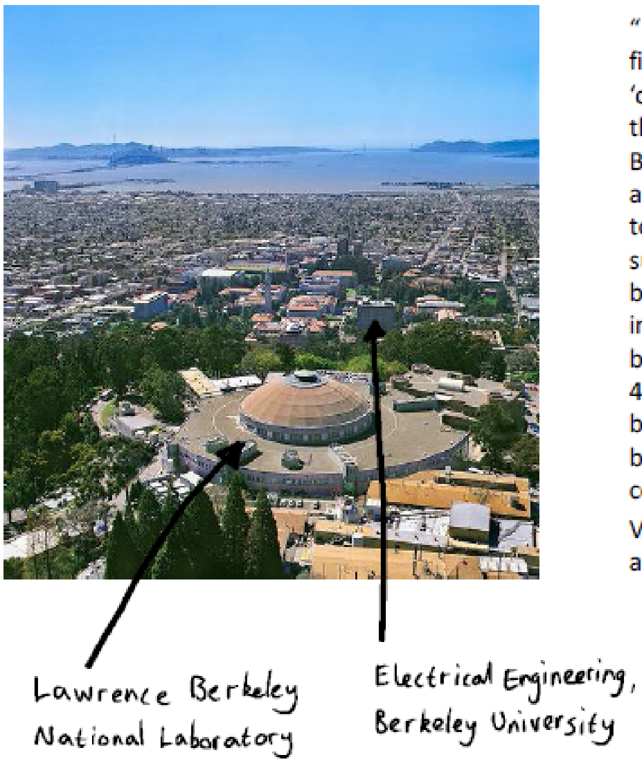
§4 TCP

Note Title 14/10/2011

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.

§4.1 The TCP Algorithm

TCP is based on window-based flow control, a simplified 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.

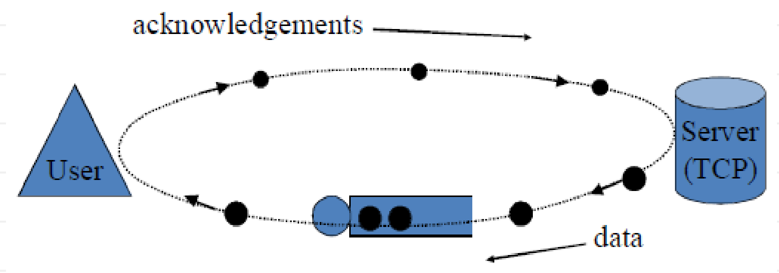


Illustration with 8 tokens.

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).

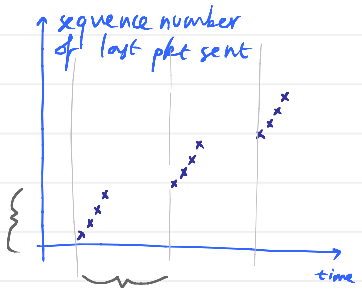
WINDOW SIZE AND TRANSMIT RATE

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

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

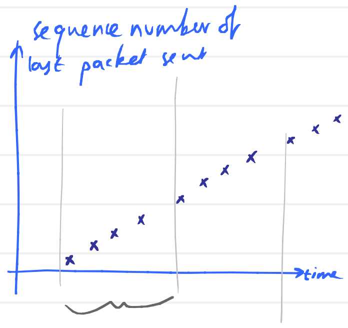
Note: window-based flow control governs the overall average transmit rate, but it doesn't control the "burstiness":

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

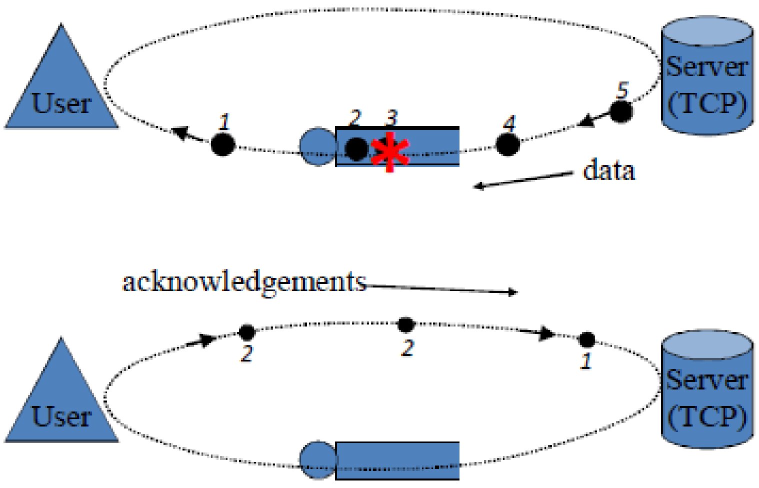
If a packet is dropped, the sender needs to detect this
— so it can retransmit the dropped packet, if necessary
— so it can reclaim the token, if necessary.

TCP detects dropped packets by <u>sequence numbering</u>:

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)



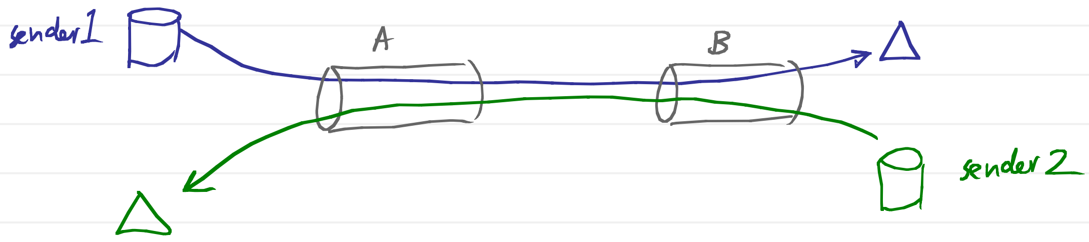
A network control problem: how many tokens should each source have?

- 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.

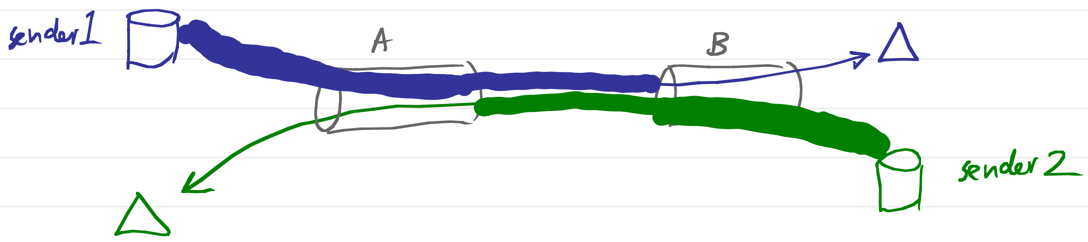
this is what pre-1988 TCP did.

- 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?

"congestion collapse" refers to a scenario in which the system's response to congestion has the effect that the network's "service effort" is wasted, which further exacerbates congestion, leading to complete network failure. Jacobson realized that pre-1988 TCP could cause congestion collapse:



Suppose at some time both links A and B are congested and they start to drop packets. Both senders respond by resending the dropped packets. This makes congestion worse:



Now sender 1 is sending lots of packets and crowding out sender 2 at link A, and vice versa at link B. Thus, both links are wasting their "effort" by serving packets that will just be dropped downstream.

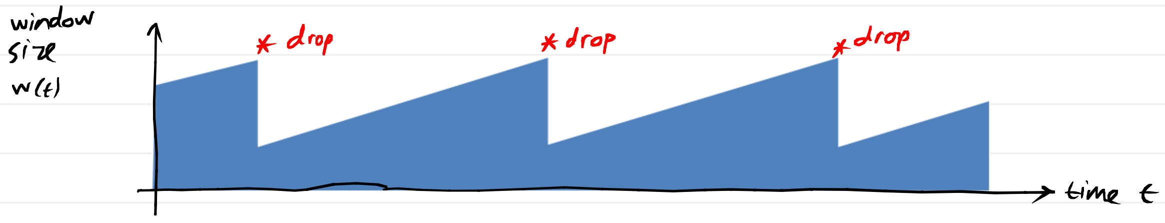
Note: Typically, congestion in the Internet causes queues to build up at routers, hence queueing delay increases, hence RTT increases, hence the transmit rate decreases, thus alleviating congestion.

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

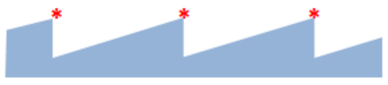
Jacobson's algorithm for congestion avoidance specifies how to adapt w:

- 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).

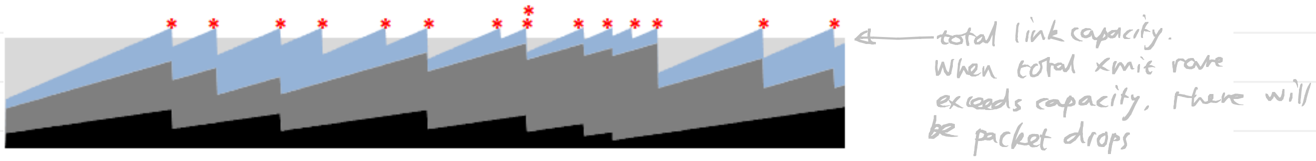


Note: In the increase phase, if window size is w then xmit rate is $x = \frac{w}{RTT}$, so I get $\frac{w}{RTT}$ ACKs per second. Each time, I increase my window by $1/w$ pkts. Hence, I increase my window by $\frac{1}{w} \times \frac{w}{RTT} = \frac{1}{RTT}$ pkts/sec.
This is called "additive increase", 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.



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.

§4.2 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 packets. 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

$$\frac{dw_t}{dt} = \frac{\mathbb{E}\,\text{change in } w_t}{\mathbb{E}\,\text{time for change}} = \frac{\mathbb{E}\,(w_{t+\delta} - w_t)}{\delta}$$

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

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

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 $\frac{w_t}{(\frac{RTT}{\delta})} = \delta\frac{w_t}{RTT}$.

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

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 $\frac{\delta w_t}{RTT} \times p$.

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

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

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

(Units of x_t are pkts/sec.)

BEHAVIOUR OF THE DRIFT MODEL

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

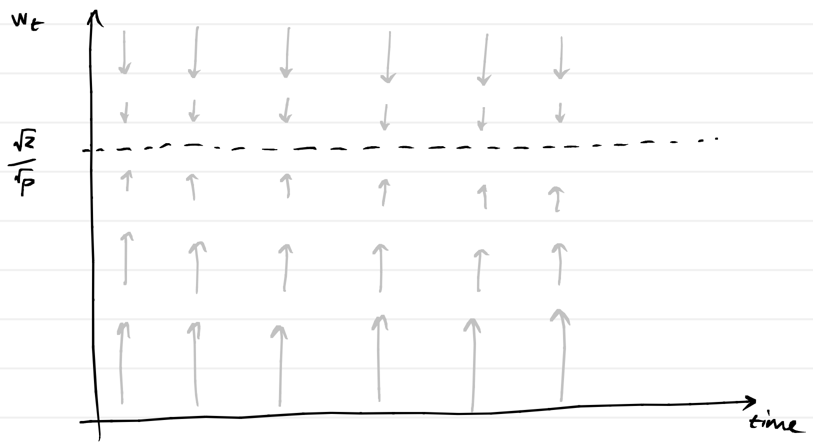
$$\frac{1}{RTT} - \frac{p\,w^2}{2RTT} = 0 \quad \Rightarrow \quad w = \frac{\sqrt{2}}{\sqrt{p}} . \quad \text{pkts.}$$

The fixed-point throughput is therefore

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

called the TCP Throughput equation

Drift diagram:

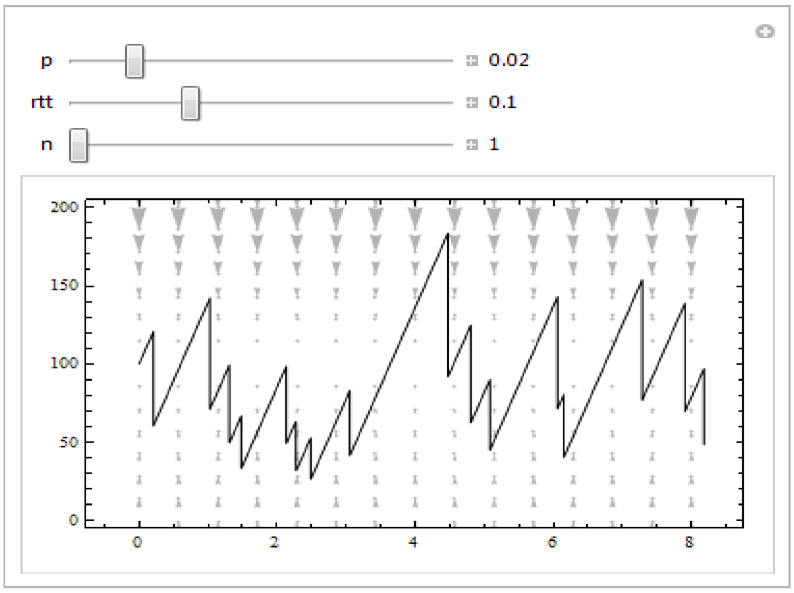


This is a stable fixed point because the arrows push you back after any deviation.

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

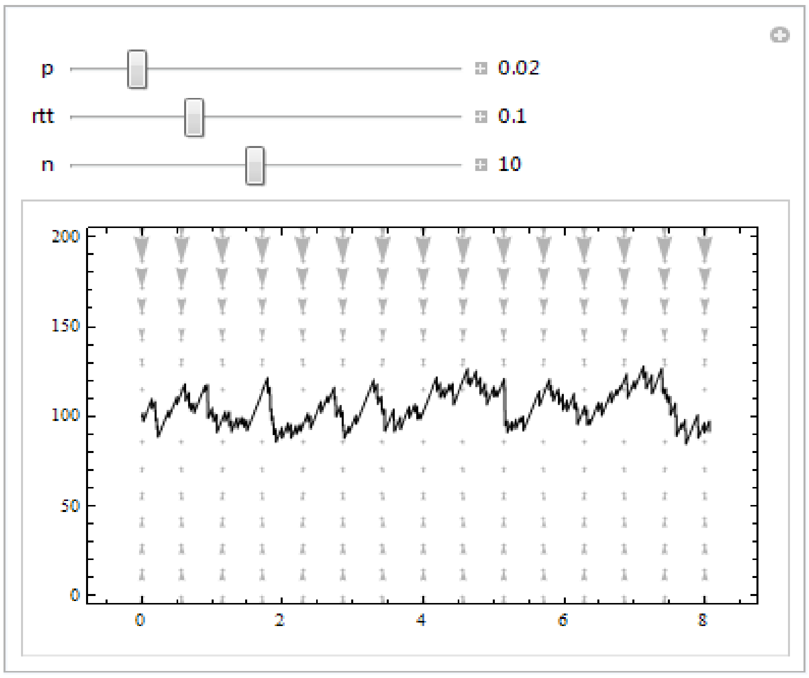
The drift model is just an approximation because it only looks at expected behaviour, whereas actual behaviour has random fluctuations. In fact, we know that TCP actually follows a sawtooth!

throughput of a single TCP flow



But when there are many TCP flows, each drop causes only one of them to cut its rate, so the aggregate traffic rate drops by much less than $\frac{1}{2}$, and the fluctuations stay nearer the fixed point.

average throughput of 10 TCP flows



In general, the drift model is good at telling us about the aggregate behaviour of a collection of independent components.

STABILITY OF THE DRIFT MODEL (not examinable)

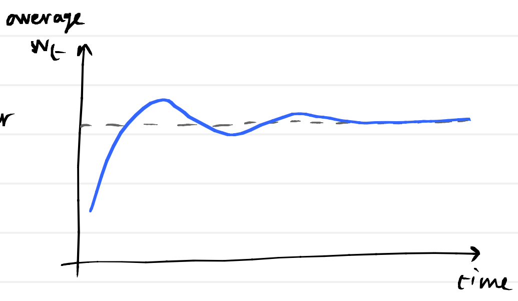
We argued that drift in w_t is $\frac{dw_t}{dt} = \frac{1}{RTT} - \frac{p w_t^2}{2RTT}$.

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

$$\frac{dw_t}{dt} \approx \frac{1}{RTT} - \underbrace{\left(p_{t-RTT} \frac{w_{t-RTT}}{RTT}\right)}_{\text{The chance I detect a drop at time } t \text{ depends on the rate I was sending at time } t-RTT\text{, and on the packet drop probability experienced by packets sent at that time}} \underbrace{\frac{w_t}{2}}_{\text{The amount by which I reduce my window depends only on my current windowsize.}}$$

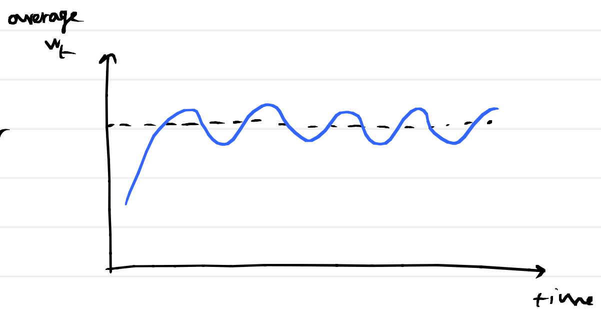
A 1-dimensional drift diagram is not adequate to illustrate this. But we can still simulate the drift model, or analyse it mathematically.
— In the case of a single link shared by N TCP flows, and a drop probability p_t that depends on the link speed, buffer size, & total traffic rate of the N flows,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

either

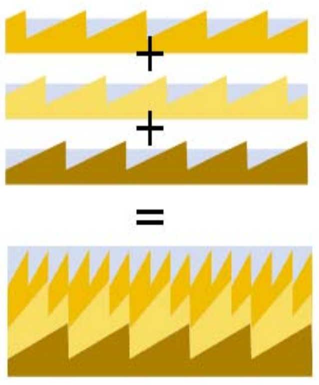


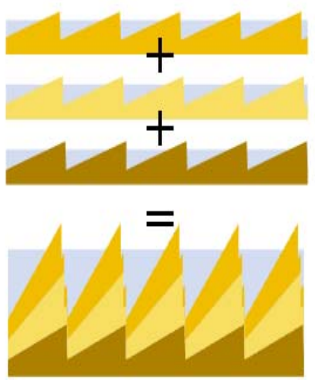
a stable fixed point, corresponding to unsynchronized TCP sawtooths

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.

§ 4.3 Fixed-point calculations

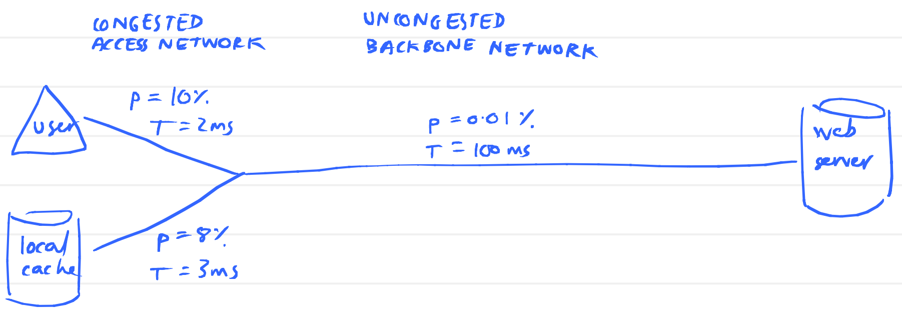
"Fixed point" has two meanings:

- Analysing the drift model, and finding the states where drift = 0. This gave us the TCP throughput equation, when we applied it to a simple 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:

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

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

$$= 1 - (1-0.1)(1-0.001) = 0.1009$$

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

If the user downloads from the local cache:

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

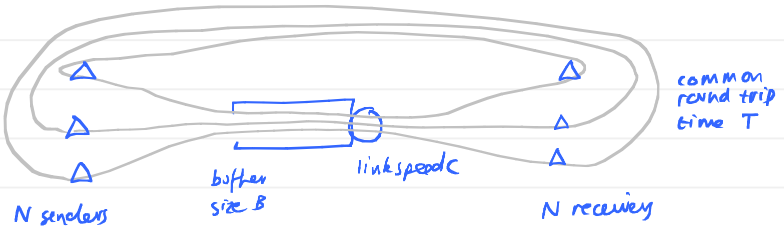
$$\text{pkt drop prob} = 1 - (1-0.1)(1-0.08) = 0.172$$

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

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 §2.6.4
It's for a FIFO M/M/1/B queue.

Rewriting these equations in terms of p and ρ,

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

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 <u>scale-invariant</u>, 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:

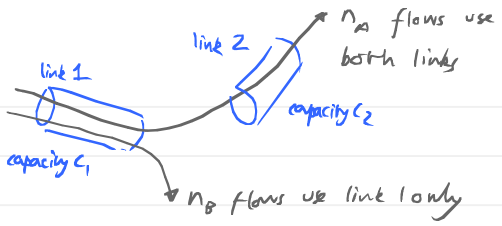
- When B is large, the formula for packet drop probability may be approximated

$$p \approx \left(1-\frac{1}{\rho}\right)^+ = \begin{cases} 0 & \text{if } \rho < 1 \quad \text{— the buffer is hardly ever full} \\ 1-\frac{1}{\rho} & \text{if } \rho \geq 1. \quad \text{— the buffer is full nearly all the time.} \end{cases}$$

  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 §2.6.4 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.:

$$p_1^{(1)} = \left(1-1/\rho_1^{(0)}\right)^+, \quad p_2^{(1)} = \left(1-1/\rho_2^{(0)}\right)^+.$$

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.