4 \text{ or } 6 \text{ and } Z = 1 \text{ or } 2) = \mathbb{P}(Z=2) = \frac{1}{6}$.

  $\mathbb{P}(X=0)\,\mathbb{P}(Y=0) = \mathbb{P}(Z = 2, 4 \text{ or } 6)\,\mathbb{P}(Z = 1 \text{ or } 2) = \frac{1}{2} \times \frac{1}{3} = \frac{1}{6}$.

  So these

- Try $x = 0$, $y = 1$.

  $\mathbb{P}(X=0 \text{ and } Y=1) = \mathbb{P}(Z = 2, 4 \text{ or } 6 \text{ and } Z = 3, 4 \text{ or } 5) = \mathbb{P}(Z=4) = \frac{1}{6}$

  $\mathbb{P}(X=0)\,\mathbb{P}(Y=1) = \mathbb{P}(Z = 2, 4 \text{ or } 6)\,\mathbb{P}(Z = 3, 4 \text{ or } 5) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$.

  So these values of x and y fail the test.

Thus, X and Y are not independent.

Exercise. Let $X = (Z-1) \bmod 2$ and $Y = (Z-1) \operatorname{div} 2$. Are X and Y independent?

Warning If X and Y are independent, it does NOT follow that
$\mathbb{P}(X = x \text{ or } Y = y) = \mathbb{P}(X = x) + \mathbb{P}(Y = y)$.

§1.8 Working with random variables

For any random variable X,

$$\mathbb{E}(aX+b) = a(\mathbb{E}X) + b \quad \text{for all constants } a \text{ and } b$$

$$\mathrm{Var}(aX+b) = a^2 \,\mathrm{Var}\, X. \quad \text{"}$$

$$\mathrm{sd}(aX+b) = a\,\mathrm{sd}(X) \quad \text{"}$$

For any two random variables X and Y,

$$\mathbb{E}(X+Y) = (\mathbb{E}X) + (\mathbb{E}Y) \qquad (*)$$

For any two independent random variables X and Y,

$$\mathbb{E}(XY) = (\mathbb{E}X)(\mathbb{E}Y)$$

$$\mathrm{Var}(X+Y) = \mathrm{Var}\,X + \mathrm{Var}\,Y$$

$$\mathrm{sd}(X+Y) = \sqrt{\mathrm{sd}(X)^2 + \mathrm{sd}(Y)^2}$$

For two random variables X and Y which are not independent, we measure how related they are by the covariance

$$\mathrm{Cov}(X,Y) = \mathbb{E}\left[(X-\mathbb{E}X)(Y-\mathbb{E}Y)\right].$$

or by the correlation

$$\mathrm{Corr}(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\mathrm{sd}(X)\,\mathrm{sd}(Y)}.$$

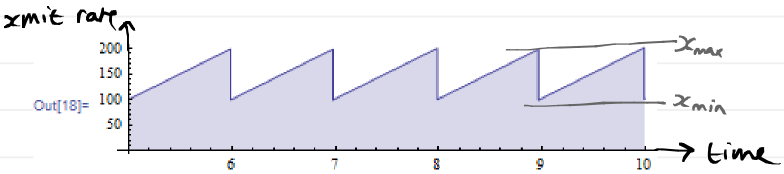
With a little algebra, we find

$$\begin{aligned}
\mathrm{Var}(X+Y) &= \mathbb{E}\left[X+Y-\mathbb{E}(X+Y)\right]^2 \\
&= \mathbb{E}\left[X+Y-(\mathbb{E}X+\mathbb{E}Y)\right]^2 && \text{by } (*) \\
&= \mathbb{E}\left[(X-\mathbb{E}X)+(Y-\mathbb{E}Y)\right]^2 && \text{by rearranging} \\
&= \mathbb{E}(X-\mathbb{E}X)^2 + \mathbb{E}(Y-\mathbb{E}Y)^2 + 2\mathbb{E}\left((X-\mathbb{E}X)(Y-\mathbb{E}Y)\right) && \text{by algebra} \\
&= \mathrm{Var}\,X + \mathrm{Var}\,Y + 2\,\mathrm{Cov}(X,Y). && \text{by definition of Var, Cov.}
\end{aligned}$$

Exercise. Example sheet 2 has a variety of questions to build up your skill at working with random variables.

Example: Statistical multiplexing

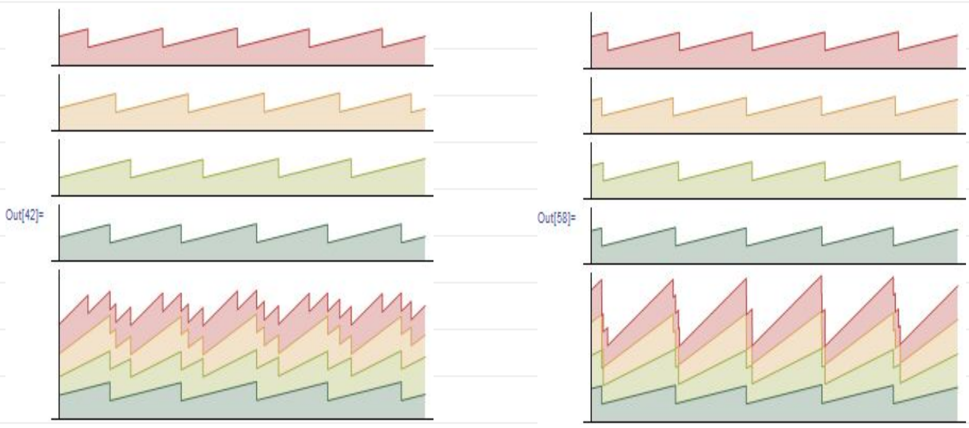
For most traffic flows on the Internet, the rate at which data is transmitted is controlled by TCP. This algorithm steadily increases transmission rate (by 1pkt/sec every round-trip time) until it detects congestion in the form of a dropped packet, whereupon it cuts its transmission rate in half.
This behaviour produces the characteristic "TCP sawtooth".



It's more convenient to think in terms of average xmit rate. Suppose that packet drops are periodic, hence xmit rate varies between x_{min} and x_{max}, and average xmit rate is $x = \frac{1}{2}(x_{min} + x_{max})$.
Also, we know from the "cut by half" rule that $x_{min} = \frac{1}{2} x_{max}$.
We can then solve:

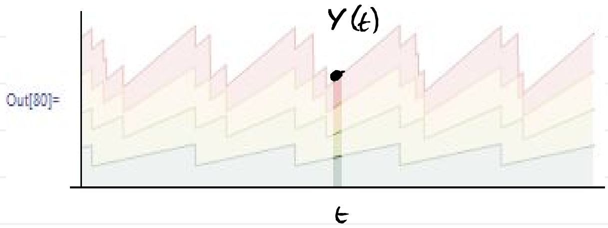
$$x = \frac{1}{2}(x_{min} + x_{max}) = \frac{1}{2}\left(\frac{1}{2}x_{max} + x_{max}\right) = \frac{3}{4} x_{max} \Rightarrow x_{max} = \frac{4}{3}x$$
$$x_{min} = \frac{2}{3}x.$$

Suppose a network operator wants to lay in enough capacity to support 1000 users each running at 30 kB/sec. How much capacity is needed?



In the worst case, all the sawteeth might be aligned, giving peak rate of $1000 \times \frac{4}{3} \times 30$ kB/sec $= 40$ MB/sec. But intuitively we might guess that perfect alignment is unlikely, and that the peaks on one sawtooth cancel out those on another. If this happens, how much capacity is needed?

At an arbitrary instant in time t, let $X_1(t), \cdots, X_n(t)$ be the transmit rates of n flows, and let $Y(t) = X_1(t) + \cdots + X_n(t)$ be the total traffic.



We might have caught a flow at any point in its sawtooth, so each X_i might take any value in the range $[\frac{2}{3}x, \frac{4}{3}x]$ where $x = 30$ kB/s is the average rate we want to support. Furthermore, each value in this range is equally likely. The appropriate distribution is

$$X_i \sim \text{Uniform}\left(\tfrac{2}{3}x, \tfrac{4}{3}x\right).$$

$$\mathbb{E}X_i = \tfrac{1}{2}\left(\tfrac{2}{3}x + \tfrac{4}{3}x\right) = x$$

$$\text{Var}\, X_i = \frac{\left(\frac{4}{3}x - \frac{2}{3}x\right)^2}{12} = \frac{x^2}{27}$$

$$\text{sd}\, X_i = \sqrt{\text{Var}\, X_i} = \frac{x}{\sqrt{27}}.$$

If all the $X_i(t)$ are independent, then

$$\mathbb{E}Y = \mathbb{E}(X_1 + \cdots + X_n) = \mathbb{E}X_1 + \cdots + \mathbb{E}X_n = x + \cdots + x = nx$$

$$\text{Var}\, Y = \text{Var}(X_1 + \cdots + X_n) = \text{Var}X_1 + \cdots + \text{Var}X_n = \frac{x^2}{27} + \cdots + \frac{x^2}{27} = \frac{nx^2}{27}$$

$$\text{sd}(Y) = \sqrt{\text{Var}\, Y} = \sqrt{n}\,\frac{x}{\sqrt{27}}.$$

Recall, Y is a random variable, i.e. every time you sample it you get a different answer. We know from §1.6b that, roughly 95% of the time, an observation of Y will lie in the range

$$[\mathbb{E}Y - 2\,\text{sd}(Y),\ \mathbb{E}Y + 2\,\text{sd}(Y)].$$

In this case, 95% of the time, for $n = 1000$ and $x = 30$ kB/s, the total traffic rate Y lies in the range

$$\left[nx - 2\sqrt{n}\frac{x}{\sqrt{27}},\ nx + 2\sqrt{n}\frac{x}{\sqrt{27}}\right] = [29.8, 30.2] \text{ MB/s}.$$

This is much smaller than the worst-case value 40 MB/s.

Exercise. Suppose the capacity of the link is actually 32 MB/s.

- Explain why the overall fraction of traffic that is lost is $\frac{\mathbb{E}[(Y-32)^+]}{\mathbb{E}Y}$, and not $\mathbb{E}\left[\frac{(Y-32)^+}{Y}\right]$.

  Here, $(Y-32)^+$ is shorthand for $\max(Y-32, 0)$.

- Calculate or compute the fraction of traffic that is lost.

  Hint: $\mathbb{E}[(Y-32)^+] = \int_{-\infty}^{\infty} (y-32)^+ f(y)\, dy$ where $f(y)$ is the density

$$= \int_{32}^{\infty} (y-32) f(y)\, dy$$

$$= \int_{32}^{\infty} y f(y)\, dy - 32\, \mathbb{P}(Y \geq 32).$$

You can find the integral either by numerical integration or by integration-by-parts. You can find $\mathbb{P}(Y \geq 32)$ either by a lookup table, a built-in function in a stats library, or numerical integration.

§1.9 Experiment design

Suppose for example that I have programmed a simulator of a link shared by TCP flows, and I want to use my simulator to learn how long a typical flow takes to complete.

What quantities should I measure, and how should I report them?

STEADY STATE.

We are usually interested in "steady state" readings. For example, suppose we run a simulation, and record flow durations. We would hope that the flow durations from the beginning of the simulation are similar to those at the end. There are two main issues to face:

- The system might be unstable, e.g. more flows arrive than can be served. If this is the case, the link gets busier and busier the longer we run the simulation, so the flow durations get longer and longer. If this is so, there is no such thing as "typical flow duration", and it is useless to report average flow duration.

  You should always check if your simulator is unstable, before reporting summary statistics. Two good checks:
  - Plot a trace, i.e. some measure of system state as a function of time. Look by eye to see if it seems stable.
  - Run your simulator and measure mean flow duration. Run it again for twice as long. Are the results comparable?

- The system might take some time to reach steady state, especially if you started it from a very atypical initial state. Normally you should discard the first few readings. There is no general rule for deciding how many readings to discard — you should simply plot a trace and judge by eye how long it takes to reach steady state.

"MEAN" MEASUREMENTS FROM A SINGLE RUN

Suppose I run my simulator and record the completion times of all the flows, $X_1, \dots, X_n$. From these readings, I can report

- the average completion time, $\bar{X} = \frac{1}{n}(X_1 + \dots + X_n)$
- the empirical distribution for waiting time, $\hat{F}(x) = \frac{1}{n}$ # of readings that are $\geq x$
- a 95% confidence interval for waiting time, i.e. a range $[x_1, x_2]$ where x_1, x_2 are chosen such that 95% of my observations lie in that range
  e.g. discard the lowest 2.5% of the values, and the highest 2.5%, and let x_1 and x_2 be the min and max of what remains.
- It's rarely worth reporting the min and max of the n readings, because the more readings you take the more extreme the min and max will be. Therefore, min and max depend on your specific experimental setup, and they are not good guides to future behaviour.
  The other three quantities (average, empirical distribution function, 95% conf.int) will get <u>more accurate</u> the more readings you take.

An important caveat:

- You may have been unlucky with your particular run, e.g. your measured $\bar{X}$ might be atypically low. You should also report how confident you are that you are reporting a typical value — see below.

MEASUREMENTS FROM MULTIPLE RUNS

Suppose I am interested in the average completion time.
From a single run I can compute the average — but how confident am I that my run was not atypical?

Answer: run several runs, and compute the average completion time for each run. Say we get averages $\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_m$ from m runs.

- The overall average is $\bar{\bar{x}} = \frac{1}{m}(\bar{x}_1 + \cdots + \bar{x}_m)$.
- We can compute a 95% confidence interval for the average completion time by discarding the lowest 2.5% and the highest 2.5% of the $\bar{x}_i$, and finding the min and max of what remains.

It is good practice to always report both $\bar{\bar{x}}$ and a 95% confidence interval for the average. When you draw plots, indicate the confidence interval with an error bar, and say that your error bars show a 95% conf. int.

Make sure you understand the difference between
(a) a 95% confidence interval for completion time, and
(b) a 95% confidence interval for average completion time.
If we take very many measurements over very many runs, we will end up with a very accurate conf. int. for completion time, and a very small conf. int. for average completion time.

An important caveat:
Recall the discussion of mice & elephants from §1.6c.
For some input distributions (e.g. if flow sizes are Pareto(α) with $\alpha \le 2$), you are likely to see many $\bar{x}_i$ that are smaller than the true mean, and occassionally an $\bar{x}_i$ that is much larger. To get an accurate reading of $\bar{\bar{x}}$, you need to run enough simulations to catch some of the elephants.

APPROXIMATE CONFIDENCE INTERVALS BASED ON STANDARD DEVIATION

Instead of discarding the top and bottom 2.5% (which is awkward if we only have a small number of readings), here is an approximation.

1. Pretend that the measurements $\bar{X}_1, \dots, \bar{X}_m$ are independent observations from the Normal(μ, σ^2) distribution, for some parameters μ and σ^2.

2. Use the maximum likelihood method (§1.5) to estimate μ and σ^2:

$$\hat{\mu} = \frac{1}{m}(\bar{x}_1 + \dots + \bar{x}_m) = \bar{\bar{x}}$$

$$\hat{\sigma} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\bar{x}_i - \hat{\mu})^2}$$

3. An approximate 95% confidence interval for μ is

$$\left[\hat{\mu} - 1.96\frac{\hat{\sigma}}{\sqrt{m-1}}, \ \hat{\mu} + 1.96\frac{\hat{\sigma}}{\sqrt{m-1}}\right].$$

CAUTION: This approximation is only accurate if the $\bar{X}_i$ are independent. Thus you can use it to find a confidence interval for the average completion time, but you cannot use it to find a confidence interval for completion time.

Where does the approx. come from? [NON-EXAMINABLE]

- The central limit theorem tells us that $\sum_i \bar{X}_i \sim \text{Normal}(m\mu, m\sigma^2)$
- Hence $\hat{\mu} = \frac{1}{m}\sum X_i \sim \text{Normal}(\mu, \frac{1}{m}\sigma^2)$.
- By the standard approximation,

$$\mathbb{P}\left(\mu - 1.96\frac{\sigma}{\sqrt{m}} \le \hat{\mu} \le \mu + 1.96\frac{\sigma}{\sqrt{m}}\right) \approx 95\%.$$

- Rearranging, $\mathbb{P}\left(\hat{\mu} - 1.96\frac{\sigma}{\sqrt{m}} \le \mu \le \hat{\mu} + 1.96\frac{\sigma}{\sqrt{m}}\right) \approx 95\%.$
- The formula above for the 95% confidence interval uses the estimate $\hat{\sigma}$ rather than the true value σ. The difference $\sqrt{m}/\sqrt{m-1}$ is a "penalty" we pay for using the estimate rather than the true value.

CAUTION: This approximation is only accurate if the $\bar{X}_i$ have finite variance. If they have infinite variance, then the normal approximation is inaccurate (see §1.6b). If the input data is heavy-tailed (§1.6c) then the $\bar{X}_i$ may have infinite variance.

§1.10 Abstract random variables [NON-EXAMINABLE]

We have so far been concerned with random numbers, and we've thought of them as outcomes from measurements or from a random number generator. But random variables can be used to describe many other things, and they can be interpreted in different ways.

Defn A <u>random variable</u> consists of a set of possible outcomes Ω, and a function $\mathbb{P}$ which assigns a <u>probability value</u> to subsets of Ω.
Subsets of Ω are called <u>events</u>.
Elements of Ω are called <u>outcomes</u>.

e.g. I pick a card at random from a deck of playing cards.
Let S be the suit.

$$\Omega = \{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$$

$$\mathbb{P}(X \in \{\diamondsuit, \heartsuit\}) = \tfrac{1}{2} \quad \text{etc.}$$

e.g. I toss a coin every bus I see, stopping when I see a 134 bus.
Let X be the sequence of coin tosses.

$$\Omega = \{ (H), (T), (HH), (HT), (TH), (TT), (HHH), (HHT), \dots \}$$

The definition of independence needs to be generalized:

Defn Two random variables X and Y are <u>independent</u> if

$$\mathbb{P}(X \in A \text{ and } Y \in B) = \mathbb{P}(X \in A)\,\mathbb{P}(Y \in B) \quad \text{for all events } A, B.$$

Equivalently,

$$\mathbb{P}(X \in A \mid Y \in B) = \mathbb{P}(X \in A) \quad \text{for all events } A \text{ and } B.$$

PHILOSOPHY OF PROBABILITY

What "is" probability $\mathbb{P}(\cdot)$? We've interpreted in two ways in this section: as a description of the characteristics of a random number generator; and as long-run frequencies in repeated measurements of the real world.
There is a third common interpretation: as representing subjective belief.

To a computer scientist:

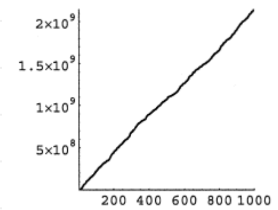
"The generation of random numbers is too important to be left to chance"
— Robert R. Coveyou

"Anyone who considers arithmetical methods of producing random numbers is, of course, in a state of sin"
— John von Neumann

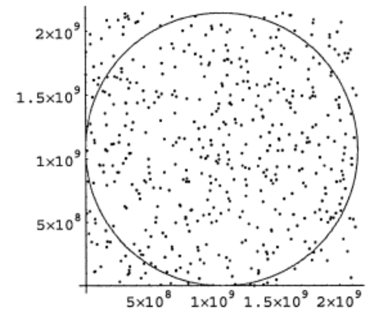
A random number is what is generated by a good random number generator.

— a long period before it repeats (2^{250} is considered long)

— all numbers are equally likely
e.g. Generate N random numbers, sort them, and plot. Should produce something close to a straight line.



— it passes other tests which probabilists say are the hallmarks of random numbers. e.g. the circle test: Plot pairs of numbers, and count what fraction lie in the unit circle. It should be close to $\pi/4$.



But in these tests, how close is close enough?

To an experimentalist:

e.g. Suppose I have a biased coin, for which the probability of heads is p, and I toss it n times, and I get N heads.

Then N is a random variable. It should be close to Np.
How much variability should I expect?

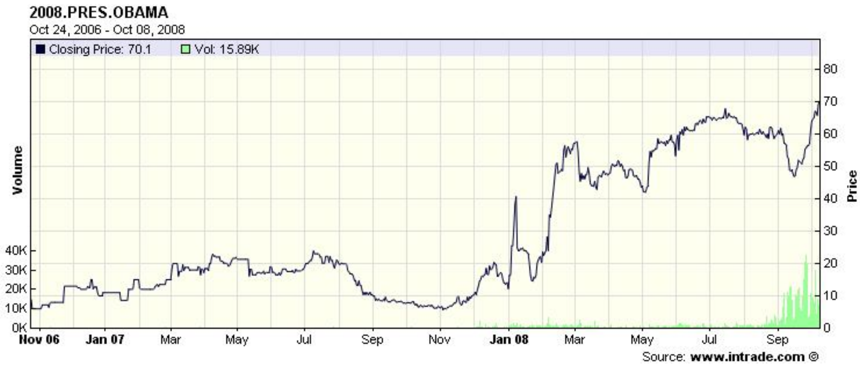
To a Bayesian:

Probability represents "subjective belief".

e.g. let X = next president of the U.S.

Then $X \in \{$Obama, McCain$\}$ (assuming nothing bizarre happens)

and I may believe $P(X = \text{Obama}) = 60\%$.



http://www.intrade.com/jsp/intrade/trading/t_index.jsp?selConID=409933

This is obviously not something which makes sense to an experimentalist (how could we get N samples of the US election?).

There are mathematical rules which explain how "subjective belief" should be updated in the light of evidence — Bayes' Rule —

and this is useful for e.g. spam filtering, trust ratings in reputation systems.

To a probabilist:

We don't try to explain what random variables are, or what they mean, we just explain how they behave.

§2 Random Processes

Most questions about network performance are questions about <u>random processes</u>, i.e. systems which evolve in time and are driven by random happenings.

Example 1 Let Q_t = # pkts in a queue at time t.
The entire "execution trace" of a simulation, ie the function $t \mapsto Q_t$, is random; it is referred to as a random process, and is sometimes written $(Q_t, t \geq 0)$.
It is driven by the sequence of packet inter-arrival times and by the packet service times, each of which may be random.
Q. What is the distribution of Q_t?
What is $\mathbb{P}(Q_t = B)$ where B is buffer size?
What is $\mathbb{E}$ queueing delay?

Example 2 Let N_t = # active TCP flows on a link at time t.
Assume that each flow is to transfer a file, and file sizes are random, with mean size m Mbit.
Suppose that flow inter-arrival times are all Exp(λ) random variables.
Suppose that the link has capacity C Mbit/s.
Q. What is $\mathbb{E}$ # jobs active?
What is average flow completion time?
What is $\mathbb{P}$(link is idle)?

Example 3 Consider a sequence of packets sent over a noisy wireless link.
Let $E_n = \begin{cases} 1 & \text{if packet } \#n \text{ is corrupted} \\ 0 & \text{if it is not.} \end{cases}$
Suppose we have a measurement trace 000111000000011000000...
It looks like noise comes in bursts, so we can't model this as a sequence of independent random variables.
What sort of random process might provide a good fit for the data, so that we can implement it in a simulator?

There are two major classes of random process that we will study:

- discrete time (like Example 3: the time index n is an integer)
- continuous time (like Examples 1 & 2: the time index t is a real number)

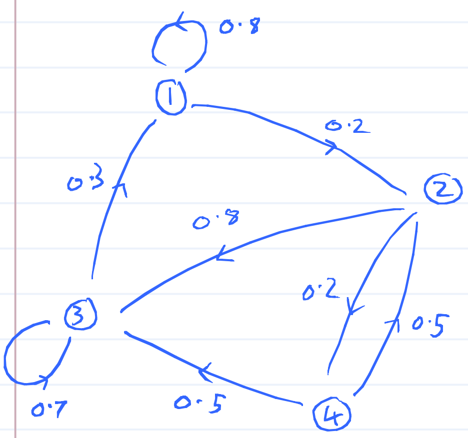
We will study Markov chains (discrete time), and Markov processes (continuous time).

§ 2.1 Markov Chains

Informally: There's a set of possible states. Each timestep you jump to a new state. The jump is random, and its distribution depends only on where you are now, not on any previous history.

There are two ways to write out the jump distributions:

state space diagram

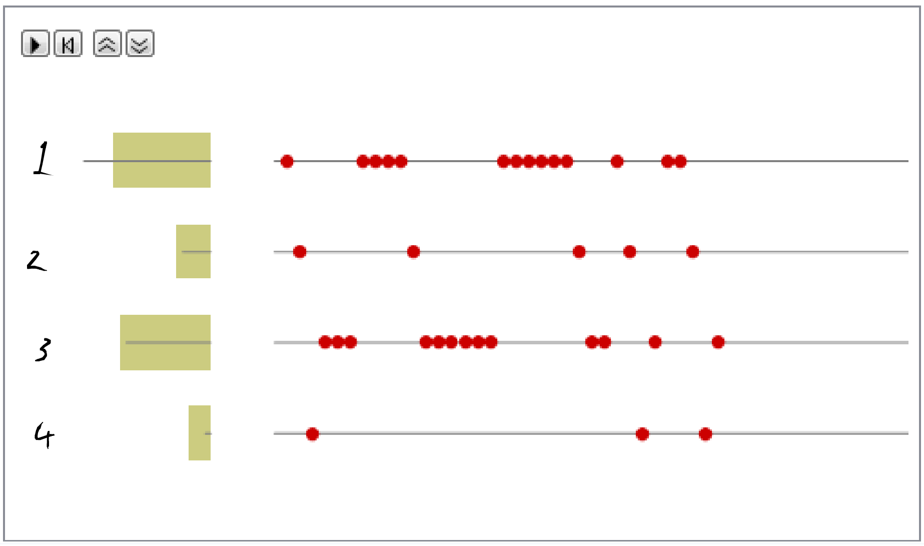


transition matrix

$$P = \begin{array}{c} \\ ① \\ ② \\ ③ \\ ④ \end{array} \begin{array}{c} \begin{array}{cccc} ① & ② & ③ & ④ \end{array} \\ \begin{bmatrix} 0.8 & 0.2 & 0 & 0 \\ 0 & 0 & 0.8 & 0.2 \\ 0.3 & 0 & 0.7 & 0 \\ 0 & 0.5 & 0.5 & 0 \end{bmatrix} \end{array}$$

$$P_{ij} = \mathbb{P}(\text{next state is } j \mid \text{current state is } i)$$

Simulation:



```
#-------------------------------------------------------
# Simulator of a Markov chain

import random

def randomselect(p):
    '''Given a list of probabilities p, pick index i with probability p[i] and return
    r = random.random()
    i = 0
    d = p[0]
    while r>d:
        i = i+1
        d = d+p[i]
    return i

p = [[.8,.2,0,0],
     [0,0,.8,.2],
     [.3,0,.7,0],
     [0,.5,.5,0]]

# Simulate 1000 steps of the Markov chain, starting from state 0
trace = [0]
for i in range(1000):
    s = trace[-1]  # our current state
    jumpprob = p[s]  # the probability distribution saying where to jump next
    nexts = randomselect(jumpprob)
    trace.append(nexts)

# Count the total number of visits to each state
visits = [0 for i in p]
for s in trace:
    visits[s] = visits[s]+1
print visits
# Measure the fraction of time spent in each state
print [v/float(sum(visits)) for v in visits]


# Alternative simulator, using a generator.
# Also, using a defaultdict to store visit counts, so we don't need
# to explicitly initialize it.
def markovchain(p,s=0):
    while True:
        yield s
        s = randomselect(p[s])
#
X = markovchain(p)
import collections
visits = collections.defaultdict(int)
for i in range(10000): visits[X.next()] += 1
print visits
```

When we simulate a Markov chain, we typically find that there is a vector π such that

- the fraction of time spent in state i approaches π_i, the longer we run the simulator
- if we interrupt the simulation and see what state it's in, the probability of finding it in state i approaches π_i the longer we run the simulator.

This distribution π is called the <u>equilibrium distribution</u> and there is a systematic way to find it:

1. Write out the <u>balance equations</u> $\pi_i = \sum_j \pi_j P_{ji}$, one equation for each state i

   the probability of finding the system in state i now

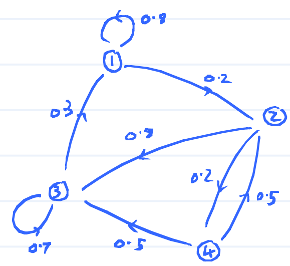
   sum over all possible states j it might have been in in the last timestep

   probability that it was in state j, and then it jumps to state i.

2. Write out the <u>normalization equation</u> $\sum_i \pi_i = 1$.

3. Solve all these equations simultaneously.

<u>Example</u> For the earlier example,



$$\pi_1 = 0.8\,\pi_1 + 0.3\,\pi_3$$
$$\pi_2 = 0.2\,\pi_1 + 0.5\,\pi_4$$
$$\pi_3 = 0.8\,\pi_2 + 0.7\,\pi_3 + 0.5\,\pi_4$$
$$\pi_4 = 0.2\,\pi_2$$

Eliminate π_4 using the last equation, $\pi_4 = 0.2\,\pi_2$

$$\pi_1 = 0.8\,\pi_1 + 0.3\,\pi_3$$
$$\pi_2 = 0.2\,\pi_1 + 0.5 \times 0.2\,\pi_2 = 0.2\,\pi_1 + 0.1\,\pi_2$$
$$\pi_3 = 0.8\,\pi_2 + 0.7\,\pi_3 + 0.5 \times 0.2\,\pi_2 = 0.9\,\pi_2 + 0.7\,\pi_3.$$

Eliminate π_3 using the first equation, $0.2\,\pi_1 = 0.3\,\pi_3 \Rightarrow \pi_3 = \frac{2}{3}\pi_1$

$$\pi_2 = 0.2\,\pi_1 + 0.1\,\pi_2$$
$$\tfrac{2}{3}\pi_1 = 0.9\,\pi_2 + 0.7 \times \tfrac{2}{3}\pi_1 \Rightarrow \pi_1 = 1.35\,\pi_2 + 0.7\,\pi_1$$

Eliminate π_2 using the first equation, $0.9\,\pi_2 = 0.2\,\pi_1 \Rightarrow \pi_2 = \frac{2}{9}\pi_1$

$$\pi_1 = 1.35 \times \tfrac{2}{9}\pi_1 + 0.7\,\pi_1 \Rightarrow 9\pi_1 = 2.7\,\pi_1 + 6.3\,\pi_1 = 9\,\pi_1.$$

In fact, this last step is always futile...

We've shown $\pi = \left(\pi_1,\ \frac{2}{9}\pi_1,\ \frac{2}{3}\pi_1,\ \frac{1}{5} \times \frac{2}{9}\pi_1\right)$.

We also know $\pi_1 + \pi_2 + \pi_3 + \pi_4 = 1$ because π is a distribution.

$$\Rightarrow \pi_1\left(1 + \tfrac{2}{9} + \tfrac{2}{3} + \tfrac{1}{5} \times \tfrac{2}{9}\right) = 1$$
$$\Rightarrow \pi_1 = \tfrac{45}{87}.$$

Thus $\pi = \left(\frac{45}{87}, \frac{10}{87}, \frac{30}{87}, \frac{2}{87}\right)$.

USING A COMPUTER TO FIND THE EQM DISTRIBUTION.

If we write out the balance equations in matrix form, we can solve them with a computer:

$$\pi = \pi P$$

$$\Rightarrow \quad \pi I = \pi P \qquad \text{where } I \text{ is the identity matrix } \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix}$$

$$\Rightarrow \quad \pi(P-I) = 0$$

Of course, we still need to solve the normalization equation too.

Many computer languages have library routines for solving $\pi(P-I)=0$, e.g. in R

```
# load in the library
library(MASS)
# Set up the rate matrix
p <- rbind(c(.8,.2,0,0),
           c(0,0,.8,.2),
           c(.3,0,.7,0),
           c(0,.5,.5,0))
# Calculate p-I
pp <- p - diag(1,nrow=nrow(p))
# Solve the balance equations pi*(p-i) = 0
eqm <- Null(pp)[,1]
# and rescale the answer to get a distribution
eqm <- eqm/sum(eqm)
eqm
```

A SHORTCUT THAT SOMETIMES WORKS

For some Markov chains, you can fairly easily find a distribution π such that

$$\pi_i P_{ij} = \pi_j P_{ji} \quad \text{for all } i \text{ and } j$$

(the <u>detailed balance</u> equations)

If this is so, then π automatically solves the balance equations.

Always give this a go first, in case it works.

§2.2 The billion-dollar formula

$$R_i = 1-d + d \sum_{j \to i} \frac{R_j}{\Delta_j}$$

where $d = 0.85$

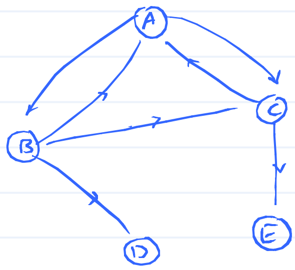
i is a web page

$\sum_{j \to i}$ means: sum up over all pages j that link to page i

Δ_j = number of links on page j.

This is the formula that defines PageRank.

It turns out that this formula arises naturally from a certain Markov Chain...



1. Start with a graph of the web, showing which pages link to which other pages. Let there be N pages in total.

2. Imagine a "random web surfer" who jumps from page to page, according to the following rules:
   - start at an arbitrary node
   - if you're at a dangling node (i.e. a node with no outgoing links) then pick one of the N pages at random (uniformly) and jump to it.
   - otherwise, toss a biased coin, with $\mathbb{P}(\text{heads}) = d$:
     - If heads, pick an outgoing link at random, and follow it
     - If tails, pick one of the N pages at random (uniformly) and jump to it.

$$P = \begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & \frac{1-d}{5} & \frac{d}{2}+\frac{1-d}{5} & \frac{d}{2}+\frac{1-d}{5} & \frac{1-d}{5} & \frac{1-d}{5} \\ B & \frac{d}{3}+\frac{1-d}{5} & \frac{1-d}{5} & \frac{d}{3}+\frac{1-d}{5} & \frac{d}{3}+\frac{1-d}{5} & \frac{1-d}{5} \end{array}$$

In other words, the random surfer is a Markov chain with transition probabilities

$$P_{ij} = \begin{cases} \frac{1}{N} & \text{if } i \text{ is dangling} \\ \frac{d\,1_{i \to j}}{\Delta_i} + \frac{1-d}{N} & \text{otherwise} \end{cases}, \quad \text{where } 1_{i \to j} = \begin{cases} 1 & \text{if } i \text{ links to } j \\ 0 & \text{otherwise} \end{cases}$$

3. Find the equilibrium distribution π.

It turns out that $R_i = \text{const} \times \pi_i$

THEOREM. If π is the equilibrium distribution of the Markov chain for the Random Surfer, and R is the PageRank defined by

$$R_i = 1-d + d \sum_{j \to i} R_j/\Delta_j$$

then there is some number θ such that $R_i = \theta \pi_i$ for all i.

PROOF. Since π is the equilibrium distribution, we know

$$\pi_i = \sum_j \pi_j P_{ji} \text{ for all } i, \quad \text{and} \quad \sum_i \pi_i = 1.$$

Using these facts,

$$\pi_i = \sum_j \pi_j P_{ji}$$

$$= \sum_j \pi_j \begin{cases} \frac{1}{N} & \text{if } j \text{ is dangling} \\ (1-d)\frac{1}{N} + d\,\frac{1_{j\to i}}{\Delta_j} & \text{otherwise} \end{cases}$$

$$= \sum_j \pi_j \begin{cases} (1-d)\frac{1}{N} + d\frac{1}{N} & \text{if } j \text{ is dangling} \\ (1-d)\frac{1}{N} + d\,\frac{1_{j\to i}}{\Delta_j} & \text{otherwise} \end{cases}$$

$$= (1-d)\frac{1}{N}\sum_j \pi_j + d\sum_j \pi_j \begin{cases} \frac{1}{N} & \text{if } j \text{ is dangling} \\ \frac{1_{j\to i}}{\Delta_j} & \text{otherwise} \end{cases}$$

$$= \frac{1-d}{N} + \frac{d}{N}\sum_{j \text{ dangling}} \pi_j + d \sum_{j \text{ not dangling}} \pi_j \frac{1_{j\to i}}{\Delta_j} \qquad \text{using } \sum_j \pi_j = 1$$

$$= \frac{1-d}{N} + \frac{d}{N}\sum_{j \text{ dangling}} \pi_j + d\sum_{j\to i} \frac{\pi_j}{\Delta_j}.$$

Now, rescale this: let $\theta = \dfrac{N}{1+\frac{d}{1-d}\sum_{j \text{ dangling}} \pi_j}$ and let $R_j = \theta \pi_j$.

Then,
$$R_i = \left(\frac{1-d}{N} + \frac{d}{N}\sum_{j \text{ dangling}} \pi_j\right)\theta + d\sum_{j\to i}\frac{\theta\pi_j}{\Delta_j}$$

$$= \left(\frac{1-d}{N}\right)\left(1 + \frac{d}{1-d}\sum_{j \text{ dangling}} \pi_j\right)\theta + d\sum_{j\to i}\frac{R_j}{\Delta_j}$$

$$= 1-d + d\sum_{j\to i}\frac{R_j}{\Delta_j}.$$

Obviously, to calculate your PageRank you need to know the entire graph of the web. But — understanding that PageRank is merely a rescaled version of the equilibrium distribution of the random surfer, you can plan:

"How can I adjust my within-site links so that, once the random surfer enters my site, he/she is pretty much trapped there?"

This will increase the long-run fraction of time he/she spends in your site, hence it will increase your pagerank.

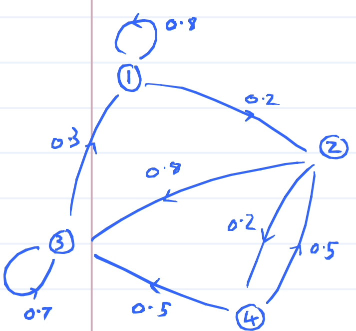
§2·3 Other calculations with Markov chains

Recall the balance equations: let $\pi_i = \mathbb{P}(\text{in state } i)$ in equilibrium, so that

$$\pi_i = \mathbb{P}(\text{in state } i) = \sum_j \mathbb{P}(\text{in state } i, \text{ and was in } j \text{ last timestep})$$
$$= \sum_j \mathbb{P}(\text{in state } i \,|\, \text{was in } j \text{ last timestep})\, \mathbb{P}(\text{was in } j \text{ last timestep})$$
$$= \sum_j P_{ji}\, \pi_j$$

Note Similar recursions can be used to calculate hitting times:
let t_i = mean number of hops, starting from i, until we hit some specific state e.
$$= 1 + \sum_j \text{mean number of extra hops starting from } j, \text{ if we jumped to } j$$
$$= 1 + \sum_j P_{ij}\, t_j .$$
Also $t_i = 0$ if $i = e$, of course.

Note Similar recursions also let us calculate hitting probabilities
let $q_i = \mathbb{P}(\text{we end up reaching state } e \text{ before we hit state } f, \text{ starting at } i)$.
$$= \sum_j \mathbb{P}(\text{we first jump to } j, \text{ and then we reach } e \text{ before hitting } f)$$
$$= \sum_j P_{ij}\, q_j$$
Also $q_e = 1$ and $q_f = 0$, of course.



Example Let t_i = mean number of hops, starting from i, until we hit state 1.

Then
$$\left.\begin{aligned} t_1 &= 0 \\ t_2 &= 1 + 0{\cdot}8\, t_3 + 0{\cdot}2\, t_4 \\ t_3 &= 1 + \cancel{0{\cdot}3\, t_1} + 0{\cdot}7\, t_3 \\ t_4 &= 1 + 0{\cdot}5\, t_3 + 0{\cdot}5\, t_2 \end{aligned}\right\}$$

The final solution is $(t_1, t_2, t_3, t_4) = (0, \frac{14}{3}, \frac{10}{3}, 5)$.

Section 2.4

Formal properties of Markov chains

Network Performance—DJW—2010/2011

Formally, we say that the random sequence $(X_0, X_1, \ldots)$ is a Markov chain with state space S, transition matrix $P \in [0,1]^{|S|\times|S|}$ and initial distribution $\rho \in [0,1]^{|S|}$ if, for all n and $x_0, x_1, \ldots, x_n \in S$

$$\mathbb{P}(X_0 = x_0, X_1 = x_1, \ldots, X_n = x_n) = \rho_{x_0} P_{x_0 x_1} \cdots P_{x_n x_{n-1}}.$$

Equivalently,

$$\mathbb{P}(X_n = x_n | X_{n-1} = x_{n-1}, \ldots, X_0 = x_0) = P_{x_{n-1} x_n} \quad \text{and} \quad \mathbb{P}(X_0 = x_0) = \rho_{x_0}.$$

Definition. A chain is *irreducible* if for every pair of states $x, y \in S$ there is a path in the state space diagram from x to y. The path may have multiple steps.

Definition. A chain is *aperiodic* if for every state x there is an integer n_x such that there is an n_x-hop path from x to x, and also a path with n_x+1 hops, and a path with n_x+2 hops, and so on. *

Theorem. If a Markov chain is irreducible and aperiodic, and if it has finitely many states, then it is possible to solve the balance and normalization equations, and this solution is unique. The solution is called the equilibrium distribution, usually written π. Furthermore,

- The equilibrium distribution is *invariant*, also known as *stationary*. That is, if $\mathbb{P}(X_n = j) = \pi_j$ for all j, then $\mathbb{P}(X_{n+1} = j) = \pi_j$ for all j.
- The equilibrium distribution is *limiting*. That is, no matter what the distribution of X_0 is, $\mathbb{P}(X_n = j) \to \pi_j$ as $n \to \infty$ for all j.
- The equilibrium distribution is *ergodic*. That is, for every simulation we run, if we let $V_n(i)$ be the number of times the chain visits state j in the first time steps, then $V_n(j)/n \to \pi_j$ as $n \to \infty$ for all j.

Theorem. If a Markov chain is irreducible and aperiodic, and it has infinitely many states, then it is always possible to solve the balance equations. It may or may not be possible to also solve the normalization equation.

- If the normalization equation can be solved, then π is invariant, limiting, and ergodic, as in the finite case. Also the process is *recurrent*, i.e. starting from any state j, $\mathbb{P}(\text{eventually returns to } j) = 1$. Furthermore it is *positive recurrent*, i.e. starting from any state j, $\mathbb{E}(\text{time to return to } j) < \infty$.
- If the normalization equation cannot be solved, then either
  - (i) the chain is *transient*, the opposite of recurrent, i.e. there is some state j such that, starting from state j, $\mathbb{P}(\text{eventually return to } j) < 1$; or
  - (ii) the chain is recurrent but not positive recurrent.

  In other words, either it might never return, or it always returns but it can take a very very long time.

Theorem. For any Markov chain, the equations for hitting time can always be solved, and the solution is unique.

Theorem. For any Markov chain, the equations for hitting probability can always be solved.

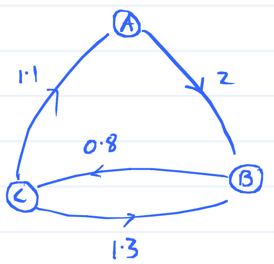
- If the chain has finitely many states, the solution is unique.
- If the chain has infinitely many states, the equations may have more than one solution. If this is the case, then the true hitting probability is given by the smallest non-negative solution.

* Useful fact: If the chain is irreducible, and you find one state x with the aperiodic property, then all the other states automatically have the aperiodic property also.

§ 2.5 Markov Processes

A Markov process is the continuous-time equivalent of a Markov chain. There is a set of states. When you arrive in a state, you wait there for a random (Exponential) length of time. Then you jump to a new state. The wait and the jump are both random, and their distributions depend on the current state. They are independent of all previous jumps & waits.

state space diagram

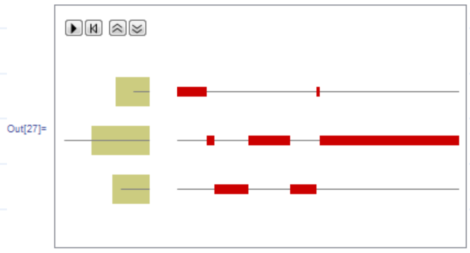


transition rate matrix r

$$
r = \begin{array}{c} \\ A \\ B \\ C \end{array}
\begin{array}{c}
\begin{array}{ccc} A & B & C \end{array} \\
\begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 0.8 \\ 1.1 & 1.3 & 0 \end{bmatrix}
\end{array}
$$

diagonal entries all $= 0$

Random sample path



Here is how to generate/simulate a Markov process.
Let S be the set of all states. In this example, $S = \{A, B, C\}$.

1. Pick some arbitrary starting state I.
2. Generate an Exponential random variable $T \sim \text{Exp}\left(\sum_{j \in S} r_{Ij}\right)$
   Generate a random variable J, with range S,
   where $P(J = j) = r_{Ij} \Big/ \sum_{k \in S} r_{Ik}$
3. Wait in state I for time T, then jump to state J.
   Let $I \leftarrow J$, and go to step 2.

```python
#--------------------------------------------------------
# Simulator of a Markov process

import random, math

def randomselect(p):
    '''Given a list of probabilities p, pick index i with probability p[i] and return i'''
    r = random.random()
    i = 0
    d = p[0]
    while r>d:
        i = i+1
        d = d+p[i]
    return i

def rexp(rate): return -1.0/rate * math.log(random.random())


r = [[0,2,0],
     [0,0,.8],
     [1.1,1.3,0]]

# Calculate the jump probabilities, and the rates for waiting time
p = [[rij/sum(row) for rij in row] for row in r]
d = [sum(row) for row in r]

# Simulate 10000 steps, starting from state 0
state = []
wait = []
s = 0
for i in range(10000):
    state.append(s)
    wait.append(rexp(d[s]))
    jumpprobs = p[s]
    s = randomselect(jumpprobs)
# function to work out the state at some arbitrary time t
def X(t):
    t0 = 0
    for s,w in zip(state,wait):
        t0 = t0+w
        if t0>=t: return s
    return None
#
print X(10)


#-------------------------------------------------------------------------

# Different code, using a generator rather than a fixed number of jumps
def markov_process(r):
    p = [[rij/sum(row) for rij in row] for row in r]
    d = [sum(row) for row in r]
    s,wait = 0,rexp(d[0])  # current state, and time left in current state
    t = 0                  # current time
    while True:
        rununtil = t+(yield s)
        while t+wait <= rununtil:
            t = t+wait
            s = randomselect(p[s])
            wait = rexp(d[s])
        wait = wait - (rununtil-t)
        t = rununtil

# Get the value of X at times [.5,1,1.5,2,2.5,3,3.5,4]
X = markov_process(r)
X.next()
print [X.send(.5) for i in range(8)]

# Get the value of X at times [1,2,3,...,100000] and measure how often it's in each state
import collections
visits = collections.defaultdict(int)
for i in range(100000): visits[X.send(1)] += 1
visits = [visits[i] for i in range(len(r))]
print [v/float(sum(visits)) for v in visits]
```

Section 2.5

Equilibrium distribution of a Markov process
Network Performance—DJW—2010/2011

1 Finding the equilibrium distribution

When we simulate a Markov process $(X_t, t \geq 0)$, we typically find that there is a vector π such that

- the fraction of time spent in state i approaches π_i, the longer we run the simulation i.e. π is *ergodic*
- if we interrupt the simulation and see what state it's in, the probability of finding it in state i approaches π_i the longer we run the simulation, i.e. π is *limiting*
- if we pick the initial state X_0 randomly with distribution π, then at any time t in the future, X_t has distribution π.

This distribution π is called the *equilibrium distribution.* To find it,

(i) Write out the *balance equations*, $\pi_i \sum_j r_{ij} = \sum_j \pi_j r_{ji}$, one equation for each i.
(ii) Write out the *normalization equation*, $\sum_i \pi_i = 1$.
(iii) Solve all these equations simultaneously.

2 Using a computer to find the equilibrium distribution

If we write out the balance equations in matrix form, we can solve them with a computer. First, let $r \in \mathbb{R}^{n \times n}$ be the rate matrix and define

$$d = \begin{pmatrix} \sum_j r_{1j} & 0 & \dots & 0 \\ 0 & \sum_j r_{2j} & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & \sum_j r_{nj} \end{pmatrix}.$$

The balance equations

$$\pi_i \sum_j r_{ij} = \sum_j \pi_j r_{ji}$$

can be written in matrix form as

$$\pi d = \pi r,$$

or equivalently $\pi(r-d) = 0$. We can use this to solve for π numerically. For example,

```
# load in the library for linear algebra
library(MASS)
# set up the rate matrix
r <- rbind(c(0,2,0),
           c(0,0,.8),
           c(1.1,1.3,0))
# calculate d
d <- diag(apply(r,1,sum))
# solve the balance equations
eqm <- Null(r-d)[,1]
# and rescale eqm so it sums to 1
eqm <- eqm/sum(eqm)
```

3 A shortcut that sometimes works

For some Markov processes, you can fairly easily find a distribution π such that

$$\pi_i r_{ij} = \pi_j r_{ji} \quad \text{for all } i \text{ and } j.$$

If this is so, then π automatically solves the balance equations.

4 Formal properties of Markov processes

Definition. A Markov process is called *irreducible* if for every pair of states i and j there is a path in the state space diagram from i to j. The path may have multiple steps.

Theorem. If a Markov process is irreducible, and if it has finitely many states, then it is possible to solve the balance and normalization equations, and this solution is unique. The solution is called the equilibrium distribution, usually written π. Furthermore,

- The equilibrium distribution is *invariant*, also known as *stationary*. That is, if $\mathbb{P}(X_t = j) = \pi_j$ for all j, then for any $s \geq 0$, $\mathbb{P}(X_{t+s} = j) = \pi_j$ for all j.
- The equilibrium distribution is *limiting*. That is, no matter what the distribution of X_0 is, $\mathbb{P}(X_t = j) \to \pi_j$ as $t \to \infty$, for all j.
- The equilibrium distribution is *ergodic*. That is, for every simulation we run, if we let $V_t(i)$ be the amount of time spent in state j over the interval $[0,t]$, then then $V_t(j)/t \to \pi_j$ as $t \to \infty$, for all j.

Theorem. If a Markov chain is irreducible and aperiodic, and it has infinitely many states, then it is always possible to solve the balance equations. It may or may not be possible to also solve the normalization equation.

- If the normalization equation can be solved, then π is invariant, limiting, and ergodic, as in the finite case. Also the process is *recurrent*, i.e. starting from any state j, $\mathbb{P}(\text{eventually returns to } j) = 1$. Furthermore it is *positive recurrent*, i.e. starting from any state j, $\mathbb{E}(\text{time to return to } j) < \infty$.
- If the normalization equation cannot be solved, then either
  (i) the process is *transient*, the opposite of recurrent, i.e. there is some state j such that, starting from state j, $\mathbb{P}(\text{eventually return to } j) < 1$; or
  (ii) the process is recurrent but not positive recurrent; or
  (iii) the process is *explosive*, i.e. $\mathbb{P}(\text{visits infinitely many states in finite time}) > 0$.
  In other words, either it might never return, or it always returns but it can take a very very long time, or it gets trapped in a 'black hole'.

OTHER CALCULATIONS WITH MARKOV PROCESSES

For Markov chains, we wrote out equations for hitting times. (see § 2.3)
For Markov processes, a similar technique applies:

Let t_i = average time, starting from i, until we hit some specific state e.

$$= \text{average wait in state } i + \sum_j \text{mean time to hit } e\text{, starting from } j\text{, if we jumped to } j$$

$$= \frac{1}{\sum_{j \in S} r_{ij}} + \sum_{j \in S} t_j \frac{r_{ij}}{\sum_{k \in S} r_{ik}}$$

For Markov chains, we wrote out equations for hitting probabilities (see § 2.3).
For Markov processes, the best way to find hitting probabilities is to look at the "jump chain".

Let $(X_t, t \geq 0)$ be a Markov process with rate matrix r.
Let $Y_0 = X_0$, and let Y_n be the state of the Markov process just after the nth jump. Then $(X_n, n \in \{0, 1, \cdots\})$ is a Markov chain. It is called the <u>jump chain</u>.

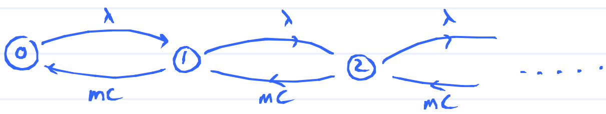
The hitting probabilities for X are exactly the same as the hitting probabilities for Y, since they both visit exactly the same states in the same order.

§ 2.6 Stability of Markov processes

When a Markov process has finitely many states, we can always solve the balance equations & the normalization equation, to calculate the equilibrium distribution.

But when it has infinitely many states, while we can always solve balance, we might not be able to solve normalization. If we can, we call it "stable". If we can't we call it "unstable". This notion of stability is very important in analysing networks (more important than performance measures like "mean delay").

Example: In §3 we will show that the Processor sharing Link can be modelled as a Markov process. Let X_t = # active flows at time t. Then $(X_t, t \geq 0)$ is a Markov process with state space diagram



where flow interarrival times are independent $\sim Exp(\lambda)$
flow sizes are independent $Exp(\frac{1}{m})$
link speed = c.

After writing out the state space diagram, our next step is to look for an eqm distn, ie solve balance, and try to solve normalization.

BALANCE EQUATIONS: $\pi_i \sum_j r_{ij} = \sum_j \pi_j r_{ji}$ for every state i

$$\left.\begin{aligned} \pi_0 \lambda &= \pi_1 mc \\ \pi_1 (\lambda + mc) &= \pi_0 \lambda + \pi_2 mc \\ \pi_2 (\lambda + mc) &= \pi_1 \lambda + \pi_3 mc \\ &\vdots \end{aligned}\right\} \text{ solution is } \pi_n = \left(\frac{\lambda}{mc}\right)^n \pi_0$$

NORMALIZATION EQUATION: $\sum_i \pi_i = 1$

$$\pi_0 + \pi_1 + \pi_2 + \cdots = 1$$

$$\Rightarrow \pi_0 \left(1 + \frac{\lambda}{mc} + \left(\frac{\lambda}{mc}\right)^2 + \cdots\right) = 1$$

Case 1: $\frac{\lambda}{mc} < 1$. Then $\pi_0 \frac{1}{1-\frac{\lambda}{mc}} = 1 \Rightarrow \pi_0 = 1 - \frac{\lambda}{mc}$.
Therefore the system is stable, eqm dist. is $\pi_n = \left(1 - \frac{\lambda}{mc}\right)\left(\frac{\lambda}{mc}\right)^n$.

Case 2: $\frac{\lambda}{mc} \geq 1$. Then $\pi_0 \, \infty = 1$, which is impossible to solve.
Therefore the system is unstable; there does not exist an equilibrium distribution.
Interpretation: more work arrives than can be served, and X_t drifts to infinity.

Exercise (1) Suppose $\frac{\lambda}{mc} < 1$. Write out the equations for hitting time, and show that, starting from state 0,

$$\mathbb{E}(\text{time to return to } 0) < \infty .$$

(2) Suppose $\frac{\lambda}{mc} = 1$. Show that, starting from state 0,

$$\mathbb{E}(\text{time to return to } 0) = \infty .$$

Also, write out the equations for hitting probability. Show that

$$\mathbb{P}(\text{hit } 0, \text{ starting from state } i) = 1, \quad \text{for every } i.$$

(3) Suppose $\frac{\lambda}{mc} > 1$. Show that, starting from 0,

$$\mathbb{P}(\text{return to } 0) < 1 .$$

§2·7 Properties of the Exponential.

In a Markov process, we assume that the wait times at any state are Exponential random variables. This assumption is necessary for the theorems about equilibrium distribution & stability to apply.

More generally, there are many network models where, as long as all the random variables are Exponential random variables, the model is mathematically tractable.

Even when the underlying random variables are not (or should not) be Exponential, it is still a useful exercise to calculate how the system would behave under Exponential random variables —

(1) it gives a useful back-of-the-envelope estimate, which says how parameters ought to relate to each other, and which can be compared to simulation

(2) in some lucky cases, it can (with sophisticated maths) be proved that the Exponential case and the non-Exponential case give identical answers.

(3) it is easier to simulate, and reason about, because memorylessness (see below) means you don't need to store much state.

1. MEMORYLESSNESS OF THE EXPONENTIAL

Let $X \sim \mathrm{Exp}(\lambda)$.

Then $\mathbb{P}(X \geq x + x_0 \mid X \geq x_0) = \mathbb{P}(X \geq x)$, for any $x_0 \geq 0$ and $x \geq 0$.

This is called <u>memorylessness</u>.

Interpretation:

Let X be the time until an event, e.g. a bus arrives. Suppose $X \sim \mathrm{Exp}(\lambda)$. Suppose we've already waited x_0 and the event hasn't happened yet. How much longer until it happens? In other words, what is the distribution of the <u>residual</u> $Y = X - x_0$, conditional on $X \geq x_0$?

The distribution function of Y is

$$F(y) = \mathbb{P}(Y \geq y \mid X \geq x_0) = \mathbb{P}(X - x_0 \geq y \mid X \geq x_0) = e^{-\lambda y}.$$

ie the residual waiting time is also $\mathrm{Exp}(\lambda)$.

2. MINIMUMS OF EXPONENTIALS

Exercise: Let $X_1 \sim Exp(\lambda_1)$, $X_2 \sim Exp(\lambda_2)$, $\ldots$, $X_n \sim Exp(\lambda_n)$ be independent.

(i) Show that $\min(X_1, \cdots, X_n) \sim Exp(\lambda_1 + \cdots + \lambda_n)$.

(ii) It can be shown that $\mathbb{P}(X_1 \leq X_2) = \frac{\lambda_1}{\lambda_1+\lambda_2}$. Use this fact to prove that

$$\mathbb{P}(\min(X_1,\cdots,X_n) = X_j) = \frac{\lambda_j}{\lambda_1+\cdots+\lambda_n}$$

We have seen one method for simulating a Markov process with rate matrix r and state space S:

METHOD 1.

1. Pick some arbitrary starting state I.
2. Generate an Exponential random variable $T \sim Exp\left(\sum_{j\in S} r_{Ij}\right)$
   Generate a random variable J, with range S,
   where $P(J=j) = r_{Ij} / \sum_{k\in S} r_{Ik}$
3. Wait in state I for time T, then jump to state J.
   Let $I \leftarrow J$, and go to step 2.

Here is an equivalent method. The exercise above proves that it yields the same distributions for T and J.

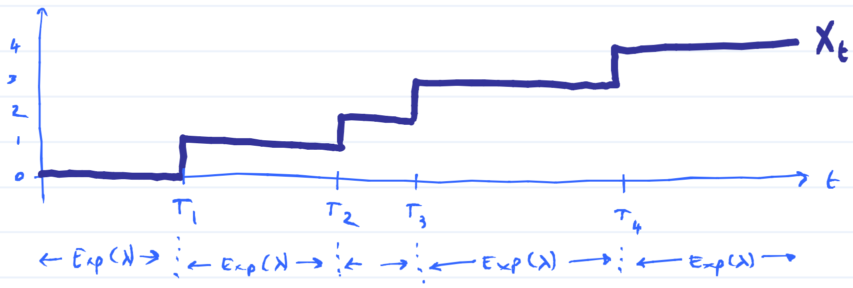
METHOD 2.

1. Pick some arbitrary starting state I.
2. Generate a collection of independent Exponential random variables, a T_j for each state j where there is an arrow from I to j.
   where $T_j \sim Exp(r_{Ij})$
   Let $T = \min_j T_j$, and let J be the index of the smallest.
3. Wait in state I for time T, then jump to state J.
   Let $I \leftarrow J$, and go to step 2.

§ 2.8 The Poisson process

A sequence of events is a Poisson process of rate λ if the first event time, and all inter-event times, are independent $\text{Exp}(\lambda)$ random variables.

We often represent it as the random process $(X_t, t \geq 0)$, where X_t = #events that have happened by time t.



It is appropriate to use a Poisson process for event times in systems which have a large number of independent agents, each of whom may trigger an event

e.g. call start times on a telephone network

e.g. web-browsing session start times

e.g. times when one of Napoleon's soldiers was kicked to death by a mule

e.g. packet arrivals at a big core Internet router

e.g. times when a chunk of radioactive matter emits a particle of radiation.

See Paxson + Floyd, "the failure of Poisson modelling", for the situations in which we may assume that arrivals are a Poisson process.

[NOT EXAMINABLE]

If $(X_t, t \geq 0)$ is a Poisson process, then

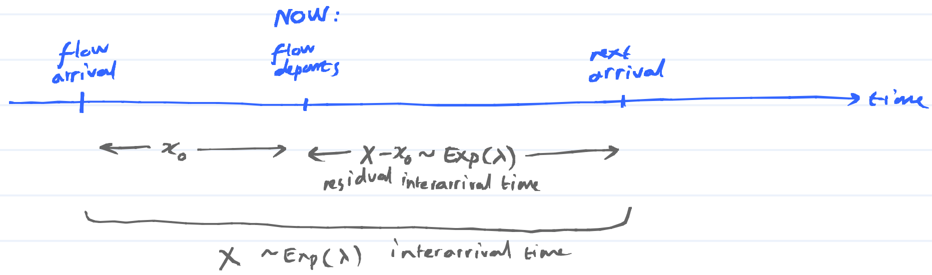
- it takes integer values, and it only ever increases
- it has independent increments, i.e. $X_{t+v} - X_t$ is independent of $X_{s+u} - X_s$ whenever the intervals $[t, t+v]$ and $[s, s+u]$ do not overlap.
- For small δ, $\mathbb{P}(X_{t+\delta} = X_t) = 1 - \delta\lambda + o(\delta)$
  $\mathbb{P}(X_{t+\delta} = X_t + 1) = \delta\lambda + o(\delta)$

In fact, the Poisson process is the ONLY process that satisfies these three properties.

MEMORYLESSNESS OF THE POISSON PROCESS

If I turn up at some arbitrary point in time, the wait until the next event is $\text{Exp}(\lambda)$, by the memorylessness property of the Exponential distribution.

Example: In the processor sharing model that we simulated, the interarrival times were $\text{Exp}(\lambda)$, ie the arrival process was a Poisson process of rate λ.
By memorylessness, from any instant in time, the residual wait until the next arrival is also $\text{Exp}(\lambda)$.
In particular, whenever #active flows changes (either due to an arrival or a departure), we can "reset" the interarrival timer.



OTHER PROPERTIES

Inter-event times are $\sim \text{Exp}(\lambda)$, so mean interevent time is $\frac{1}{\lambda}$, so mean time for n events is n/λ, so the average rate at which events occur is $\frac{n}{n/\lambda} = \lambda$ events per time unit.
This is why λ is called the "rate".

The rate parameter λ has units time^{-1}, e.g. sec^{-1} or minute^{-1}.
Changing time units corresponds to multiplying or dividing λ.
For example, let $(X_t, t \geq 0)$ be a Poisson process of rate $\lambda = 0.5/\text{sec}$.
Let $Y_u = X_{60u}$, ie Y_u = number of events in u minutes.
If the event times for X are $T_1, T_2, \dots$
then the event times for Y are $T_1/60, T_2/60, \dots$
So the inter-event times for Y are 60 times smaller, so they are $\text{Exp}(60 \times 0.5)$.
(see calculation in §1.6a, about rescaling an Exp. random variable.)
So Y is a Poisson process of rate $30/\text{min}$.

If flies arrive as a Poisson process of rate λ, and each fly has probability p of landing in my soup, then the arrivals 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 as a Poisson process of rate μ, then the arrivals of insects is a Poisson process of rate $\lambda + \mu$. This is called "superposition".

For any given t, $P(X_t = r) = e^{-\lambda t} (\lambda t)^r / r!$
This is called the Poisson distribution.
Do not confuse the Poisson process — the entire process $(X_t, t \geq 0)$
with the Poisson distribution — the value of X_t at some specific t.

§3 Markov job models

A *job* represents a task which arrives at a system, takes up capacity for some time, then departs

- a TCP flow
- a person queueing at the supermarket
- a web-browsing session

We're interested in knowing e.g. average waiting time, # jobs in the system, whether the system is overloaded.

When job arrivals form a Poisson process, we can often represent the system as a Markov process; then we can use the powerful techniques for Markov processes to calculate the performance of the system. The general approach is

(1) Write out the state space, ie the set of all possible states, plus arrows showing which transitions are possible
(2) Work out the rates for all these transitions
(3) calculate the equilibrium distribution.

§3.1 Processor-sharing model of TCP

Suppose that users want to transfer files over a link of capacity C Mb/s using TCP.
These requests arrive, remain active for a time, then depart once the file is transferred.
Let N_t = # active connections at time t.
TCP aims to achieve "fair sharing", ie each file should get throughput C/N_t Mb/s.

Suppose that file transfer requests arise as a Poisson process of rate λ files/s,
and that file sizes are $\sim \mathrm{Exp}(\frac{1}{m})$ Mb/s.

What is the mean time to transmit a file?
How busy is the system, ie what is the utilization?
How many jobs are there in progress, on average?

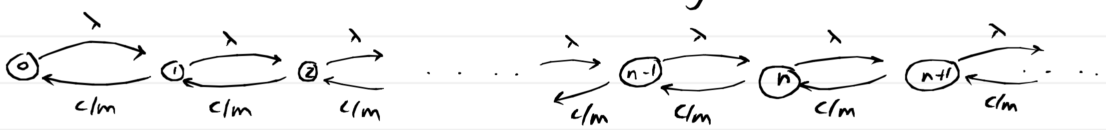
(1) Find the state space and the possible transitions.

N_t can take any value in $\{0,1,2,\dots\}$.
N_t can increase by 1 (when a new flow starts) or decrease by 1 (when an existing flow ends).
It could be that two flows start simultaneously, or end simultaneously, but we'll split those into two steps. Technically, $\mathbb{P}$(two events happen simultaneously) = 0 here, so we don't need to include it.

(2) Work out the transition rates.

Suppose we turn up at time t, and see N_t has just arrived at $N_t = n$.
How long until N_t changes? We'll show later that

- the time until the next arrival is $\sim \mathrm{Exp}(\lambda)$
- the time until the next call leaves, assuming no arrivals, is $\sim \mathrm{Exp}(C/m)$



(3) Calculate the equilibrium distribution.

The balance equations are: $\left(\pi_i \sum_j r_{ij} = \sum_j \pi_j r_{ji} \quad \text{for all } i\right)$

$$\pi_n\left(\lambda + \frac{c}{m}\right) = \pi_{n-1}\lambda + \pi_{n+1}\frac{c}{m} \qquad \text{for all } n$$

It's always good to try to solve detailed balance first: $\left(\pi_i r_{ij} = \pi_j r_{ji} \quad \text{for all } i \text{ and } j\right)$

$$\left.\begin{aligned} \pi_n \lambda &= \pi_{n+1} c/m && (m=n+1) \\ \pi_n c/m &= \pi_{n-1}\lambda && (m=n-1) \\ \pi_n 0 &= \pi_m 0 && (\text{otherwise}) \end{aligned}\right\}$$

Or, more succinctly,

$$\pi_n = \pi_{n-1} \times \frac{\lambda m}{c}$$

Hence $\pi_1 = \pi_0 \times \frac{\lambda m}{c}$, $\pi_2 = \pi_1 \times \frac{\lambda m}{c} = \pi_0 \times \left(\frac{\lambda m}{c}\right)^2$, $\pi_3 = \pi_0 \times \left(\frac{\lambda m}{c}\right)^3$, etc.

The general solution is $\pi_n = \pi_0 \rho^n$ where $\rho = \frac{\lambda m}{c}$.

Next, try to solve the normalization equation, $\left(\sum_i \pi_i = 1\right)$

$$\pi_0\left(1+\rho+\rho^2+\cdots\right) = 1.$$

- If $\rho \geq 1$, then $1+\rho+\rho^2+\cdots = \infty$, and it is impossible to pick π_0 to solve the normalization equation. Hence the system is unstable. (see §2.6)
- If $\rho < 1$, then $1+\rho+\rho^2+\cdots = \frac{1}{1-\rho}$, so we set $\pi_0 = 1-\rho$.

The system is stable, and the equilibrium distribution is $\pi_n = (1-\rho)\rho^n$.

(4) Draw conclusions.

If $\rho<1$ the system is stable, and the equilibrium distribution is

$$\mathbb{P}(\#\text{active flows} = n) = (1-\rho)\rho^n$$

This distribution is limiting, invariant, and ergodic (see §2).

It's a good idea to write the distribution in a standard form.

From the random variable reference sheet:

if $X \sim \text{Geom}(p)$ then $\mathbb{P}(X=r) = (1-p)^{r-1}p$.

To match this up with the formula for the equilibrium distribution, rewrite:

$$\mathbb{P}(\#\text{active flows} = n-1) = \rho^{n-1}(1-\rho)$$

$$\Rightarrow \quad \mathbb{P}(\#\text{active flows}+1 = n) = \rho^{n-1}(1-\rho)$$

$$\Rightarrow \quad \#\text{active flows}+1 \sim \text{Geom}(1-\rho)$$

$$\Rightarrow \quad \#\text{active flows} \sim \text{Geom}(1-\rho) - 1.$$

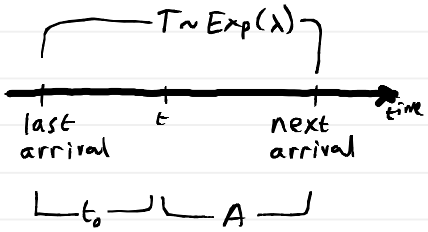
The average number of active flows is

$$\mathbb{E}(\#\text{active flows}) = \mathbb{E}\left[\text{Geom}(1-\rho)-1\right] = \left[\mathbb{E}\,\text{Geom}(1-\rho)\right]-1 = \frac{1}{1-\rho}-1 = \frac{\rho}{1-\rho}.$$

JUSTIFICATION OF TRANSITION RATES FOR PROCESSOR SHARING

Q1. Suppose we turn up at some time t and wait for the next arrival. How soon is it?

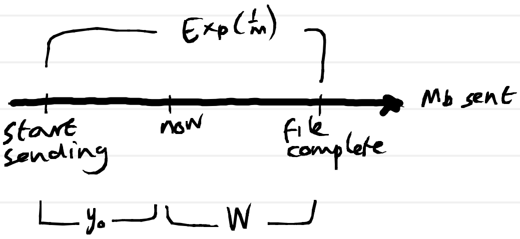
Let it be t_0 since the last arrival, and let T be the interarrival time; we know $T \geq t_0$.
Arrivals form a Poisson process of rate λ, so $T \sim \text{Exp}(\lambda)$.
Let A be the time until the next arrival, $A = T - t_0$. We want to know: what's the distn. of A?



$\mathbb{P}(A \geq a) = \mathbb{P}(T - t_0 \geq a \mid T \geq t_0) = e^{-\lambda a}$.
So $A \sim \text{Exp}(\lambda)$.

This property is called the "memoryless property of the exponential distribution". In sloppy words: you're waiting for an $\text{Exp}(\lambda)$ timer to go off, then no matter how long you've been waiting you still have $\text{Exp}(\lambda)$ left to go.

Q2. Suppose a flow is still being transferred, after having already transferred y_0 Mb. How much is left to go?



The question says that file sizes are $\text{Exp}(\frac{1}{m})$.
By the memoryless property of the exponential distribution, the amount still waiting to be sent, W, is also $\text{Exp}(\frac{1}{m})$.

Q3. Let $W_1, \cdots, W_n$ be the residual file sizes of the n active flows.
How long until one of them ends, assuming no new arrivals?

The smallest residual file size is $\min(W_1, \cdots, W_n) \sim \text{Exp}(n \times \frac{1}{m})$

Each active flow gets throughput c/n.
So the time until the earliest departure is

$$\frac{\min(W_1, \cdots, W_n)}{c/n} \sim \frac{\text{Exp}(n \times \frac{1}{m})}{c/n} \sim \text{Exp}\left(\frac{c}{n} \times n \times \frac{1}{m}\right) = \text{Exp}\left(\frac{c}{m}\right).$$

See §2.7 for minimums of exponentials.
See §1.6a for scaled exponential.

§3.2 The Erlang Link

Suppose a network operator has a link with C circuits (i.e. it can accomodate C simultaneous telephone calls).
Let N_t = number of circuits in use at time t.
Suppose calls arrive as a Poisson process of rate λ, and call holding times are $\sim \text{Exp}(\mu)$.

How big does C need to be to ensure that the blocking probability (i.e. the probability that an incoming call has to be dropped) is small?

The theory of Markov processes was begun by the Danish mathematician Erlang in ≈ 1910, while working for the Copenhagen Telephone Company. He devised the theory to answer precisely this question.

(1) Find the state space

N_t can take any value in $\{0, 1, \dots, C\}$
It can increase by 1 (when a new call starts) or decrease by 1 (when a call finishes).

(2) Work out the transition rates.

Suppose we turn up at time t, and see N_t has just arrived at $N_t = n$.
How long until N_t changes? We'll show that

- the time until the next arrival is $\sim \text{Exp}(\lambda)$
- the time until the next call leaves, assuming no arrivals, is $\sim \text{Exp}(n\mu)$

(3) Calculate the equilibrium distribution

Like the processor-sharing model, we'll solve the detailed balance equations

$$\pi_{n-1} \lambda = \pi_n \cdot n\mu$$

$$\Rightarrow \quad \pi_n = \frac{\lambda^n}{n!\,\mu^n} \pi_0 = \frac{\rho^n}{n!} \pi_0, \quad \text{where } \rho = \frac{\lambda}{\mu}. \quad = \text{"traffic load, in Erlangs"}$$

To normalize it so $\sum_{n=0}^{C} \pi_n = 1$, set $\pi_0 = \dfrac{1}{\sum_{n=0}^{C} \rho^n / n!}$

(4) Draw conclusions

New calls are blocked whenever the system is full, ie $N_t = C$.
The long-run fraction of time spent in this state is π_C.
Thus, the <u>blocking probability</u> is $\pi_C = \dfrac{\rho^C/C!}{\sum_{n=0}^{C} \rho^n/n!}$, also written $E(\rho, C)$.

NOTE: if you try to compute this naively, and $\rho > 1$ and C is large, you'll run into problems with numerical overflow. To avoid this, note that

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

You can probably find library functions for $\mathbb{P}(X=n)$ and $\mathbb{P}(X\le C)$:

```
# R has built-in functions for Prob(X=C) and Prob(X<=C) that we can use.
erlang <- function(rho,C) dpois(C,rho)/ppois(C,rho)
```

```
import scipy.special
# pdtr(k,m) gives Prob(X<=k) where X~Poisson(m)
def dpois(C,rho): return scipy.special.pdtr(C,rho)-scipy.special.pdtr(C-1,rho)
def erlang(rho,C): return dpois(C,rho)/scipy.special.pdtr(C,rho)
```

JUSTIFICATION OF TRANSITION RATES FOR ERLANG LINK.

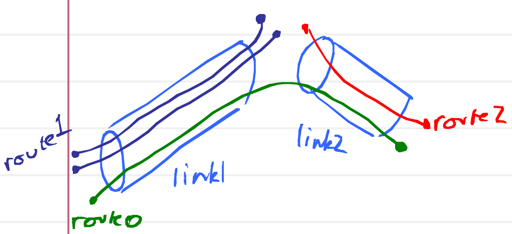
Q1. Suppose a call is still active, after having been in the system for time t_0. How long does it have left to go?
Let Y be the lifetime of the call; we know $Y \sim \text{Exp}(\mu)$, and $Y \geq t_0$.
By the memoryless property of the exponential distribution, the residual duration is also $\text{Exp}(\mu)$.

Q2. Suppose I turn up at some time t, and see there are $N_t = n$ calls active. How long until one of them finishes?
Let their residual durations be $X_1, \dots, X_n$.
We want to know the distribution of $Z = \min(X_1, \cdots, X_n)$.
We know $X_i \sim \text{Exp}(\mu)$, and we assume they're independent; hence $Z \sim \text{Exp}(n\mu)$.

Q3. Suppose we turn up at some time t and wait for the next arrival. How soon is it?
It's $\sim \text{Exp}(\lambda)$, just like it was for processor sharing.

§3.3 Network examples

EXAMPLE 1: LOSS NETWORKS (a generalization of the Erlang link).



Consider a network with two links:
link 1 can hold C_1 circuits, link 2 can hold C_2.
Suppose there are three routes through the network:

- route 0 calls occupy a circuit at each of the links
- route 1 calls only use link 1
- route 2 calls only use link 2.

Suppose that route i calls arrive as a Poisson process of rate λ_i, and that their durations are $Exp(\mu_i)$.

Let $(N_0(t), N_1(t), N_2(t))$ be the number of calls on each route at time t.
The state space is $S = \{ (n_0, n_1, n_2) : n_0 + n_1 \le C_1 \text{ and } n_0 + n_2 \le C_2 \}$
The transition rates are

From (n_0, n_1, n_2):

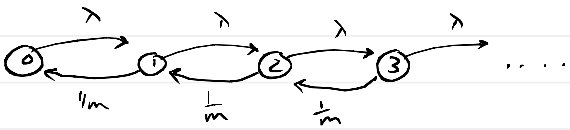
- to (n_0-1, n_1, n_2): rate $n_0\mu_0$
- to (n_0, n_1-1, n_2): rate $n_1\mu_1$
- to (n_0, n_1, n_2-1): rate $n_2\mu_2$
- to (n_0+1, n_1, n_2): rate $\begin{cases} \lambda_0 & \text{if } n_0+1+n_1 \le C_1 \text{ and } n_0+1+n_2 \le C_2, \\ 0 & \text{otherwise} \end{cases}$
- to (n_0, n_1+1, n_2): rate $\begin{cases} \lambda_1 & \text{if } n_0+n_1+1 \le C_1 \\ 0 & \text{otherwise} \end{cases}$
- to (n_0, n_1, n_2+1): rate $\begin{cases} \lambda_2 & \text{if } n_0+n_2+1 \le C_2 \\ 0 & \text{otherwise} \end{cases}$

EXAMPLE 2: MULTICLASS SYSTEMS

See Exercise Sheet 5, question 4.

§3.4 FIFO Queue

Suppose that jobs arrive to a queue, and jobs are served in the order they arrive. Suppose that jobs arrive as a Poisson process of rate λ jobs/sec, and that job service times are exponential with mean m sec.
Let Q_t = #jobs in the queue (including the one being served).



Exercise. Justify these transition rates.
Let $\rho = \lambda m$. Supposing that $\rho < 1$, show that

$$\mathbb{P}(Q_t = q) = (1-\rho)\rho^q, \qquad \mathbb{P}(Q_t \geq q) = \rho^q, \qquad \mathbb{E}Q_t = \frac{\rho}{1-\rho}.$$

Now suppose that there is a limited buffer, of size b, and when full then incoming jobs are dropped.
Show that

$$\mathbb{P}(Q_t = b) = \frac{(1-\rho)\rho^b}{1-\rho^{b+1}}$$

§3.5 Kendall notation

arrival process / service time for a job / number of servers, ie. number of jobs that can be served simultaneously at their peak rate / buffer size

arrival process:

- M : Poisson arrivals
- GI : independent interarrival times, but with an arbitrary distribution
- • : arbitrary arrivals

service time for a job:

- M : exponential
- D : deterministic, ie constant
- GI : arbitrary distribution, independent.

buffer size: this is omitted if there is an infinite buffer

e.g. TCP processor sharing :	$M_\lambda / M_{C/m} / 1 - PS$ — to say that the service discipline is processor sharing;
Erlang link:	$M_\lambda / M_\mu / C / C$
FIFO queue:	$M_\lambda / M_{1/m} / 1$
FIFO queue with buffer b:	$M_\lambda / M_{1/m} / 1 / b$
A pipe, ie a device which delays every packet by a fixed amount	$\bullet / D / \infty$

§3·6 The PASTA Property

I have glossed over a subtle problem.....
In e.g. the Erlang link, there are two different ways to measure blocking probability:

(1) Every time a call arrives, note down what the current state of the system is. Measure the long-run fraction of calls that find the system full, ie the fraction of calls that get blocked.

This corresponds to an intuitive idea: the blocking probability is the fraction of calls that get blocked.

(2) Each time the system transitions, record what state it was in and how long it spent there. Measure the long-run fraction of time (the "time-average") that the system spends full.

we calculated the equilibrium distribution of the Markov process. We know that the equilibrium tells us about time-averages (it's ergodic).

Theorem Suppose that arrivals to a system form a Poisson process. The long-run fraction of arrivals that find the system in state n is equal to the long-run fraction of time the system spends in state n, for all states n.

If we have reason to believe arrivals actually are Poisson, we can use this result to our advantage:
Often, in simulations, it's easier to get a good estimate of time-averages than to get a good estimate of on-arrival averages. The theorem tells us they both give the same answer.

In queues and similar systems, we typically find that "burstier-than-Poisson" arrivals leads to bigger average queue sizes.

Exercise. See Exercise sheet 5 q 7 for a question about a processor-sharing link with bursty arrivals.

§3.7 Symmetric Queues

For some systems, the queue size distribution is the same whatever the service time distribution. They are called "symmetric queues". The ones you will encounter are

- M/M/1-PS, ie processor sharing model for TCP
- M/M/C/C ie Erlang link
- M/M/1-LCFS, ie a queue which devotes all its service to the last job to arrive (and which interrupts whatever else it is doing)

For example, consider a link shared by TCP flows, where file sizes have a Pareto distribution with mean m, arrivals are Poisson of rate λ, link has capacity C. This is an M_λ / Pareto / 1-PS queue.
We know that the queue size distribution is exactly the same as for an $M_\lambda / M_{C/m} / 1\text{-PS}$ queue, ie as if arrivals were $\text{Exp}(\frac{1}{m})$, and we worked it out in §3.1.

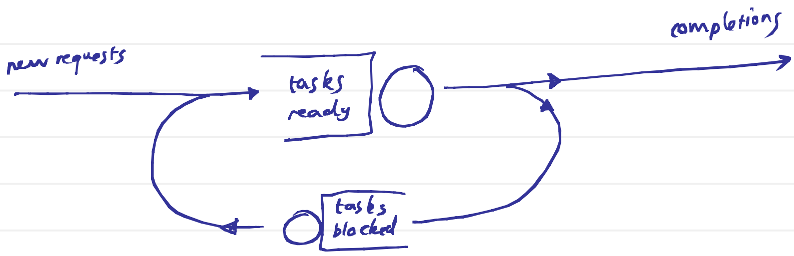
§ 3.8 Networks of queues

Here is an example:

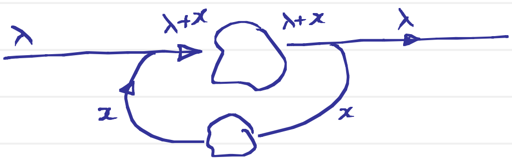
Question 10. Here is a model for a web server with active server pages, i.e. pages that cannot be served directly from the disk but instead require processing e.g. in PHP.

Suppose requests arrive at rate λ. Upon arrival they are placed in a 'task ready' queue, where they wait for the next available worker thread. The server has m worker threads. The CPU can execute c instructions per second, and when there are M threads active then each executes c/M instructions per second. When a thread becomes free, it starts work on the next task in the 'task ready' queue. When a thread is working on a task, it executes an average of i instructions, and then either it completes or it blocks, e.g. to wait for I/O. On average, each request will block b times before completing. If the task blocks, the thread is freed and the task is placed in a 'task blocked' pool. Each blocked task waits for an average of t seconds to unblock, and then it is placed in the 'task ready' queue.

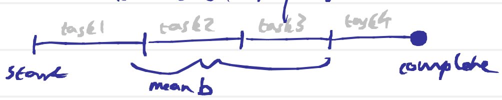
What is the maximum rate at which this web server can serve requests? What is the average request completion time?



STEP 1: work out the flow rates, under the assumption that the system is stable (ie none of the queues are filling up)



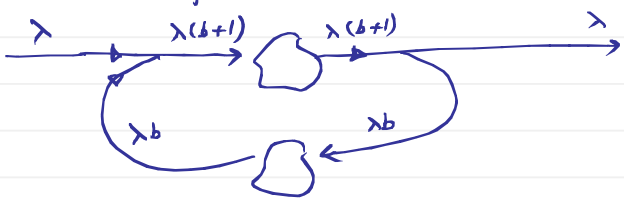
We're told that each request blocks on average b times before completing.



So, the fraction of tasks that end blocking is $\frac{b}{b+1}$.

So $\quad \dfrac{x}{\lambda+x} = \dfrac{b}{b+1} \quad \Rightarrow \quad x = \lambda b.$

So the overall flow rates are



STEP 2. analyse each component on its own.
Pretend that arrivals are Poisson ...

The 'tasks ready' queue is an $M_{\lambda(b+1)} / M_{C/i} / 1$ queue, with a particular service discipline given by m, the number of threads.
We could draw a state space diagram for it, and learn stability condition, average queue size, average delay etc. (see §4 for more.)
It is in fact stable if $\lambda (b+1) i < C$.

The 'tasks blocked' queue is an $M_{\lambda b} / G / \infty$ queue.
This is always stable, and obviously the mean waiting time is t.

STEP 3. Find the bottleneck, and other quantities.

The only bottleneck here is the CPU. It is only stable if $\lambda (b+1) i < C$.
Thus, the maximum rate at which the server can handle requests is $\dfrac{C}{(b+1) i}$.

Suppose we have calculated the average queueing delay at the 'tasks ready' queue, and found it to be d.
Then the average request completion time is

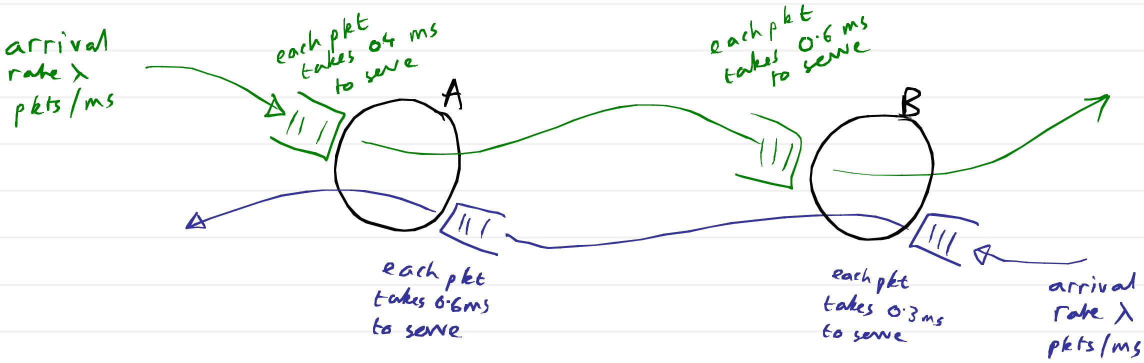
$$\underbrace{d\,(b+1)}_{\text{split into } b+1 \text{ tasks, each of which takes time } d} + \underbrace{t\,b}_{\text{blocks } b \text{ times, for duration } t}.$$

Theorem NOT EXAMINABLE

The above analysis is valid if
- external arrivals are Poisson
- routing decisions are independent
- each individual queue is "quasi-reversible",
  e.g. $M/M/1$, $M/M/C/C$, $M/M/1\text{-PS}$.
  More generally, if the queue has a statespace diagram in which the only permitted jumps are $+1$ or -1, and EITHER it is symmetric OR service times are exponential, then it is quasi-reversible.

NON-EXAMINABLE

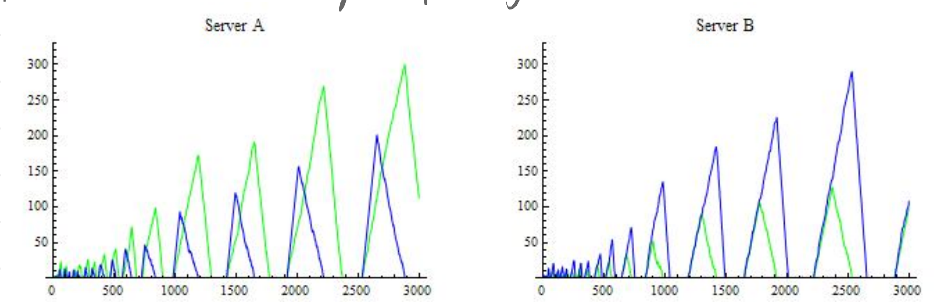
Here is an interesting example due to Rybko + Stolyar (1992) which shows that when the queues are not quasi-reversible, strange things can happen.



We'd expect A to be stable if tot arr. rate × mean job duration < 1 (see § 3.4)
ie if $2\lambda \times 0.5 < 1 \iff \lambda < 1$. same for B.

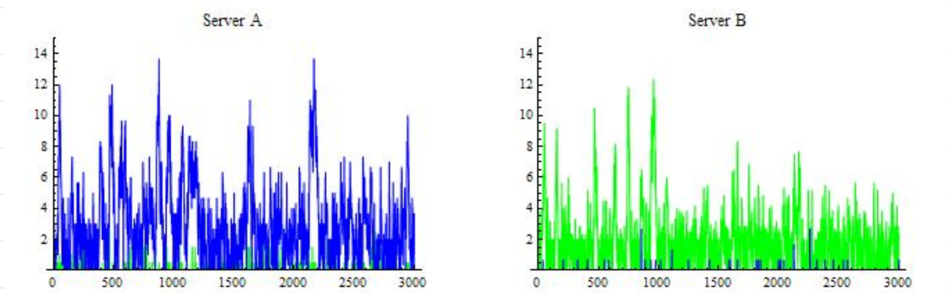
In fact, stability depends on the scheduling rule.
Here are two simulations, for exactly the same arrival pattern, at $\lambda = 0.9$.
Note the very different scales on the y-axis.

If servers A and B give priority to internal arrivals,



in fact, this is only stable when $\lambda < \frac{1}{1.2} = 0.8333\ldots$

If servers A and B give priority to external arrivals,



in fact, this is stable whenever $\lambda < 1$.

§4 Tools for analysing random processes

In this section we will study techniques for analysing random processes, e.g. the job models of §3.

We will learn three approximation techniques:

- drift models
- fixed point analysis
- operational laws.

These three tools are useful for situations where it is too hard to calculate the equilibrium — eg any non-trivial network. In networks the state space is usually enormous ("state space explosion").

They are also useful when the system is not exactly Markov, e.g. when arrivals are not Poisson.

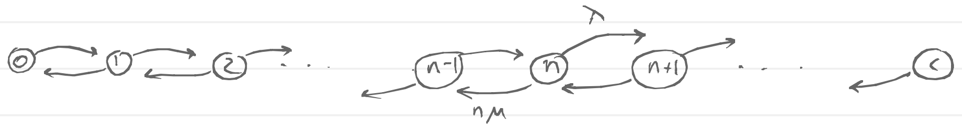
§4·1 Drift Models

The *drift* is the expected rate of change in a quantity.
A *drift diagram* uses arrows to show the direction & strength of drift.
A *drift model* is a deterministic approximation, based on following the drift.

The drift diagram shows us at a glance a qualitative picture of how the entire system behaves. The drift model is a quick and rough approximation, much faster than simulating the real system.

"A study of networks simulation efficiency: Fluid simulation vs. packet-level simulation"
Liu, Figueiredo, Guo, Kurose, Towsley, INFOCOM 2001.
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.3249

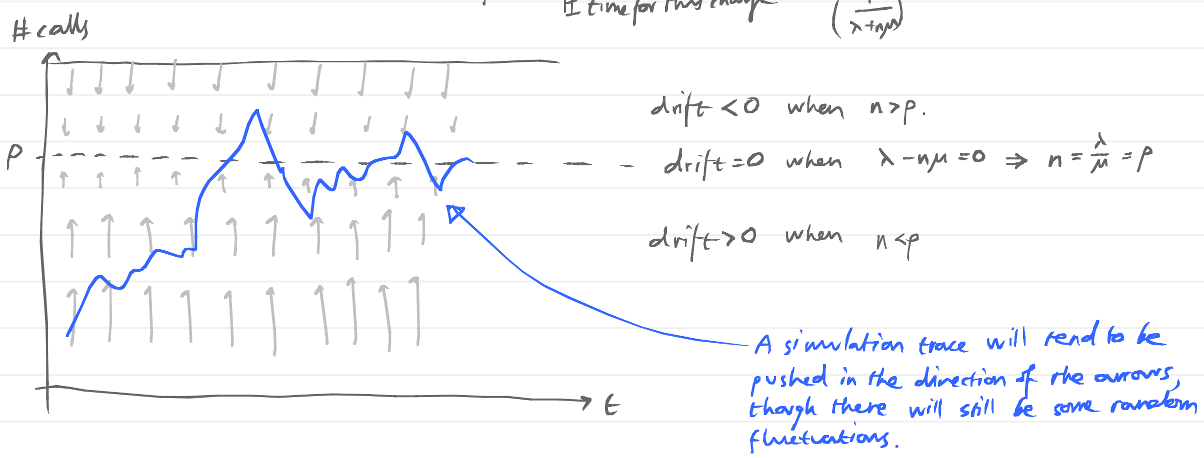
Example: Erlang link



From state n:
$$\mathbb{P}(\text{jump up by } 1) = \frac{\lambda}{\lambda+n\mu}.$$
$$\mathbb{P}(\text{jump down by } 1) = \frac{n\mu}{\lambda+n\mu}$$
$$\mathbb{E}(\text{change in \# calls}) = \frac{\lambda}{\lambda+n\mu}\times 1 + \frac{n\mu}{\lambda+n\mu}\times(-1) = \frac{\lambda-n\mu}{\lambda+n\mu}.$$

$$\mathbb{E}(\text{time spent in } n\text{, on this visit}) = \frac{1}{\lambda+n\mu}$$

Thus, $$\text{drift} = \frac{\mathbb{E}\,\text{change in \# calls}}{\mathbb{E}\,\text{time for this change}} = \frac{\left(\frac{\lambda-n\mu}{\lambda+n\mu}\right)}{\left(\frac{1}{\lambda+n\mu}\right)} = \lambda - n\mu.$$



Here, the state $n=\rho$ is called a *fixed point*, because drift=0.
It is called a *stable fixed point* because the arrows push you back towards $n=\rho$ after any fluctuation.

DRIFT MODEL

The drift equation tells us the <u>expected</u> rate of change in n_t.
If we pretend that n_t is non-random, and that its rate of change is given <u>exactly</u> by the drift equation, then we can use a computer to see how n_t evolves over time.

We will pretend that n_t changes smoothly with time. How much does it change over a short interval of time, say from time t to $t+\delta$?

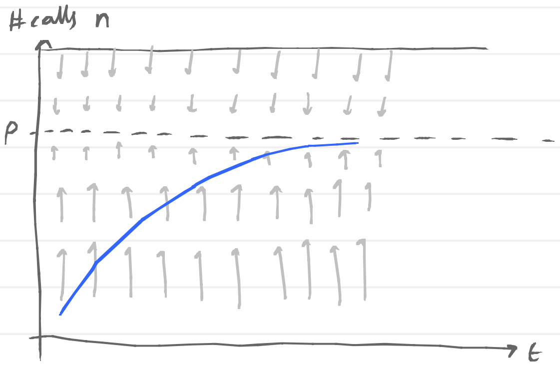
$$\frac{n_{t+\delta} - n_t}{\delta} = \frac{\text{change in } n_t}{\text{time for this change}} = \text{drift.}$$

Rearranging,

$$n_{t+\delta} = n_t + \delta \times \text{drift.}$$

	A	B	C
	time	n	drift
1	0	0	$= \lambda - B1 \times \mu$
2	0·1	$= B1 + (A2-A1) \times C1$	$= \lambda - B2 \times \mu$
3	0·2	$= B2 + (A3-A2) \times C2$	$= \lambda - B3 \times \mu$
	⋮	⋮	⋮

simple Excel simulator of the drift model — much faster than a full-blown packet-level simulation.



If we plot n from the Excel simulator, we typically see smooth convergence to a fixed point. Sometimes the plot jumps all over the place; this probably means δ is ~~too~~ big.

Other languages, e.g. R, have methods for adapting δ automatically, to this problem. See handout.

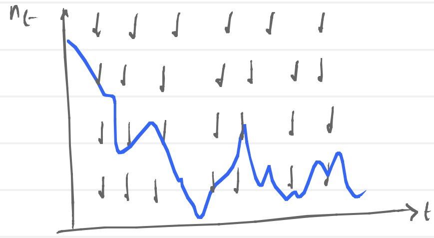
Mathematicians write "drift in n_t" as $\frac{dn_t}{dt}$. For example, for this Erlang link, we write $\frac{dn_t}{dt} = \lambda - n_t\mu$

and mathematicians can solve some of these problems exactly, without needing a computer. Here, $n_t = \frac{\lambda}{\mu} + (n_0 - \frac{\lambda}{\mu})e^{-\mu t}$.

Example: Processor sharing

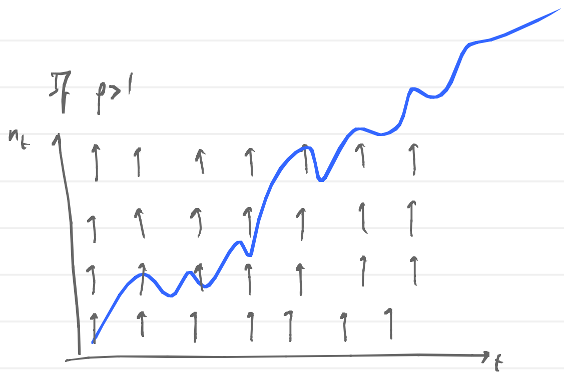
drift from $n = \dfrac{\mathbb{P}(\text{jump by } +1) \times 1 + \mathbb{P}(\text{jump by } -1) \times (-1)}{\mathbb{E}\text{ time to jump from } n} = \dfrac{\frac{\lambda}{\lambda + c/m} - \frac{c/m}{\lambda + c/m}}{\frac{1}{\lambda + c/m}} = \lambda - \frac{c}{m} = \frac{c}{m}(\rho - 1)$.

If $\rho < 1$:



Fixed point: $n_t = 0$

If $\rho > 1$



No fixed point; unstable.

Why is there a fixed point at 0?
Drift at $n > 0$ is $\lambda - \frac{c}{m} < 0$.
Drift at $n = 0$ is $\lambda > 0$.
The actual system cannot go below $n_t = 0$, so it will "jitter" ie bounce up and down very close to $n_t = 0$. To understand exactly what's going on in a jittering system, use a Markov model as in §3.1.

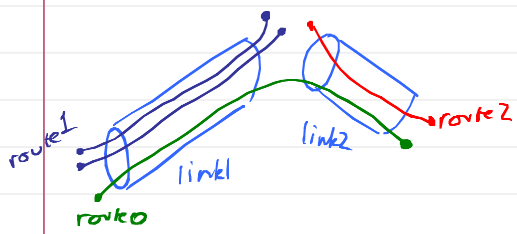
To determine whether or not a system is stable, we can look at the drift diagram. If there is some finite region such that all drift model trajectories head towards this finite region, then it is stable. Otherwise it is unstable.

"Stability of Fluid and Stochastic Processing Networks", Dai, 1999.

§4.2 Erlang fixed point

The general idea of the fixed point method is this:
write down a set of equations, one for each part of the system, and in each equation treating the rest of the system as given. Now, solve all the equations simultaneously.

Here we use the fixed point method to analyze a loss network.
In §5 we'll use it for TCP.



Suppose links 1 and 2 have capacities C_1 and C_2.
Arrival rates on the three links are $\lambda_0, \lambda_1, \lambda_2$.
Call durations are all Exponential distributions with mean $\frac{1}{\mu}$.

Consider link 1 first.
It's an Erlang link with a certain arrival rate, and C_1 circuits: thus

$$\mathbb{P}(\text{link 1 has all } C_1 \text{ circuits busy}) = E\left(\begin{smallmatrix}\text{tot. arrival rate}\\ \text{to link 1}\end{smallmatrix} / \mu,\ C_1\right).$$

Call this B_1.
The total arrival rate of calls which want to use link 1 is

$$\lambda_1 + \underbrace{\lambda_0(1-B_2)}$$

direct traffic on route 1

there are λ_0 calls/sec for route 0, but a fraction B_2 of them are blocked because link 2 is busy.

Thus $$B_1 = E\left(\frac{1}{\mu}[\lambda_1 + \lambda_0(1-B_2)],\ C_1\right)$$
Similarly $$B_2 = E\left(\frac{1}{\mu}[\lambda_2 + \lambda_0(1-B_1)],\ C_2\right)$$

Now we have to solve these two equations.

ITERATIVE METHOD FOR SOLVING FIXED-POINT EQUATIONS

Start at some arbitrary guess, e.g. $B_1^0 = \frac{1}{2}$, $B_2^0 = \frac{1}{2}$. Update these guesses by

$$B_1^{n+1} = E\left(\tfrac{1}{\mu}\left[\lambda_1 + \lambda_0(1-B_2^n)\right], c_1\right)$$

$$B_2^{n+1} = E\left(\tfrac{1}{\mu}\left[\lambda_2 + \lambda_0(1-B_1^{n+1})\right], c_2\right)$$

The point (B_1^n, B_2^n) should (hopefully) settle down as $n \to \infty$, to some (B_1^∞, B_2^∞), which satisfies the simultaneous equations. In practice, we keep iterating until there is little further change between successive iterations.

```
variables = initial guesses
while True:
    oldvariables = variables
    for i in range(len(equations)):
        update variable i using equation i
    delta = max(abs(variables-oldvariables))
    if delta very small: break
```

It is an active area of research, studying the accuracy of the fixed point approximation, and the convergence of this numerical method

It may be that your estimates (B_1^n, B_2^n) don't settle down.
If so, it's a good idea to try to use "baby steps":

$$B_1^{n+1} = (1-\varepsilon)\, B_1^n + \varepsilon\, E\left(\tfrac{1}{\mu}\left[\lambda_1 + \lambda_0(1-B_2^n)\right], c_1\right)$$

$$B_2^{n+1} = (1-\varepsilon)\, B_2^n + \varepsilon\, E\left(\tfrac{1}{\mu}\left[\lambda_2 + \lambda_0(1-B_1^{n+1})\right], c_2\right).$$

Choose $\varepsilon \in (0,1]$ small enough, and the procedure is likely to settle down. Choose ε too small, and it'll take ages to settle down.

We can think of the update equations as a drift model:

$$B_1^{new} = E\left(\tfrac{1}{\mu}\left[\lambda_1 + \lambda_0(1-B_2)\right],\ C_1\right)$$
$$B_2^{new} = E\left(\tfrac{1}{\mu}\left[\lambda_2 + \lambda_0(1-B_1^{new})\right],\ C_2\right)$$

$$\Rightarrow \quad \text{drift in } B_1 = \frac{\text{change in } B_1}{\text{time to change}} = \frac{B_1^{new} - B_1}{1} = E\left(\tfrac{1}{\mu}\left[\lambda_1+\lambda_0(1-B_2)\right], C_1\right) - B_1$$

$$\text{drift in } B_2 = \frac{\text{change in } B_2}{\text{time to change}} = \frac{B_2^{new} - B_2}{1} = E\left(\tfrac{1}{\mu}\left[\lambda_2+\lambda_0(1-B_1^{new})\right], C_2\right) - B_2$$

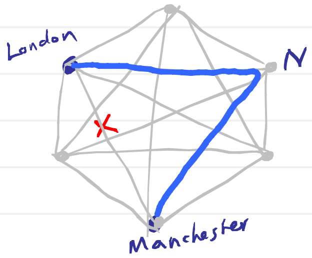
We can then sketch the drift diagram. If the arrows push us towards a single point, that is a sign that the iterative fixed point method will converge.

A similar drift model might even represent the actual dynamics of the system. E.g. if B_2 is very small, many calls will be accepted, and a little later B_2 will be bigger. In terms of the drift model: if B_2 is small, then the drift is positive, so after a few updates B_2 will get bigger.

The numerical procedure for finding fixed points tends to work more reliably when applied to actual system dynamics than to an arbitrary update equation.

§4.3 Dynamic Alternative Routing

We now look at an application of the fixed point method, to the telephone network.
We'll also see the relationship between drift models & fixed point.



A telephone network typically has a fully-connected core (of ≈ 50 nodes for BT).
It may be that e.g. all the circuits on the London — Manchester link are busy, or that the link was cut by mistake.
Then it makes sense to allow calls to be routed indirectly, over a 2-link path, if there is spare capacity.

Consider the following procedure (called dynamic alternative routing):
When a call arrives wanting to connect two nodes L and M,

(1) If the direct link $L \leftrightarrow M$ has a free circuit, admit the call on that circuit.

(2) Otherwise, pick some other node N at random.
If there is a free circuit on $L \leftrightarrow N$, and on $N \leftrightarrow M$, admit the call on $L \leftrightarrow N \leftrightarrow M$.

(3) Otherwise, block the call.

Let there be n nodes in total, and suppose each link has c circuits.
Let the arrival rate of calls between each node-pair be λ, and let call durations be $\sim \mathrm{Exp}(\frac{1}{\mu})$.
[The total arrival rate of calls to the entire network is $\frac{1}{2}n(n-1)\lambda$.]
Let B be the probability that a given link has all its circuits busy.
[All links are symmetrical, so the probability is the same on each link.]

Then $\mathbb{P}(\text{a call is admitted}) = (1-B) + B\,(1-B)^2$

either the call is admitted on the direct link, or, it is blocked on the direct link and admitted on both the alternate links it chooses.

Also,

total traffic offered to a given link $L \leftrightarrow M$ $= \lambda + \underbrace{2(n-2)}\lambda \; B \; \frac{1}{n-2} \; (1-B)$

direct traffic for link $L \leftrightarrow M$

number of other node-pairs e.g. $L \leftrightarrow N$ that could use $L \leftrightarrow M$ as part of a two-hop route $L \leftrightarrow M \leftrightarrow N$

$\mathbb{P}$(such a call is blocked on its direct route $L \leftrightarrow M$ and tries a 2-link route)

$\mathbb{P}$(it chooses a 2-hop route that uses link $L \leftrightarrow M$, e.g. $L \leftrightarrow M \leftrightarrow N$)

$\mathbb{P}$(the other leg of this 2-link route does not block the call)

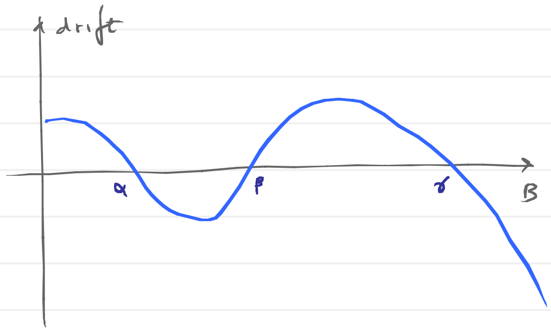
Thus $$B = E\left(\frac{\lambda}{\mu}\left[1 + 2B(1-B)\right],\; c\right).$$

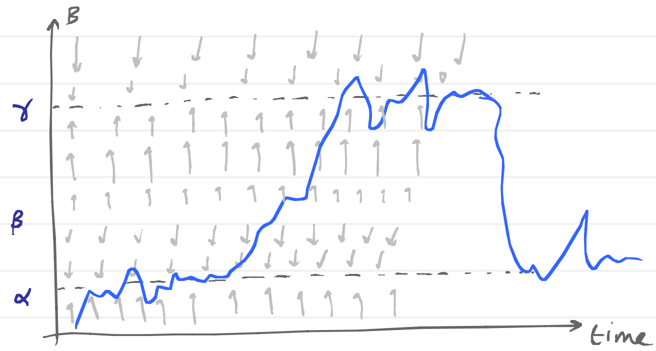
(Interestingly, this only depends on $\frac{\lambda}{\mu}$ rather than on the individual values of λ and μ. If we're simulating, that means one less dimension to explore.)

According to the iterative fixed-point method, we might

- pick an arbitrary starting guess, $B^{(0)}$
- update it by $B^{(n+1)} = E\left(\frac{\lambda}{\mu}\left[1 + 2\, B^{(n)} (1 - B^{(n)}) \right], C \right)$
- keep on updating until $B^{(n)}$ settles down.

The drift for this update method is $\quad \text{drift} = E\left(\frac{\lambda}{\mu}\left[1 + 2\, B (1-B) \right], C \right) - B.$





The drift diagram shows three fixed points (ie values of B where drift is zero)
Two of them are stable (ie. if there's a small fluctuation, the arrows push you back)
One of them is unstable (ie. if there's a small fluctuation, the arrows push you away).

We expect the system to stay near a stable fixed point, with small fluctuations; and every now and then to flip to the other stable fixed point.
This is called BISTABILITY. It's generally undesirable.

In this case, the large-B fixed point corresponds to the situation where most calls find their direct link busy and are forced onto a two-link route; thus most calls occupy two links not one; thus the network's capacity is halved.
MORAL. By making the network more flexible/adaptable, we have permitted it to get "stuck" in a bad state.

Q. Are there any "dampeners" we can put in place, to prevent the bad state while retaining the benefits of the good state?

A. Trunk reservation: it turns out that reserving a handful of circuits on each link for direct calls, e.g. reserve 10 when $C \gg 10000$, is enough.

"Loss networks", F.P.Kelly, 1991. http://www.statslab.cam.ac.uk/~frank/loss/

§4.3 Operational Laws

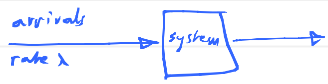
Suppose you count the number of jobs which pass certain points in the networks. There are some relationships which must always be true, regardless of any probability models, e.g.

in system = # arrivals − # departures.

These relationships are called <u>operational laws</u>.

LITTLE'S LAW

Treat the system (queue, network etc.) as a black box, with arrivals and departures. Suppose the system is stable, ie over a long timescale there is only a small discrepancy between arrivals and departures. Then



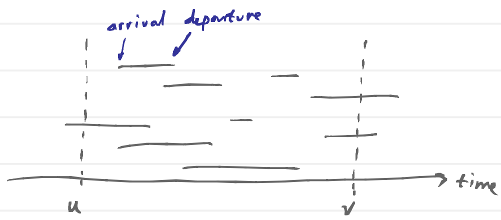
$$N = \lambda W$$

N: average occupancy of the system

λ: arrival rate

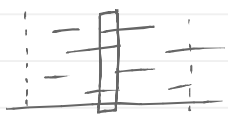
W: mean time that a job spends in the system

Proof Over a measurement period $[u,v]$, draw a line for every job that enters the system. What is the total length of line? There are two ways to measure it:



(1) $\mathbb{E}$ total length of line $= \mathbb{E}\#$ arrivals × average length of a line $= \lambda(v-u) \times W$.

(2) Split the time period $[u,v]$ into small boxes



Total length of line

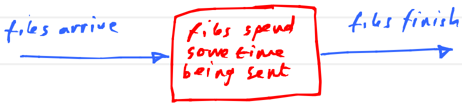
$$= \sum_{\text{all boxes}} \text{length of line in box}$$

$$\approx \sum_{\text{all boxes}} \text{width of box} \times \#\text{ flows present in that box, if boxes all are thin}$$

So $\mathbb{E}$ total length of line $= (v-u) \times \mathbb{E}\#$ flows present $= (v-u)\,N$.

So $\lambda(v-u)\,W = (v-u)\,N \Rightarrow N = \lambda W$.

Example Consider a processor-sharing link with arrival rate λ, mean file size m, link speed C. How long does it take to send a file, on average?

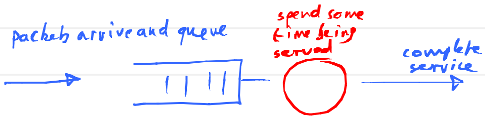


We know $\mathbb{E}\,\#\text{active jobs} = \rho/1-\rho$ where $\rho = \frac{\lambda m}{C}$.

The average time it takes to transmit a file is therefore

$$W = \frac{N}{\lambda} = \frac{1}{\lambda}\frac{\rho}{1-\rho} = \frac{m}{C-\lambda m}.$$

Example Consider a FIFO queue. What fraction of time does it spend busy, ie what is its utilization?



Consider the system to be the server.

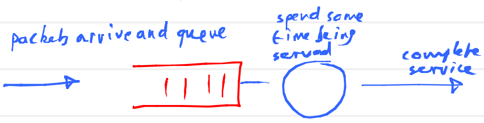
Either it is busy (occupancy $=1$) or idle (occupancy $=0$). Thus

$$N = \mathbb{E}\,\text{occupancy} = \mathbb{P}(\text{busy}).$$

Let m be the average service time for a packet, and let λ be the arrival rate. Then

$$\mathbb{P}(\text{busy}) = \lambda m.$$

Example Consider a FIFO queue. What is the average wait before beginning service?



We know that $\mathbb{E}\,(\text{total \# jobs in queue + at server}) = \frac{\rho}{1-\rho}$ where $\rho = \lambda m$.

But $\mathbb{E}\,(\text{total \# jobs in queue + at server}) = \mathbb{E}\,\text{queue size} + \mathbb{E}\,\text{occupancy of server}$

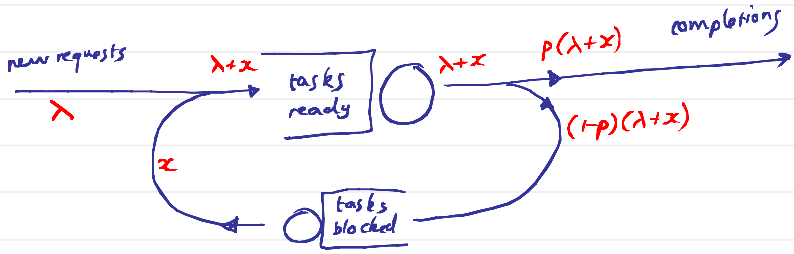
hence $\mathbb{E}\,\text{queue size} = \frac{\rho}{1-\rho} - \rho = \frac{\rho^2}{1-\rho}$.

Applying Little's law,

$$\frac{\rho^2}{1-\rho} = \lambda \times \text{av. queueing delay} \Rightarrow \text{av. queueing delay} = \frac{1}{\lambda}\frac{\rho^2}{1-\rho} = m\frac{\rho}{1-\rho}.$$

FLOW CONSERVATION LAW

If a queue is stable, then average arrival rate = average departure rate.
We used this in §3.8 to analyse a "queueing network" model of a webserver.



Let total rate of new requests $= \lambda$,
and let rate of tasks leaving the "tasks blocked" queue be x.
Then the total rate into "tasks ready" is $\lambda + x$.
Assuming stability, the total rate out of "tasks ready" is also $\lambda + x$.
Suppose that a fraction p of these tasks are completions,
and the remaining $(1-p)$ go to "tasks blocked".
Thus, the rate at which tasks enter "tasks blocked" is $(1-p)(\lambda+x)$.
Assuming stability, the rate in and the rate out of "tasks blocked" agree.
Thus

$$(1-p)(\lambda+x) = x \quad \Rightarrow \quad (1-p)\lambda = px \quad \Rightarrow \quad x = \frac{1-p}{p}\lambda .$$

We now know all the flow rates. (Assuming stability.)

UTILIZATION LAW

The utilization of a component over an interval is

$$\text{utilization} = \frac{\text{work done over the interval}}{\text{max work do-able}}$$

$$\approx \frac{\text{work arriving}}{\text{max work doable}}$$

assuming the system is stable, so that work in ≈ work out.

$$= \frac{\#\text{arrivals} \times \text{av. job size}}{\text{max work doable}}$$

$$= \frac{\overbrace{\left(\frac{\#\text{arrivals}}{\text{length of interval}}\right)}^{\text{the arrival rate, } \lambda} \times \overbrace{\text{av. job size}}^{\text{call this } m}}{\underbrace{\left(\frac{\text{max work doable}}{\text{length of interval}}\right)}_{\text{call this the service rate, } c}}$$

$$= \frac{\lambda m}{c}$$

Example: Processor sharing

$$\text{utilization} = \text{job arrival rate [jobs/sec]} \times \frac{\text{mean job size [bits/job]}}{\text{service rate [bits/sec]}}$$

Example: Erlang link.

$$\text{utilization} = \text{offered call arrival rate} \times \left(1 - \text{blocking prob}\right) \times \frac{\text{mean call duration}}{\#\text{ circuits}}$$

Example: FIFO queue.

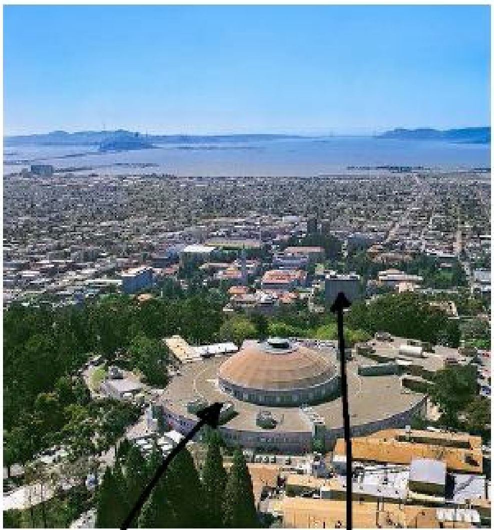
$$\text{utilization} = \text{pkt arrival rate [pkt/sec]} \times \text{mean service time per packet [sec/pkt]}$$

$$= \text{pkt arrival rate [pkt/sec]} \times \frac{1}{\text{service rate [pkts/sec]}}$$

§ 5 TCP

The history of the Internet

- 1974: First draft of TCP/IP
  "A protocol for packet network interconnection",
  Vint Cerf and Robert Kahn
- 1983: ARPANET switches on TCP/IP
- 1986: Congestion collapse
- 1988: Congestion control for TCP
  "Congestion avoidance and control", Van Jacobson



Lawrence Berkeley National Laboratory

Electrical Engineering, Berkeley University

"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."

Van Jacobson, "Congestion avoidance and control", 1988

In this section, we will develop a mathematical model for how TCP works. We will use fixed point method & drift analysis.

§5.1 The TCP Algorithm

TCP is based on window-based flow control, a simple version of which is as follows:

- When a flow starts, the sender is allocated a certain number of tokens.
- Each time he sends a packet, he uses up a token
- When the receiver receives a packet, he replies with an acknowledgement. When the sender receives an acknowledgement, he gets one token back.

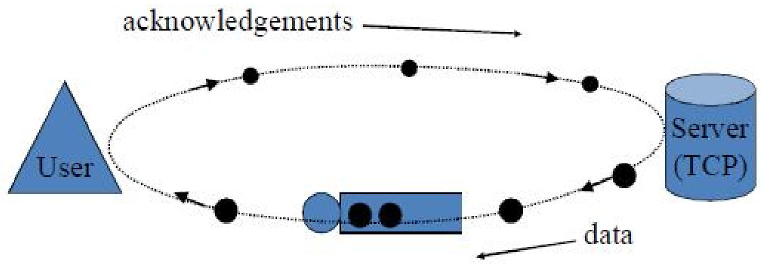


Illustration with 8 tokens.

The number of tokens is called the "window size".
The time for a packet to reach the destination & the acknowledgement to reach the sender is called the "round trip time" (RTT).

WINDOW SIZE AND TRANSMIT RATE

If a flow has window size W, (and if this number remains constant), there are W packets sent every RTT. Thus, the transmit rate is

$$x = \frac{W}{RTT} \text{ pkts/sec.}$$

Note: window-based flow control governs the overall average transmit rate, but it doesn't control the "burstiness":

start out with four tokens. Send out four packets back-to-back

one round trip time (RTT) later, four tokens come back, so we can send out four new packets

or

if the initial allocation of 4 tokens were spaced out, then packets would not be sent in bursts — and the next rounds of packets will automatically be smooth too.

COPING WITH DROPPED PACKETS

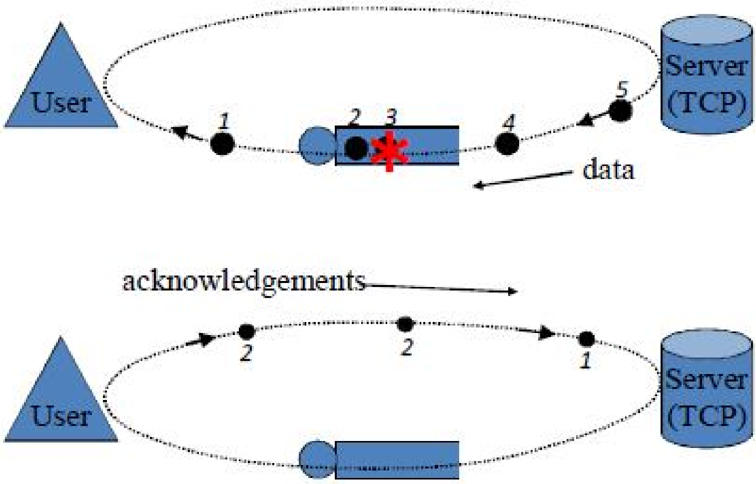
If a packet is dropped, the sender needs to detect this
— so it can retransmit the dropped packet, if necessary
— so it can reclaim the token, if necessary.

TCP detects dropped packets by sequence numbering:

The sender puts a sequence number on each packet

When the receiver receives a packet, it send back an ACK with the highest contiguous sequence number it has received

If the sender receives duplicate ACKs, it knows that a packet was dropped (or that something was delivered out of order)



A network control problem: how many tokens should each source have?

- If the receiver has a buffer of 10 packets, it could tell the sender "Start with 10 tokens; I'll issue you a new token whenever a packet is cleared from my buffer". This ensures that the receiver's buffer cannot overflow.
- Maybe the receiver needs to receive data at some average rate x_{min}, to ensure smooth video playback. It could tell the sender "Use $w = x_{min} \times RTT$ tokens".
- The transmit rate is $x = w/RTT$. There could be centralized admission control, like the Erlang link—issue new tokens only if the total traffic rate will not exceed the service rate.
- Is it possible to allocate tokens in a distributed way, so as to ensure that the network does not suffer from congestion collapse?

Jacobson realized that the problem with the original version of TCP is that it didn't limit the total number of tokens in the network. When there are too many tokens present, the total transmit rate is too high, and the network starts to drop packets. Flows respond by resending their dropped packets, further exacerbating the problem.

Jacobson devised an algorithm by which flows could adapt their window sizes to suit network conditions.

Note: Typically, congestion in the Internet causes queues to build up at routers, hence queueing delay increases, hence RTT increases, hence the transmit rate decreases, thus alleviating congestion.

It may however be the case that buffers at queues aren't big enough to cause there to be large enough queueing delays to alleviate congestion. Jacobson realized that queueing delay wasn't enough — there also needed to be a way to limit window sizes.

JACOBSON'S ALGORITHM

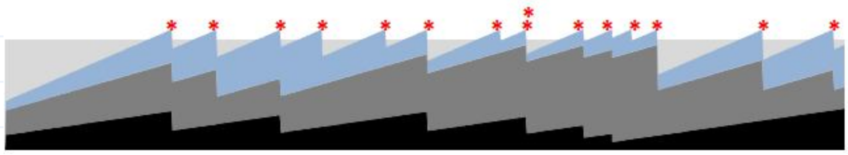
Jacobson's algorithm for congestion avoidance specifies how to adapt w:

- every ACK that the sender receives, increase w by $\frac{1}{w}$
- every data packet drop that the sender detects, decrease w by $\frac{w}{2}$ (but no more than once per RTT).

Note: In the increase phase, if window size is w then xmit rate is $x = \frac{w}{RTT}$, so I get $\frac{w}{RTT}$ ACKs per second. Each time, I increase my window by $1/w$ pkts. Hence, I increase my window by $\frac{1}{w} \times \frac{w}{RTT} = \frac{1}{RTT}$ pkts/sec. This is called "additive increase", and it's why the above diagram shows linear increases.
The decrease rule is called "multiplicative decrease".

WHAT TCP IS MEANT TO ACHIEVE

Each user increases his/her transmission rate when the network seems underused, and cuts it when one of his/her packets is dropped.



total link capacity. When total xmit rate exceeds capacity, there will be packet drops

If all users do this, the network should end up near-100% used, and the capacity should be shared fairly.

The Internet is the first large-scale network to be able to regulate itself – to share capacity fairly – without a central controller.

§ 5.2 Drift Model for TCP

A drift model is an equation for the expected rate of change in a quantity. Drift models are useful for showing us fixed points and stability / bistability / instability.

Suppose a TCP flow is using a link with packet drop probability p, and it has round trip time RTT. Let the window size at time t be W_t pkts. To find the drift in W_t, let's first write down how W_t evolves: for short durations δ (short enough that it's unlikely there's more than one ACK or drop),

$$W_{t+\delta} = W_t + \begin{cases} \frac{1}{W_t} & \text{if there is an ACK received in } [t, t+\delta] \\ -\frac{W_t}{2} & \text{if there's a drop detected in } [t, t+\delta]. \end{cases}$$

Thus, drift in W_t is

$$\frac{dW_t}{dt} = \frac{\mathbb{E}\,\text{change in } W_t}{\mathbb{E}\,\text{time for change}} = \frac{\mathbb{E}(W_{t+\delta} - W_t)}{\delta}$$

$$= \frac{1}{\delta}\left[\frac{1}{W_t}\mathbb{P}\left(\begin{matrix}\text{ACK received}\\ \text{in } [t,t+\delta]\end{matrix}\right) - \frac{W_t}{2}\mathbb{P}\left(\begin{matrix}\text{drop}\\ \text{in } [t,t+\delta]\end{matrix}\right)\right]$$

$$= \frac{1}{\delta}\left[\frac{1}{W_t} \times \delta\frac{W_t}{RTT}(1-p) - \frac{W_t}{2} \times \frac{\delta W_t}{RTT}p\right] = \frac{1-p}{RTT} - \frac{pW_t^2}{2RTT}$$

If I have window size W_t, I send W_t packets per RTT. Imagine splitting one RTT's worth of time into $\frac{RTT}{\delta}$ chunks of length δ. Of these, W_t contain a packet. The chance that a given chunk of length δ (chosen at random) contains a packet is therefore $\frac{W_t}{(RTT/\delta)} = \delta\frac{W_t}{RTT}$.

The chance that the chunk contains a packet AND the packet wasn't dropped, ie that it generates an ACK, is $\delta\frac{W_t}{RTT} \times (1-p)$.

The chance that this chunk of duration δ should contain a packet is $\delta\frac{W_t}{RTT}$.

The chance that it should contain a packet but the packet was dropped is $\delta\frac{W_t}{RTT} \times p$.

If p is small, it's reasonable to approximate the drift by $\frac{dW_t}{dt} \approx \frac{1}{RTT} - \frac{pW_t^2}{2RTT}$.

You may also see it written in terms of throughput $x_t = W_t/RTT$:

$$\frac{dx_t}{dt} = \frac{1}{RTT^2} - p\,x_t^2/2.$$

BEHAVIOUR OF THE DRIFT MODEL

The fixed point (ie where drift = 0) is at

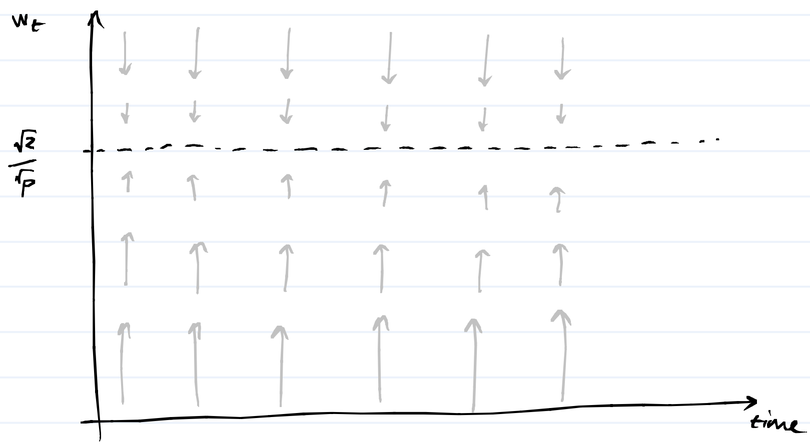
$$\frac{1}{RTT} - \frac{p w^2}{2 RTT} = 0 \Rightarrow w = \frac{\sqrt{2}}{\sqrt{p}} \text{ pkts.}$$

The fixed-point throughput is therefore

$$x = \frac{w}{RTT} = \frac{\sqrt{2}}{RTT\sqrt{p}} \text{ pkts/sec}$$

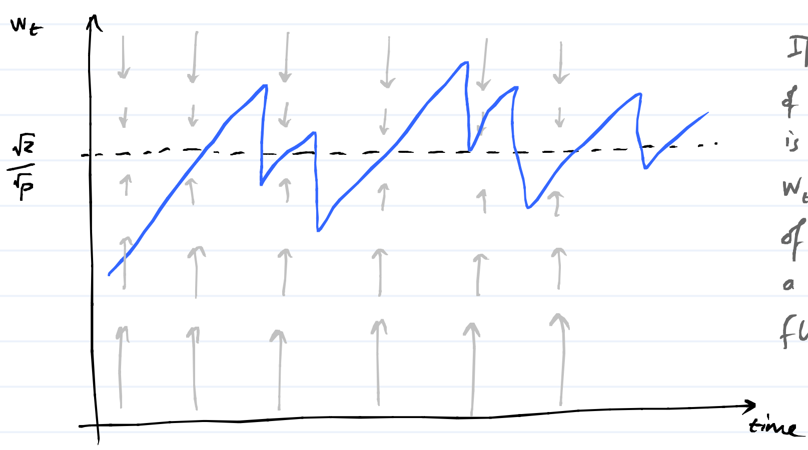
called the TCP Throughput equation

Drift diagram:



This is a stable fixed point because the arrows push you back after any deviation.

The drift model is just an approximation because it only looks at expected behaviour, whereas actual behaviour has random fluctuations. We know that TCP actually follows a sawtooth:



If w_t is very high, the probability of a drop is high, so the sawtooth is "pushed" down. Similarly if w_t is very low, when the prob. of a drop is low. This means that a simulation trace will tend to fluctuate around the fixed point.

STABILITY OF THE DRIFT MODEL (not examinable)

We argued that drift in w_t is $\frac{dw_t}{dt} = \frac{1}{RTT} - \frac{p w_t^2}{2RTT}$.

But in fact the ACKs/drops received at time t depend on what was sent at time $t-RTT$; and the drop probability may vary as well. Taking all this into account gives

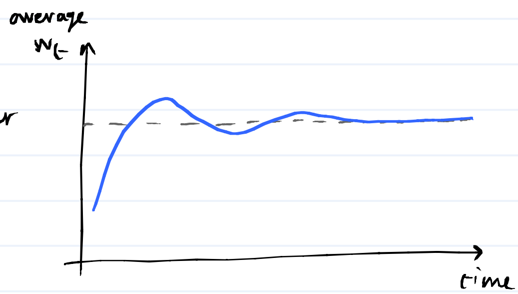
$$\frac{dw_t}{dt} \approx \frac{1}{RTT} - \underbrace{\left(P_{t-RTT} \frac{w_{t-RTT}}{RTT}\right)}\ \underbrace{\frac{w_t}{2}}$$

The chance I detect a drop at time t depends on the rate I was sending at time $t-RTT$, and on the packet drop probability experienced by packets sent at that time

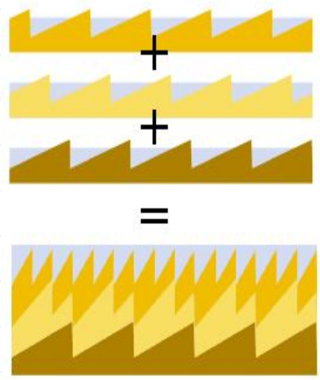
The amount by which I reduce my window depends only on my current windowsize.

A 1-dimensional drift diagram is not adequate to illustrate this. But we can still simulate the drift model, or analyse it mathematically.
— In the case of a single link shared by N TCP flows, and a drop probability p_t that depends on the link speed, buffer size, & total traffic rate of the N flows, we may see

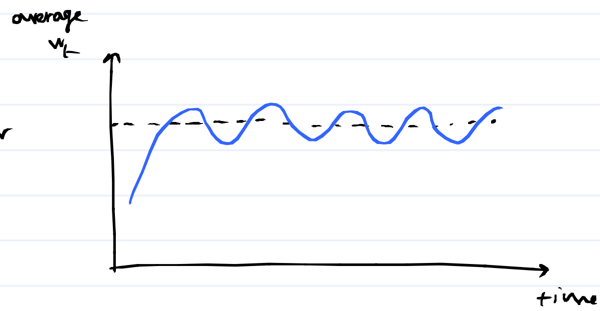
either



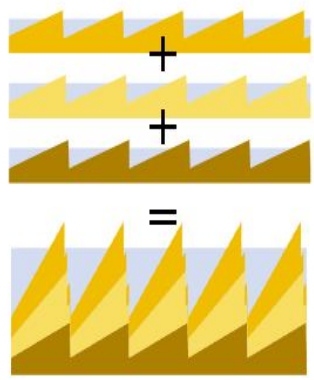
a stable fixed point, corresponding to unsynchronized TCP sawtooths



or



oscillations about the fixed point corresponding to synchronization between TCP sawtooths.



The problem of "How can we design congestion controllers that get good throughput and are stable?" has been a big research focus for the past 10 years.

§ 5.3 Fixed-point calculations

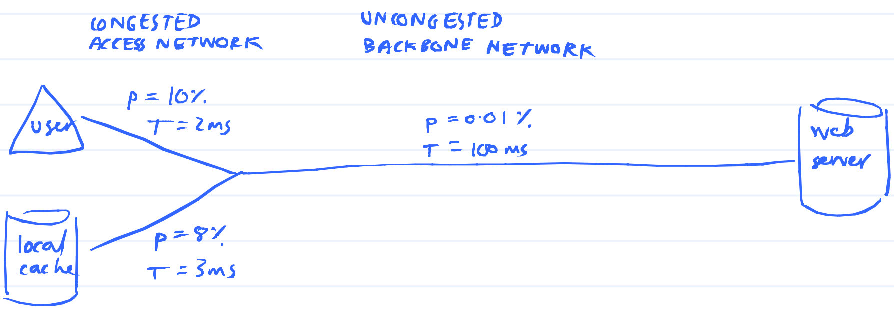
"Fixed point" has two meanings:

- Analysing the drift model, and finding the states where drift = 0. This gave us the TCP throughput equation, when we applied it to a single setup with a single TCP flow and a fixed packet drop probability.
- Writing down equations for each part of the system (such as the TCP throughput equation) and solving them simultaneously, e.g. with the iterative update procedure described in §4.2.

In this section we will work through three examples. The third example will illustrate the deep connection between the two meanings of "fixed pt".

EXAMPLE 1: CONTENT PLACEMENT

Suppose a user can download a piece of content either from a local cache or from the remote server; suppose also that the local network is congested. Which does the user prefer?



If the user downloads from the server:

$$RTT = 2 \times (0.002 + 0.1) = 0.204 \text{ sec}$$

$$\text{pkt drop prob} = 1 - \mathbb{P}(\text{pkt not dropped}) = 1 - \mathbb{P}(\text{not dropped on first link AND not dropped on 2nd link})$$

$$= 1 - (1-0.1)(1-0.0001) = 0.1009$$

$$\text{throughput} = \frac{\sqrt{2}}{RTT\sqrt{p}} = \frac{\sqrt{2}}{0.204\sqrt{0.1009}} = 21.8 \text{ pkt/sec} = 32.0 \text{ kB/sec} \quad (\text{at } 1 \text{ pkt} = 1500 \text{ Bytes})$$

If the user downloads from the local cache:

$$RTT = 2 \times (0.002 + 0.003) = 0.01 \text{ sec}$$

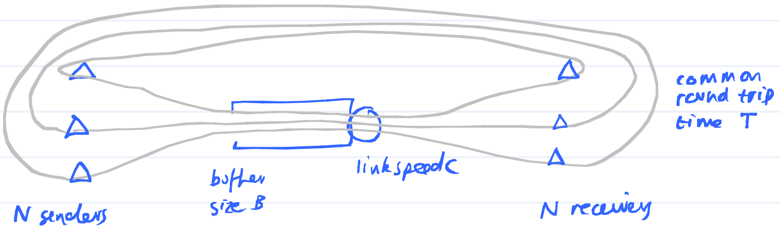
$$\text{pkt drop prob} = 1 - (1-0.1)(1-0.08) = 0.172$$

$$\text{throughput} = \frac{\sqrt{2}}{RTT\sqrt{p}} = \frac{\sqrt{2}}{0.01\sqrt{0.172}} = 341 \text{ pkt/sec} = 500 \text{ kB/sec}$$

TCP prefers short congested links to long uncongested links.
This is silly! Surely it'd be better to use up spare capacity in the backbone, rather than adding more traffic to an already congested access network!

EXAMPLE 2: BUFFER SIZING

Suppose a link is shared by several users. How big should the buffer be, to ensure good utilization & pkt drop probability? — or, because it's hard to define "good" — how do utilization & pkt drop prob. depend on buffer size?



We know that TCP throughput depends on the packet drop probability it experiences. Clearly, the packet drop probability depends on the load at the link. We therefore have two "subsystems", each of which depends on the other. We begin by writing out performance equations for each subsystem.

TCP subsystem: each flow gets throughput $x = \sqrt{2}/T\sqrt{p}$, where p = pkt drop probability

Queue subsystem: Packet drop prob. is $p = \dfrac{\rho^B(1-\rho)}{1-\rho^{B+1}}$ where $\rho = \dfrac{\text{total arr. rate}}{\text{service rate}} = \dfrac{Nx}{C}$

This formula is from §3.4. It's for a FIFO M/M/1/B queue.

Rewriting these equations in terms of p and ρ,

$$\rho = \frac{\sqrt{2}}{\frac{CT}{N}\sqrt{p}} \qquad \text{and} \qquad p = \frac{\rho^B(1-\rho)}{1-\rho^{B+1}}.$$

We can then use the iterative fixed-point method (or any other method you like) to solve these simultaneous equations for ρ and p.

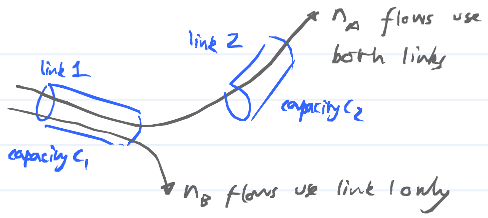
- It's up to you, the modeller, to choose which variables to use. I've chosen ρ and p because they are simple to understand, and because I expect them to be reasonably scale-invariant, that is, they have the same units no matter what units I choose for C or B.
- Also, note that I've gathered all the constants into one place, as $C \cdot T/N$. This is good practice. It lets you see at a glance, for example, that if you double both C and N then nothing changes. You should also ask yourself: "does $C \cdot T/N$ have a natural interpretation, so that I can explain the results intuitively without having to go through the maths?"

Model variations:

- When B is large, the formula for packet drop probability may be approximated
$$p \approx \left(1-\frac{1}{\rho}\right)^+ = \begin{cases} 0 & \text{if } \rho < 1 \text{ — the buffer is hardly ever full} \\ 1-\frac{1}{\rho} & \text{if } \rho \geq 1 \text{ — the buffer is full nearly all the time.} \end{cases}$$
This makes it easier to solve the equations.
- We could instead use $x = \dfrac{\sqrt{2}}{(T+\frac{B}{C})\sqrt{p}}$ ie incorporate queueing delay in the term for round trip time.
- You should ask yourself: what modelling assumptions are hidden in my use of the formula from §3.4 for packet drop probability?

EXAMPLE 3: A NETWORK

What are the loss probabilities & link utilization levels in this simple network?



Let's write down equations for each subsystem:

Let p_1, p_2 be pkt drop probabilities on the two links
x_A, x_B be per-flow throughputs on the two routes.

Then,

$$x_A = \frac{\sqrt{2}}{RTT_A\sqrt{1-(1-p_1)(1-p_2)}}$$

see Example 1 for why this is the correct formula for overall drop prob.

$$x_B = \frac{\sqrt{2}}{RTT_B\sqrt{p_1}}$$

$$p_1 = \left(1 - \frac{1}{\rho_1}\right)^+ \quad \text{where} \quad \rho_1 = \frac{n_A x_A + n_B x_B}{c_1}$$

$$p_2 = \left(1 - \frac{1}{\rho_2}\right)^+ \quad \text{where} \quad \rho_2 = \frac{n_A x_A (1-p_1)}{c_2}$$

This says: only a fraction $1-p_1$ of traffic on route A makes it through link 1 and onto link 2.

Here I'm using the large-buffer approximate formula for drop probability, described at the end of Example 2.

We could solve this with the iterative fixed-point method

— start with guesses, e.g. $p_1^{(0)} = p_2^{(0)} = 2\%$, then calculate $x_1^{(0)}$ and $x_2^{(0)}$, then calculate $\rho_1^{(0)}$ and $\rho_2^{(0)}$, and use these to update your values for drop prob.:

$$p_1^{(1)} = (1 - 1/\rho_1^{(0)})^+, \quad p_2^{(1)} = (1 - 1/\rho_2^{(0)})^+ .$$

and keep doing this until the values "settle down".

Sometimes, this iteration doesn't work very well: the values jump all over the place. (E.g. if we decide $p_1 = 0$, then what is x_B?) If this happens, try taking "baby steps":

— start with guesses e.g. $p_1^{(0)} = p_2^{(0)} = 2\%$, then use the eqns to get $x_1^{(0)}$ and $x_2^{(0)}$. and $\rho_1^{(0)}$ and $\rho_2^{(0)}$. Then, update your values for drop prob. by

$$p_1^{(1)} = (1-\delta)\, p_1^{(0)} + \delta\left(1 - 1/\rho_1^{(0)}\right)^+, \quad p_2^{(1)} = (1-\delta)\, p_2^{(0)} + \delta\left(1 - 1/\rho_2^{(0)}\right)^+ .$$

Then use the eqns to get $x_1^{(1)}, x_2^{(1)}, \rho_1^{(1)}, \rho_2^{(1)}$; repeat until the values settle down.

How should you choose δ? Why did I use "baby steps" for p_1 and p_2 but I used the equations straight for x_1, x_2, ρ_1, ρ_2? There is no general answer. This is an art, not a science, in many cases.

Alternatively, you could set up a drift model for the entire system:

$$\frac{dx_A(t)}{dt} = \frac{1}{RTT_A^2} - \frac{x_A(t)^2}{2}\Big(1 - (1-p_1(t))(1-p_2(t))\Big)$$

$$\frac{dx_B(t)}{dt} = \frac{1}{RTT_B^2} - \frac{x_B(t)^2}{2}\, p_1(t)$$

$$p_1(t) = \left(1 - \frac{1}{\rho_1(t)}\right)^+ \qquad \rho_1(t) = \frac{n_A x_A(t) + n_B x_B(t)}{C_1}$$

$$p_2(t) = \left(1 - \frac{1}{\rho_2(t)}\right)^+ \qquad \rho_2(t) = \frac{n_A x_A(t)(1-p_1(t))}{C_2}$$

and solve this as you would solve any drift model (e.g. in Excel). It should settle down to a fixed point (ie where drift = 0).

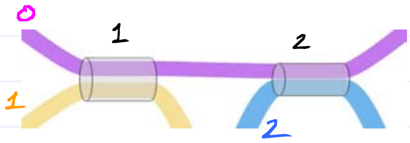
Note: if drift = 0, then the resulting values for $x_A, x_B, p_1, p_2, \rho_1, \rho_2$ solve the original fixed-pt equations. In other words, you get the same answer whether you start with fixed-pt equations and solve them using the iterative method, or if you write down the drift model and run it until you reach a fixed point i.e. a point where drift = 0.

This is why we use the name "fixed point" for both approaches

The drift model is rather like the "baby steps" approach, except it does a better job of choosing the right-size steps.

§5.4 Teleology of TCP

TCP is a distributed algorithm for assigning window size (ie rate) to flows.
We should step back and ask: is its allocation of rates sensible?
For example: does it make good use of available capacity? Is it fair?



This seems fair (all flows have the same rate), but link 1 is underutilized

Should we still call this fair? Flow 1 gets a larger chunk of link 1: but if we make link 1 share fairly then flow 2 must be disadvantaged.

Should flow 0 be penalized for taking up two links? In this allocation, each flow consumes the same total capacity – is this fairer?

TCP throughput is $x = \frac{\sqrt{2}}{RTT\sqrt{p}}$: the dependence on RTT brings in even thornier questions about what we mean by "fair".

First, recall the fixed-point equations for this network:

$$x_0 = \frac{\sqrt{2}}{RTT_0 \sqrt{p_1+p_2}}$$

Before, we wrote down $1-(1-p_1)(1-p_2) = p_1+p_2-p_1p_2$. But if p_1 & p_2 are small, this is $\approx p_1+p_2$.

$$x_1 = \frac{\sqrt{2}}{RTT_1 \sqrt{p_1}} \qquad p_1 = \left(1 - \frac{c_1}{x_0+x_1}\right)^+$$

$$x_2 = \frac{\sqrt{2}}{RTT_2 \sqrt{p_2}} \qquad p_2 = \left(1 - \frac{c_2}{x_0+x_2}\right)^+$$

I'm assuming the buffers are large. Otherwise the formula is more fiddly.

Consider this optimization problem, invented by Mo + Walrand in 2000:

$$\text{maximize} \quad \frac{-2}{RTT_0^2 x_0} + \frac{-2}{RTT_1^2 x_1} + \frac{-2}{RTT_2^2 x_2} - D_{c_1}(x_0+x_1) - D_{c_2}(x_0+x_2) \quad \text{over } x_0, x_1, x_2$$

where $D_c(x) = \max(x,c) - c - c\log\max(x,c)$, which implies $\frac{d}{dx} D_c(x) = (1 - c/x)^+$

We solve this in the usual way, setting derivatives equal to 0: for example

$$\frac{d}{dx_0} = \frac{2}{RTT_0^2 x_0^2} - \left(1 - \frac{c_1}{x_0+x_1}\right)^+ - \left(1 - \frac{c_2}{x_0+x_2}\right)^+ = 0$$

$$\Rightarrow \quad x_0 = \frac{\sqrt{2}}{RTT_0 \sqrt{\left(1-\frac{c_1}{x_0+x_1}\right)^+ + \left(1-\frac{c_2}{x_0+x_2}\right)^+}}$$

& similar equations for x_1 and x_2.

In other words, solving Mo + Walrand's optimization problem gives us a solution to the fixed point equations. Or, another way to put it: TCP is a distributed algorithm for solving Mo + Walrand's problem.

NOT EXAMINABLE:

In an arbitrary network, the general form of the optimization that TCP solves is the following:

$$\text{minimize} \quad \sum_{\text{all flows } r} U_r(x_r) - \sum_{\text{all links } j} D_j\Big(\sum_{\substack{\text{flows } r \text{ that} \\ \text{use link } j}} x_r\Big)$$

over all flow rate allocations $x_r \geq 0$

where $U_r(x) = \dfrac{-2}{RTT_r^2 x}$ and $D_j(y) = \displaystyle\int_0^y \text{pkt drop prob on link } j \text{ when arrival rate is } z \; dz.$

Jacobson had no intention to solve this optimization problem — but nonetheless it is there, hidden in his code. I call this TELEOLOGY which means "study of the intention or ultimate purpose of actions".

- It's generally hard to find distributed algorithms to solve complicated optimization problems. But here, we have a worldwide optimization problem, which is successfully distributed over every single computer connected to the Internet.

- This optimization problem can be shown to have a unique solution. Therefore TCP has only one fixed point, so it's not bistable like Dynamic Alt. Routing. Often, if a distributed system solves a well-behaved optimization problem, then the distributed system is likely to behave well. Also, whereas the iterative fixed point method and the drift model method may sometimes run into problems (ie the values keep jumping around when you try to solve them), there are robust methods for solving optimization problems of this sort.

The optimization problem is called by economists a "social welfare optimization".

$$\text{maximize} \quad \sum_r U_r(x_r) - \sum_j D_j(z_j) \quad \text{over } \underline{x}, \text{ where } z_j = \sum_{r:\, \text{user } r \text{ uses } j} x_r$$

Utility or "happiness" of individual r with his/her allocation x_r

the cost to society when resource j is used at level z_j

Jeremy Bentham (1748–1832)
Radical political theorist, secularist, founder of UCL, and father of the political/social theory of utilitarianism:

"The good is whatever brings the greatest happiness to the greatest number of people."

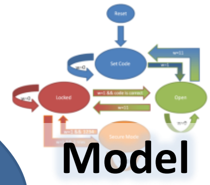
Teleology gives us a "birds-eye" picture of what TCP is "trying to achieve". It lets us ask:

- do we really want the network to solve this particular social welfare problem, or would a different utility function make more sense, eg not discount high-rate users so much?
- The utility function reflects a tradeoff between users, ie it DEFINES FAIRNESS.
  - It is easy to invent arbitrary measures of fairness, e.g. measuring how far we are from an equal allocation. But is "equal allocation" a sensible objective in a network, where some users use more links than others? The nice thing about a social welfare notion of fairness is that it gives a clean elegant consistent answer.

See the Handouts section for an application of teleology to routing problems, "Braess's paradox".

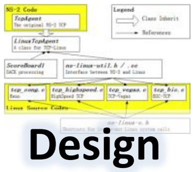
What is modelling?

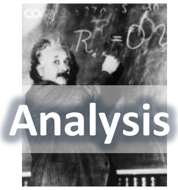
simplify



It’s very hard to know what to include in your model. If you choose the model right, you can express the real heart of what you want to understand about the real world, without any useless detail. This will make your model much easier to analyse.

make inferences





We come up with models because we want guidelines about how a system should be designed/built/operated. The model and analysis are only useful if we can translate the modelling conclusions back into real-world predictions.

What is modelling good for?

- Hacker insight is good for some problems.
- In other problems (especially distributed systems with adaptive behaviour), the network can have surprising behaviour.
- Modelling is a quick way to get insight into large-scale emergent behaviour. It can suggest where problems are likely to occur, and you can then check these out with more detailed models or simulation or experiment.

bistability of DAR

instability of wireless backoff

TCP's bad teleology

Braess's paradox

What is modelling good for?

Is it unstable?
e.g. processor sharing when $\rho>1$
If the system is unstable then it's useless to take measurements; we need to think about control systems to keep it stable.

Are there stable oscillations?
e.g. route flap, TCP synchronization. This may cause problems to some users.

Is it bistable?
e.g. dynamic alternative routing. Then there is unpredictable flapping, and the network can be hard to manage.

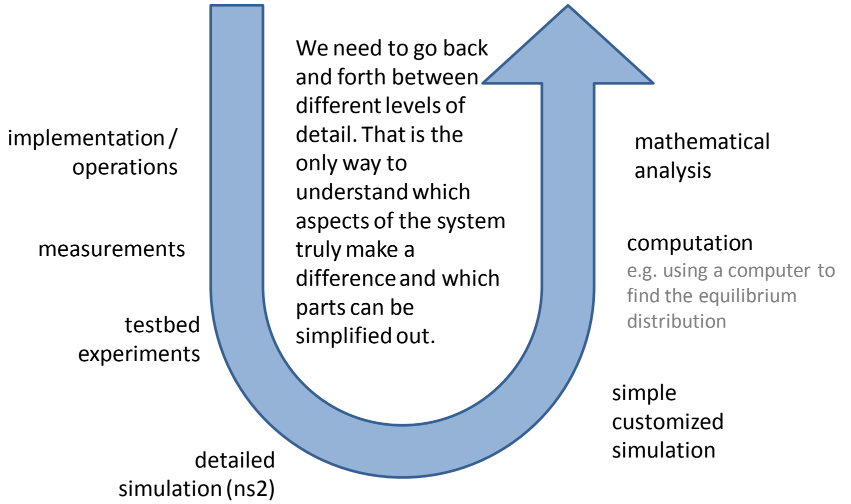
What is the teleology?
Is the network trying to achieve what I want it to achieve?

What are the causes of the behaviour we see?
Do we still see the behaviour when we create a simplified model, ditching certain real-world properties?

What are the parameters that matter?
e.g. for TCP, we decided that the relevant parameter is *wnd=RTT C/N*. This saves us from having to explore all three parameters separately.

What parameters should we investigate?
e.g. for what parameter values do we predict the system becomes unstable? What is the behaviour when the system is too large to simulate?

What should we model?



Tools we have learnt

- Random variables
  Describing data
- Poisson process
  Normal approximation
- Markov chains and processes
- Job models
  (Erlang link, processor sharing)
- Drift models,
  fixed points,
  operational laws

- **Microscopic description**
  fine-grained rules of behaviour, e.g. TCP code, Markov jump rates, detailed simulation
- **Macroscopic description**
  formulae for aggregates or averages, e.g. TCP throughput equation, Erlang fixed point, drift model
- **Teleological description**
  an optimization problem which has as its solution the fixed-point equations