§3 Analysing systems

In this section we will study three interconnected tools for analysing networked systems:

- drift models
- fixed point analysis
- operational laws.

These are heuristic / approximation tools. They are for getting insight into complex behaviour in situations where exact analysis is impossible and detailed simulation is overwhelming.

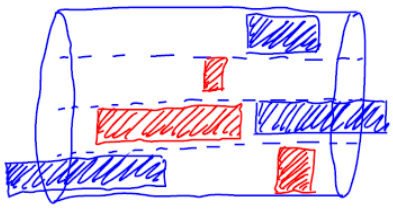
§3.1 Drift models

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



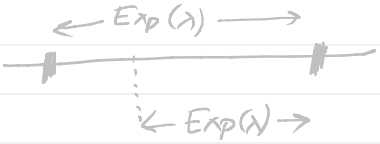
This is a simple model for a link in a circuit-switched telephone network, i.e. a network in which each call takes up a fixed amount of capacity and lasts for a given duration regardless of the other calls in the system.

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

Assume arrivals are a Poisson process of rate λ, and job durations are independent $\text{Exp}(\frac{1}{m})$ random variables. What is the drift in the number of active jobs?

Suppose at some point in time there are $n < C$ jobs active. What happens next (in the next iteration of an event-driven simulation loop)? Either an arrival or a departure. How long until either of these?

- Let X be the time until the next arrival. By the memoryless property, $X \sim \text{Exp}(\lambda)$.



- Let $Y_1 \dots Y_n$ be the times until the next departure of each one of the current jobs. Again, by the memoryless property, each $Y_i \sim \text{Exp}(\frac{1}{m})$

  Let $Z = \min(Y_1, \dots, Y_n)$. By the rule for minimums of exponentials, $Z \sim \text{Exp}(\frac{1}{m} + \dots + \frac{1}{m}) \sim \text{Exp}(\frac{n}{m})$.

By the rules for minimums of exponentials,

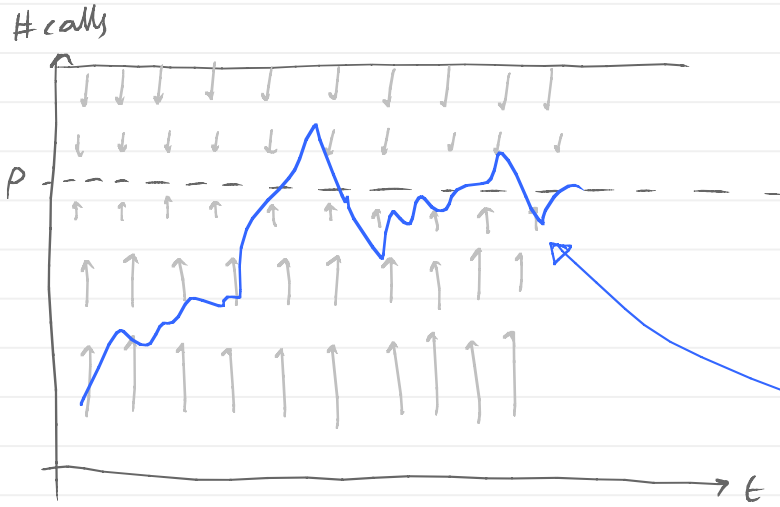
- time until next event $= \min(X, Z) \sim \text{Exp}(\lambda + \frac{n}{m})$
- probability that the next event is an arrival is $\mathbb{P}(X < Z) = \frac{\lambda}{\lambda + \frac{n}{m}}$

Now we have the information we need to calculate the drift in n.

$$\text{drift} = \frac{\mathbb{E}\,\text{change}}{\mathbb{E}\,\text{time to change}} = \frac{-1 \times \mathbb{P}(\text{next is dep.}) + 1 \times \mathbb{P}(\text{next is arr.})}{\mathbb{E}\,\text{time until next event}}$$

$$= \frac{-1\left(1 - \frac{\lambda}{\lambda + \frac{n}{m}}\right) + 1\,\frac{\lambda}{\lambda + n/m}}{\frac{1}{\lambda + \frac{n}{m}}} = \lambda - \frac{n}{m}.$$

A drift diagram is a plot which shows, for each value of the state, what direction the drift is and how big it is.



Let $\rho = \lambda m$.

drift < 0 when $n > \rho$.

drift $= 0$ when $\lambda - n\mu = 0 \Rightarrow n = \frac{\lambda}{\mu} = \rho$

drift > 0 when $n < \rho$

A simulation trace will tend to be pushed in the direction of the arrows, though there will still be some random fluctuations.

Here, the state $n = \rho$ is called a <u>fixed point</u>, because drift $= 0$.
It is called a <u>stable fixed point</u> 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 $t+\delta$?

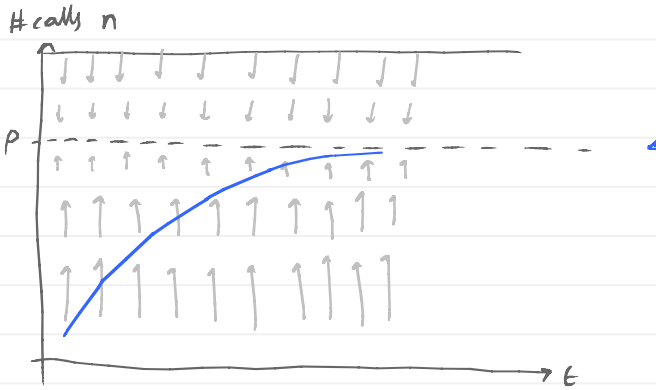
$$\frac{n_{t+\delta} - n_t}{\delta} = \frac{\text{change in } n_t}{\text{time for this change}} = \text{drift}.$$

Rearranging,

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

	A	B	C
	time	n	drift
1	0	0	$= \lambda - B1/m$
2	0.1	$= B1 + (A2-A1) \times C1$	$= \lambda - B2/m$
3	0.2	$= B2 + (A3-A2) \times C2$	$= \lambda - B3/m$
	⋮	⋮	⋮

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



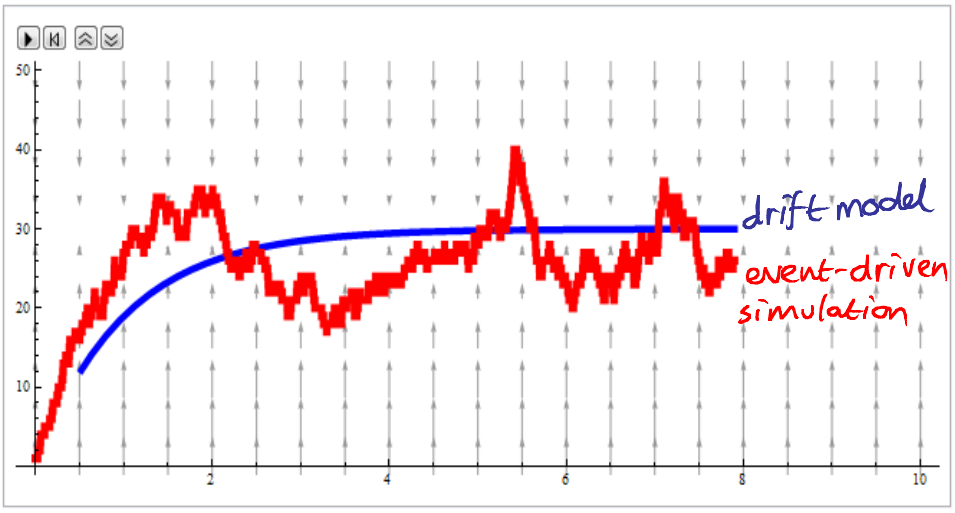
If we plot n from the Excel simulator we typically see smooth convergence to a fixed point. Sometimes the plot jumps all over the place; this probably means δ is too big.

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 $\frac{dn_t}{dt} = \lambda - n_t/m$

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

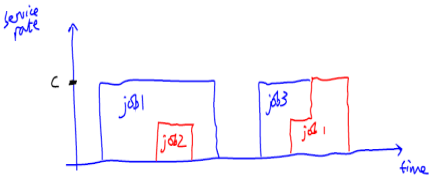
WHAT DOES THE DRIFT MODEL TELL US ABOUT THE REAL SYSTEM?



The drift model tells us about the "general direction". From this, we can in fact determine whether the system is stable:

THEOREM. If there is some finite region of the state space such that all fluid trajectories head towards this region, then it is stable. Otherwise it is unstable.

Example: processor-sharing link



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

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

Suppose job arrivals are a Poisson process of rate λ, and job sizes are independent $\text{Exp}(\frac{1}{m})$ random variables. Using exactly the same method as above, we find

$$\text{drift in \#active jobs} = \lambda - \frac{C}{m} = \frac{C}{m}(\rho - 1), \text{ where } \rho = \frac{\lambda m}{C}.$$

Alternatively, let's work out the drift in <u>workload</u> i.e. the sum of all work that has arrived and not yet been transmitted. Consider a short time-interval of length δ. How much work arrives and how much is served in this interval?

- service = δC, assuming that the link isn't idle, ie that workload > 0.
- $\#\text{arrivals} \sim \text{Poisson}(\lambda\delta)$, $\mathbb{E}\,\#\text{arrivals} = \lambda\delta$.
  Each arrival brings an average amount of work m.
  So the expected amount of new work is $\lambda\delta m$.

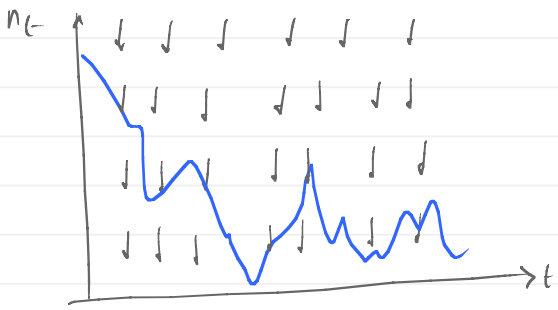
So,

$$\text{drift in workload} = \frac{\mathbb{E}\,\text{change in workload}}{\mathbb{E}\,\text{time for change}} = \frac{\lambda\delta m - \delta C}{\delta}$$

$$= \lambda m - C = C(\rho - 1).$$

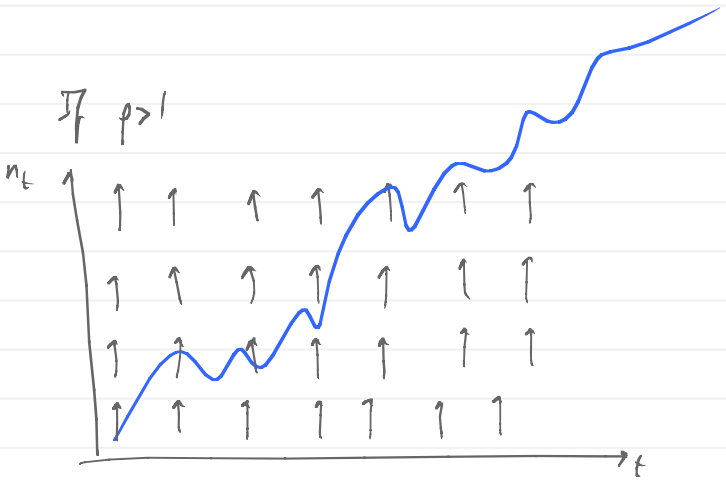
This is nearly the same as the drift in #active flows; they only differ by a factor $\frac{1}{m}$. This calculation is nicer because you don't have to assume that job sizes are $\text{Exp}(\frac{1}{m})$ random variables; this works for any distribution.

If $\rho < 1$:



Fixed point: $n_t = 0$

If $\rho > 1$



No fixed point; unstable.

WHY IS THERE A FIXED POINT AT 0? [non-examinable]

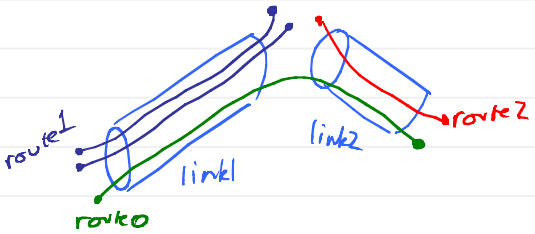
Drift at $n>0$ is $\lambda - \frac{c}{m} < 0$.
Drift at $n=0$ is $\lambda > 0$.
The actual system cannot go below $n_t = 0$, so it will "jitter" ie bounce up and down very close to $n_t = 0$.

We can in fact calculate something about its jittering.
Let π_0 = fraction of time that $n=0$,
$1-\pi_0$ = fraction of time that $n>0$
} while it's jittering around $n=0$.
The overall drift is
$$\pi_0 \lambda + (1-\pi_0)(\lambda - \tfrac{c}{m}).$$
We've decided that $n=0$ is a fixed point, so it must be that overall drift is zero. Hence
$$\pi_0 \lambda + (1-\pi_0)(\lambda - \tfrac{c}{m}) = 0$$
$$\Rightarrow \quad \lambda = (1-\pi_0)\tfrac{c}{m}$$
$$\Rightarrow \quad 1-\pi_0 = \frac{\lambda m}{c} = \rho$$
$$\Rightarrow \quad \pi_0 = 1-\rho.$$

In other words, the link spends a fraction of time $1-\rho$ idle.

§3.2 Fixed point models

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 Erlang links, and it's called the "Erlang fixed point". In §4 we'll use it to analyse TCP.



Suppose links 1 and 2 have capacities C_1 and C_2
Arrival rates on the three links are λ_0, λ_1, λ_2.
Call durations are all Exponential distributions with mean m

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

$$\mathbb{P}(\text{link 1 has all } C_1 \text{ circuits busy}) = E(\text{tot. arrival rate to link 1} * m, C_1).$$

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

$$\lambda_1 + \lambda_0(1-B_2)$$

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 $$B_1 = E\left(m\left[\lambda_1 + \lambda_0(1-B_2)\right], C_1\right)$$
Similarly $$B_2 = E\left(m\left[\lambda_2 + \lambda_0(1-B_1)\right], C_2\right)$$

Now we have to solve these two equations.

WHAT DOES THIS HAVE TO DO WITH THE DRIFT ANALYSIS IN §3.1? [Non-examinable]

The drift analysis says that the fixed point is #active calls $= \lambda m$.
(The C term, # circuits available, didn't appear in that analysis.)

If $\lambda m \ll C$, then the link very rarely fills up, and the drift analysis is fine.
If $\lambda m \approx C$ or $\lambda m > C$, then the link is frequently full, and the call blocking probability is determined by the "jitteriness" of the process at the boundary. Fluid analysis doesn't tell us anything about the size of the jitters. So, if blocking is big enough for us to worry about, we need something more than drift analysis — we need the Erlang formula

ITERATIVE METHOD FOR SOLVING FIXED-POINT EQUATIONS

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

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

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

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":

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

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

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 these update equations as a drift model:

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

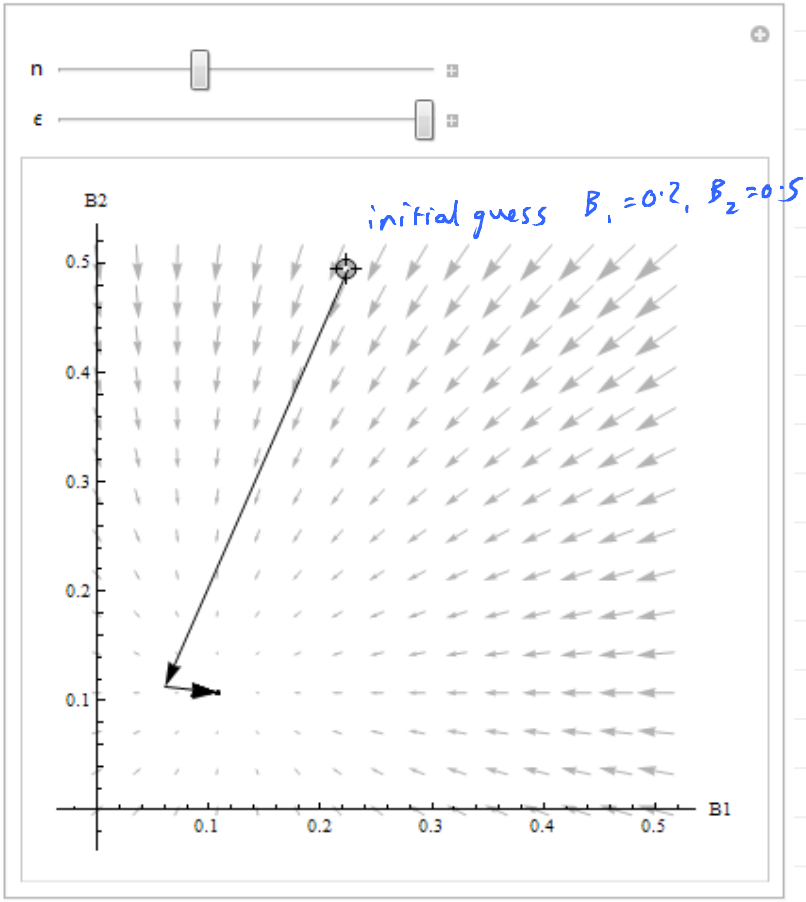
$$B_2^{new} \approx B_2 + \varepsilon\left(E\left(m\left[\lambda_2 + \lambda_0(1-B_1)\right], c_2\right) - B_2\right).$$

i.e. a drift model with timestep ε and drift equations

$$\text{drift in } B_1 = E\left(m\left[\lambda_1 + \lambda_0(1-B_2)\right], c_1\right) - B_1$$

$$\text{drift in } B_2 = E\left(m\left[\lambda_2 + \lambda_0(1-B_1)\right], c_2\right) - B_2.$$

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 NOTE ON 2D DRIFT DIAGRAMS

In this problem, there are two variables B_1 and B_2 whose drift I want to depict. So I've plotted a 2D drift diagram, B_1 on the x-axis, B_2 on the y-axis. The arrows from a given point show

(drift in B_1, drift in B_2).

For example, at $B_1 = B_2 = 0.5$, drift is $(-0.44, -0.39)$, so I've drawn an arrow pointing southwest.

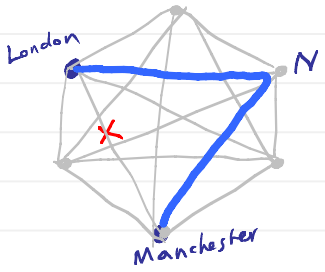
A SIMULATION TIP [Non-examinable]

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.

§3.3 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 $\approx$ 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 $\sim \text{Exp}\left(\frac{1}{m}\right)$
[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 $$\mathbb{P}(\text{a call is admitted}) = (1-B) + B(1-B)^2$$

- $(1-B)$: either the call is admitted on the direct link,
- $B(1-B)^2$: or, it is blocked on the direct link and admitted on both the alternate links it chooses.

Also,

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

- λ: direct traffic for link $L \leftrightarrow M$
- $2(n-2)$: number of other node-pairs e.g. $L \leftrightarrow N$ that could use $L \leftrightarrow M$ as part of a two-hop route $L \leftrightarrow M \leftrightarrow N$
- B: $\mathbb{P}$(such a call is blocked on its direct route $L \leftrightarrow M$ and tries a 2-link route)
- $\frac{1}{n-2}$: $\mathbb{P}$(it chooses a 2-hop route that uses link $L \leftrightarrow M$, e.g. $L \leftrightarrow M \leftrightarrow N$)
- $(1-B)$: $\mathbb{P}$(the other leg of this 2-link route does not block the call)

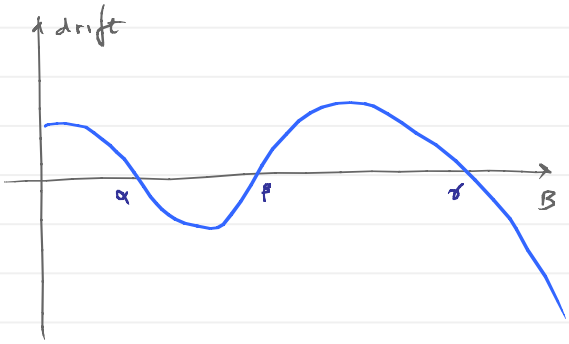
Thus $$B = E\left(\lambda m\left[1+2B(1-B)\right], C\right).$$

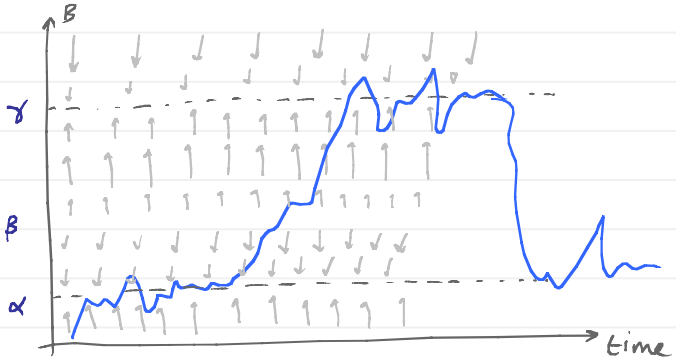
(Interestingly, this only depends on λm rather than on the individual values of λ and m. If we're simulating, that means one less dimension to explore.)

According to the iterative fixed-point method, we might

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

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





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/

§3.4 Operational Laws

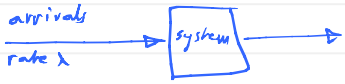
Suppose you count the number of jobs which pass certain points in the networks. There are some relationships which must always be true, regardless of any probability models, e.g.

in system = # arrivals − # departures.

These relationships are called operational laws.

A. 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

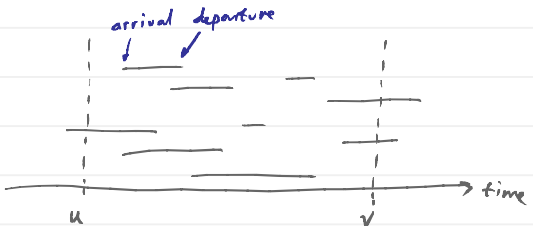


$$N = \lambda W$$

- N: time-average occupancy of the system
- λ: arrival rate
- W: mean time that a job spends in the system

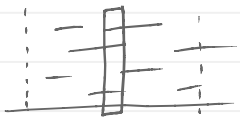
Proof [Non-examinable]

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

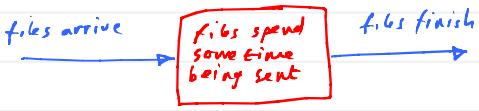
$$= \sum_{\text{all boxes}} \text{length of line in box}$$

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

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 have learnt that $\mathbb{P}(\#\text{active jobs} = r) = (1-\rho)\rho^r$, where $\rho = \frac{\lambda m}{C}$.
From this we can calculate $\mathbb{E}\,\#\text{active jobs} = \rho/1-\rho$.
The average time it takes to transmit a file is therefore

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

How to calculate $\mathbb{E}\,\#\text{active jobs}$:
Write N for the (random) number of active jobs, in equilibrium. We know

$$\mathbb{P}(N=r) = (1-\rho)\rho^r.$$

It's always a good idea to write this in terms of standard distributions, if we can. The closest random variable on the Random Variable Ref. Sheet is

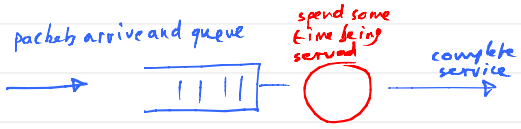
$$\mathbb{P}(X=r) = (1-p)^{r-1}p \quad \cdots\cdots \quad X \sim \text{Geometric}(p) \quad \cdots\cdots \quad \mathbb{E}X = \frac{1}{p}.$$

$$\mathbb{P}(N+1=r) = \mathbb{P}(N=r-1) = (1-\rho)\rho^{r-1}$$

Matching up the terms, we see $N+1 \sim \text{Geometric}(1-\rho)$, $\mathbb{E}(N+1) = \frac{1}{1-\rho}$.

So, $\mathbb{E}(N+1) = \frac{1}{1-\rho} \Rightarrow (\mathbb{E}N)+1 = \frac{1}{1-\rho} \Rightarrow \mathbb{E}N = \frac{1}{1-\rho} - 1 = \frac{\rho}{1-\rho}$.

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



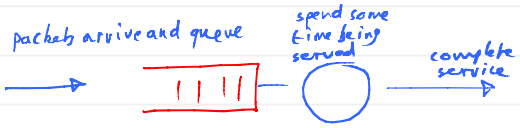
Consider the system to be the server.
Either it is busy (occupancy = 1) or idle (occupancy = 0). Thus

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

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?



When arrivals are Poisson and job service times are independent $\text{Exp}(\frac{1}{m})$ rand. vars, we learnt $\mathbb{P}(\#\text{jobs in queue + at server} = r) = (1-\rho)\rho^r$, where $\rho = \lambda m$.

As for the processor-sharing link example, $\mathbb{E}(\#\text{jobs in queue + at server}) = \frac{\rho}{1-\rho}$.

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

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

Applying Little's law,

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

B. FLOW CONSERVATION LAW.

Suppose you count the number of jobs which enter and leave a certain component in a network. Clearly,

$$\#\text{ inside at time } 0 + \#\text{ arrivals in } [0,T] = \#\text{ inside at time } T + \#\text{ departures in } [0,T].$$

Rearranging, and dividing by T,

$$\text{arrival rate over } [0,T] = \text{departure rate over } [0,T] + \frac{(\#\text{ inside at } T - \#\text{ inside at } 0)}{T}.$$

If the component is stable, then # inside at T doesn't grow steadily as T grows. Therefore, taking $T \to \infty$, for a stable component,

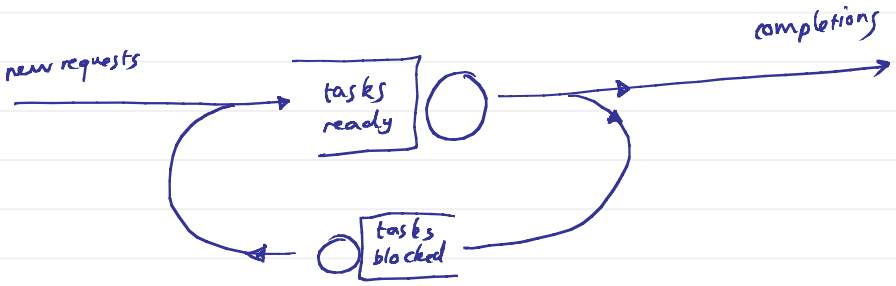
$$\text{arrival rate} = \text{departure rate}.$$

Here is an example:

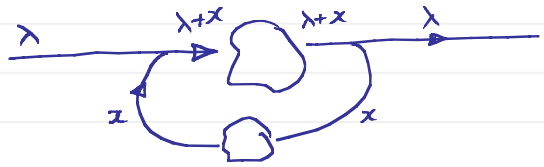
Question 10. Here is a model for a web server with active server pages, i.e. pages that cannot be served directly from the disk but instead require processing e.g. in PHP.

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?



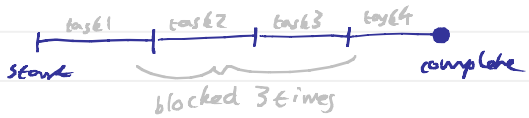
STEP 1: work out the flow rates, under the assumption that the system is stable (ie none of the queues are filling up)



We want to solve for x, where $\frac{x}{\lambda+x}$ is the fraction of tasks that block rather than complete. Clearly, the question text that says, on average,

> each request will block b times before completing.

has something to tell us about this.



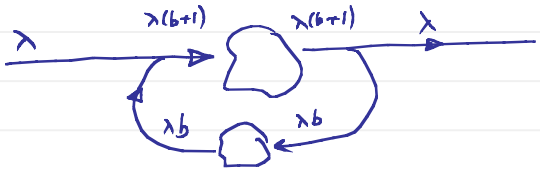
Let X = # tasks/request; then $\mathbb{E}(X-1) = b$.

Consider a large number of requests, made of $X_1, \ldots, X_n$ tasks. Then,

$$\text{fraction of tasks that block rather than complete} = \frac{(X_1-1)+\cdots+(X_n-1)}{X_1+\cdots+X_n} \approx \frac{nb}{n(b+1)} = \frac{b}{b+1}.$$

Thus $\frac{x}{\lambda+x} = \frac{b}{b+1} \Rightarrow x = \lambda b$.

So, the overall flow rates are



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

For this question it's even easier, since all we're asked for is stability. Consider the "tasks ready" queue.

Each arriving task brings with it i instructions on average (after which the task leaves this component, either because the job completes or is blocked).
So, work arrives at rate $\lambda(b+1) * i$ instructions/sec.
And work is performed at rate C instructions/sec.
Stability requires $\lambda(b+1)i < C$.

The 'tasks blocked' queue is an "open-loop workload component" (§2.6)
This is always stable, and obviously the mean waiting time is t.

STEP 3. Find the bottleneck, and other quantities.

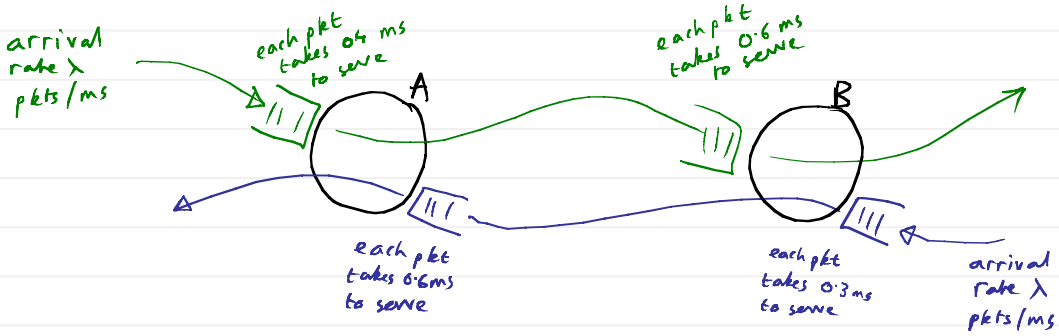
The only bottleneck here is the CPU. It is only stable if $\lambda(b+1)i < C$.
Thus, the maximum rate at which the server can handle requests is $\frac{C}{(b+1)i}$.

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

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

Step 2 said "analyse each component on its own".
Sometimes, it's not legitimate to analyse components on their own — the whole system "conspires". This is most commonly a problem when the components prioritize one flow over another.

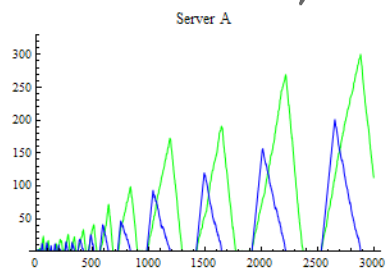
Here is a nice example due to Rybko + Stolyar (1992).

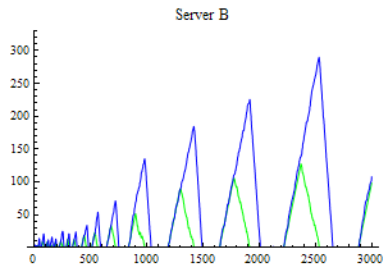


We'd expect A to be stable if tot arr.rate × mean job duration < 1
ie if $2\lambda \times 0.5 < 1 \iff \lambda < 1$. same for B.

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.

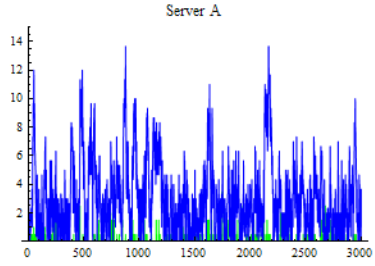
If servers A and B give priority to internal arrivals,

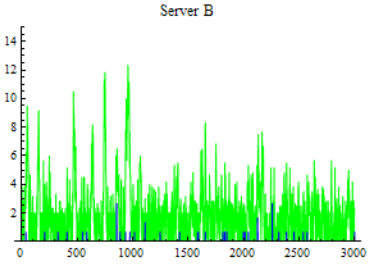




in fact, this is only stable when $\lambda < \frac{1}{12} = 0.333...$

If servers A and B are strictly FIFO,





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

C. UTILIZATION LAW

The utilization of a component over an interval is

$$\text{utilization} = \frac{\text{work done over the interval}}{\text{max work doable}}$$

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

assuming the system is stable, so that work in $\approx$ work out.

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

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

$\frac{\#\text{arrivals}}{\text{length of interval}}$ → the arrival rate, λ

av. job size → call this m

$\frac{\text{max work doable}}{\text{length of interval}}$ → 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.

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

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