Control Problems in Decentralized Computing Systems

CDS 101/110
Friday 25, October, 2002

Eric Klavins
klavins@caltech.edu

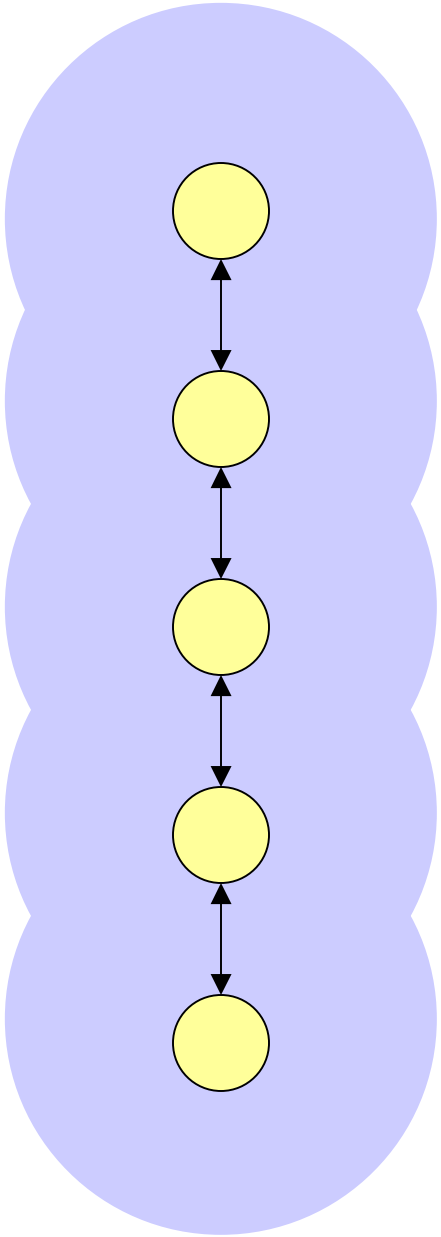
Context

In distributed computing:

- Processors do no know the global state.
- Communication may have delay and is usually unreliable.
- State spaces are not continuous.

Nevertheless, we want:

- Performance
- Fault tolerance, robustness
- Guarantees



Overview

Context

Brief examples

- Phase regulation
- Self-assembly

Detailed examples

- Self-stabilizing protocols
- Internet congestion control

Decentralized Phase Regulation

Example: consider a network of oscillators that try to maintain certain phase relationships with their neighbors.

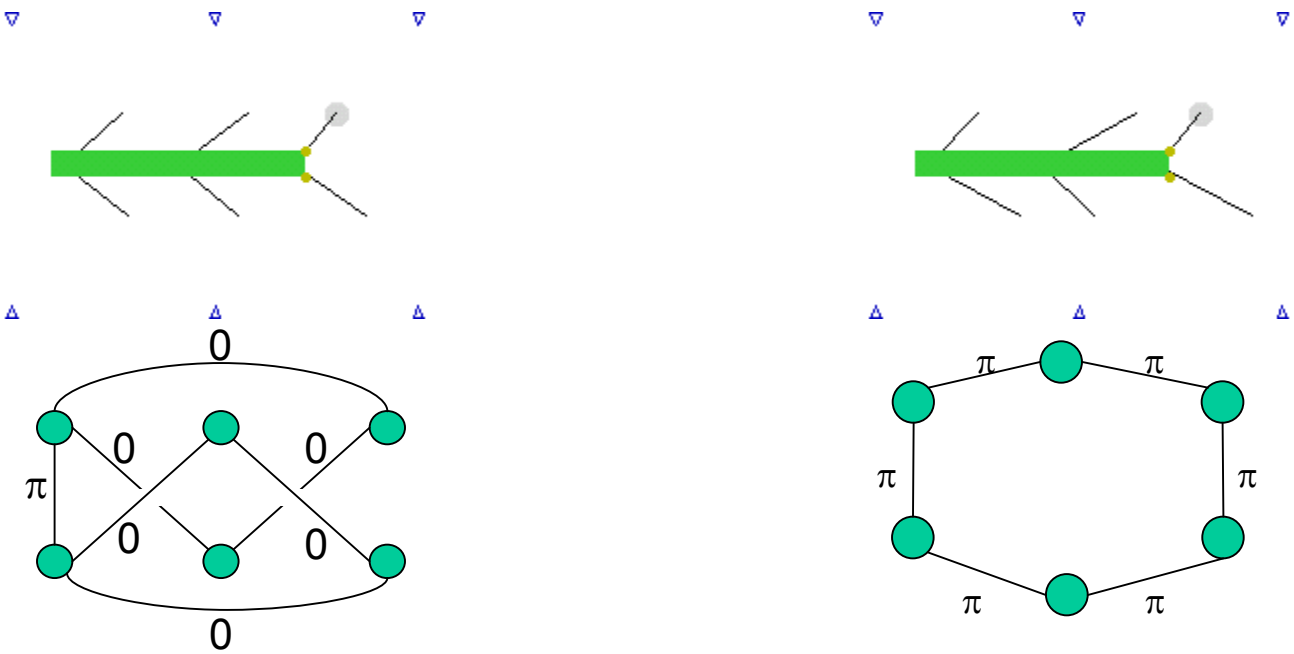
The network structure is vital to stability.

Equation for Individual Oscillator

$$\dot{\varphi}_i = 1 - \sum_{j=1}^{n} C_{i,j} \sin(\varphi_i - \varphi_j)$$

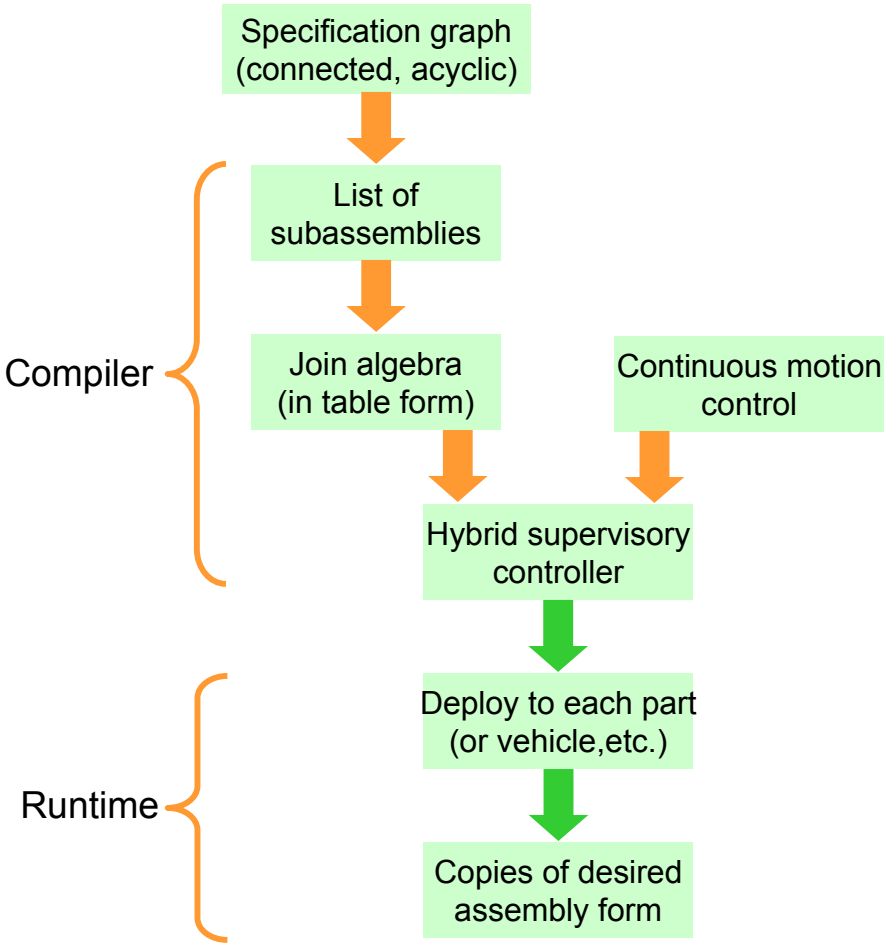
Desired limiting behaviors

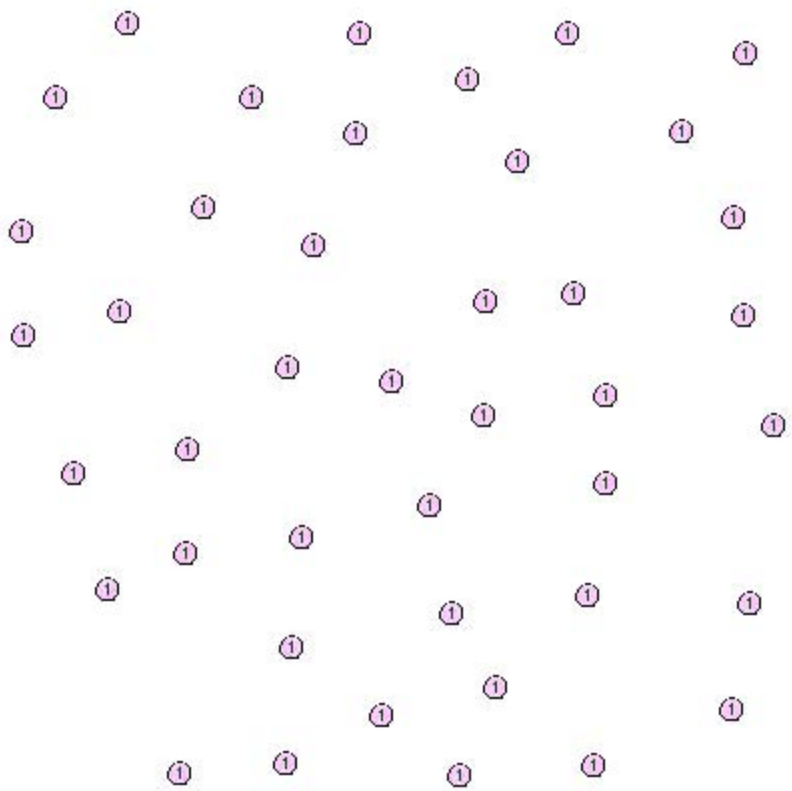
Coupling part (neighbors only)



Programmable Self Assembly

Example: It is sometimes possible to devise a program to be run by each agent so that the collective behavior "emerges".





Self-Stabilizing Protocols

Edsger Dijkstra

11 May 1930 - 6 August 2002

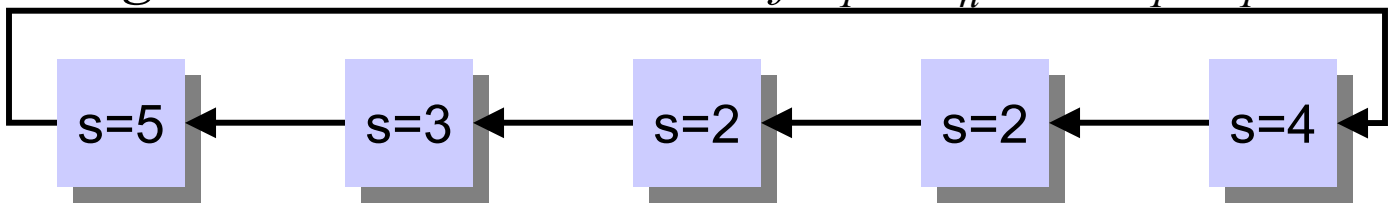
One reason to take a controls approach in networking is to ensure robustness.

Dijkstra (1974) proposed, a way to design self stabilizing protocols.

Here's a super simple example:

Original Protocol also has: if $s_1 = s_n$ *then* $s_1 = s_1 + 1$

a token ring!



Control Law(i): if $s_{i-1} \neq s_i$ *then* $s_i = s_{i-1}$

One enabled rule is chosen per step.

SSSS* Behavior

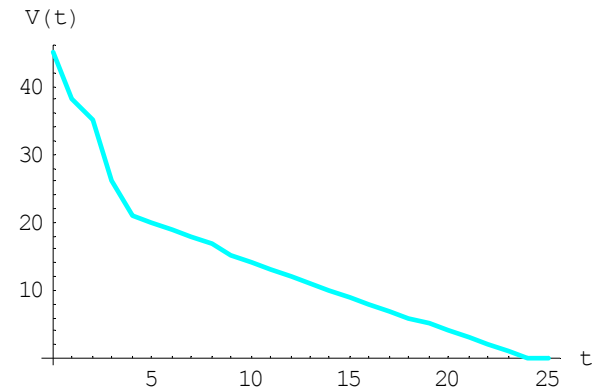
example run

	9	4	6	2	7	8	5	10	3	1
	9	4	6	6	7	8	5	10	3	1
	9	4	6	6	7	8	5	5	3	1
	9	9	6	6	7	8	5	5	3	1
	9	9	6	6	7	7	5	5	3	1
	9	9	6	6	7	7	5	5	3	3
	9	9	6	6	7	7	7	5	3	3
	9	9	6	6	7	7	7	5	5	3
	9	9	6	6	7	7	7	7	5	3
	9	9	6	6	7	7	7	7	7	3
	9	9	9	6	7	7	7	7	7	3
24										
	9	9	9	6	6	7	7	7	7	7
	9	9	9	9	6	7	7	7	7	7
	9	9	9	9	6	6	7	7	7	7
	9	9	9	9	6	6	6	7	7	7
	9	9	9	9	9	6	6	7	7	7
	9	9	9	9	9	6	6	6	7	7
	9	9	9	9	9	6	6	6	6	7
	9	9	9	9	9	6	6	6	6	6
	9	9	9	9	9	9	6	6	6	6
	9	9	9	9	9	9	9	6	6	6
	9	9	9	9	9	9	9	9	6	6
	9	9	9	9	9	9	9	9	9	6
	9	9	9	9	9	9	9	9	9	9
	9	9	9	9	9	9	9	9	9	9

From any initial condition (or any disturbance) the network converges to the situation where all states are the same.

Pf: Let B be the number of blocks and let b_i be the size of block i. Then

$$V(t) = \sum_{i=1}^{B(t)} i b_i(t)$$



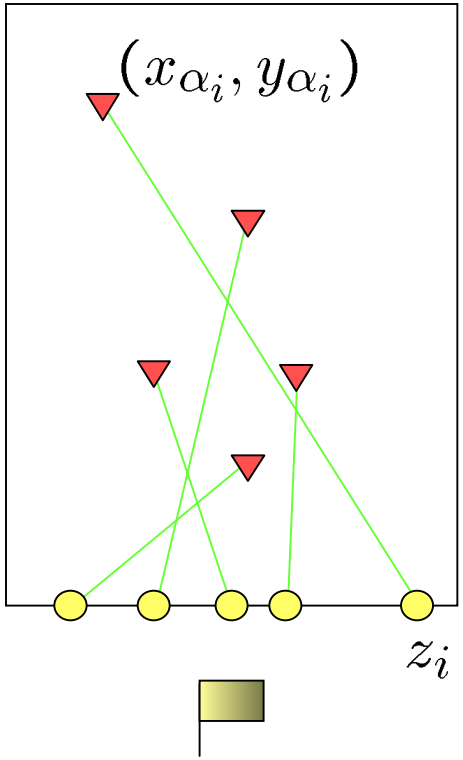
decreases at each (active) step since

$$i b_i + (i+1) b_{i+1} < i(b_i + 1) + (i+1)(b_{i+1} - 1)$$

V is a "Lyapunov function"!

*Super Simple Self Stablizer

RoboFlag Example



Goal: Arrive at a good assignment of defenders to attackers.

$$r_{i,j} = \begin{cases} 1 \text{ if } y_{\alpha_j} < |z_i - x_{\alpha_j}| \\ 0 \text{ else} \end{cases}$$

$$switch(i,j) =$$

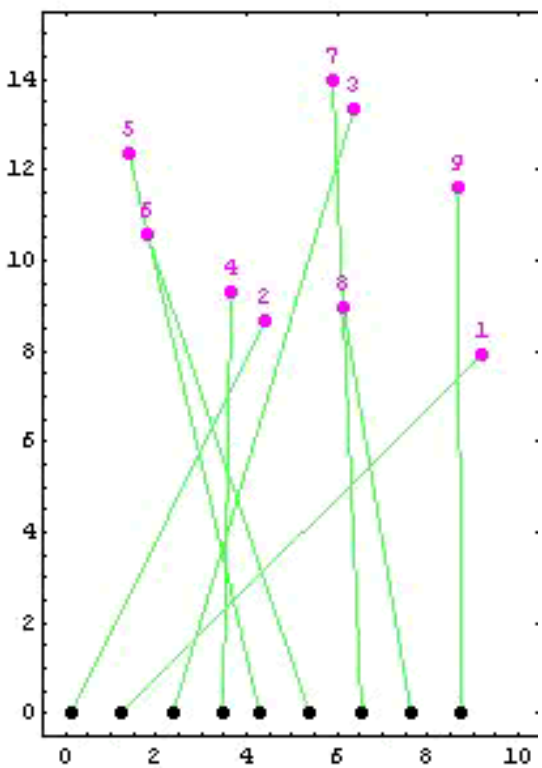
$$r_{i,j} + r_{j,i} < r_{i,i} + r_{j,j} \vee$$
$$(r_{i,j} + r_{j,i} = r_{i,i} + r_{j,j} \wedge x_{\alpha_i} > x_{\alpha_j})$$

The rule for each pair $(i, i{+}1)$ of neighbors is

$$\text{if } switch(i, i+1) \text{ then } (\alpha_i, \alpha_{i+1}) \leftarrow (\alpha_{i+1}, \alpha_i)$$

Problem: find a Lyapunov function (that decreases every time the rule is applied).

RoboFlag Protocol Behavior



Define

$$\gamma_{i,j} = \begin{cases} 1 \text{ if } x_{\alpha_i} > x_{\alpha_j} \\ 0 \text{ else} \end{cases}$$

$$\sigma = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \gamma_{i,j}$$ ..... # conflicts

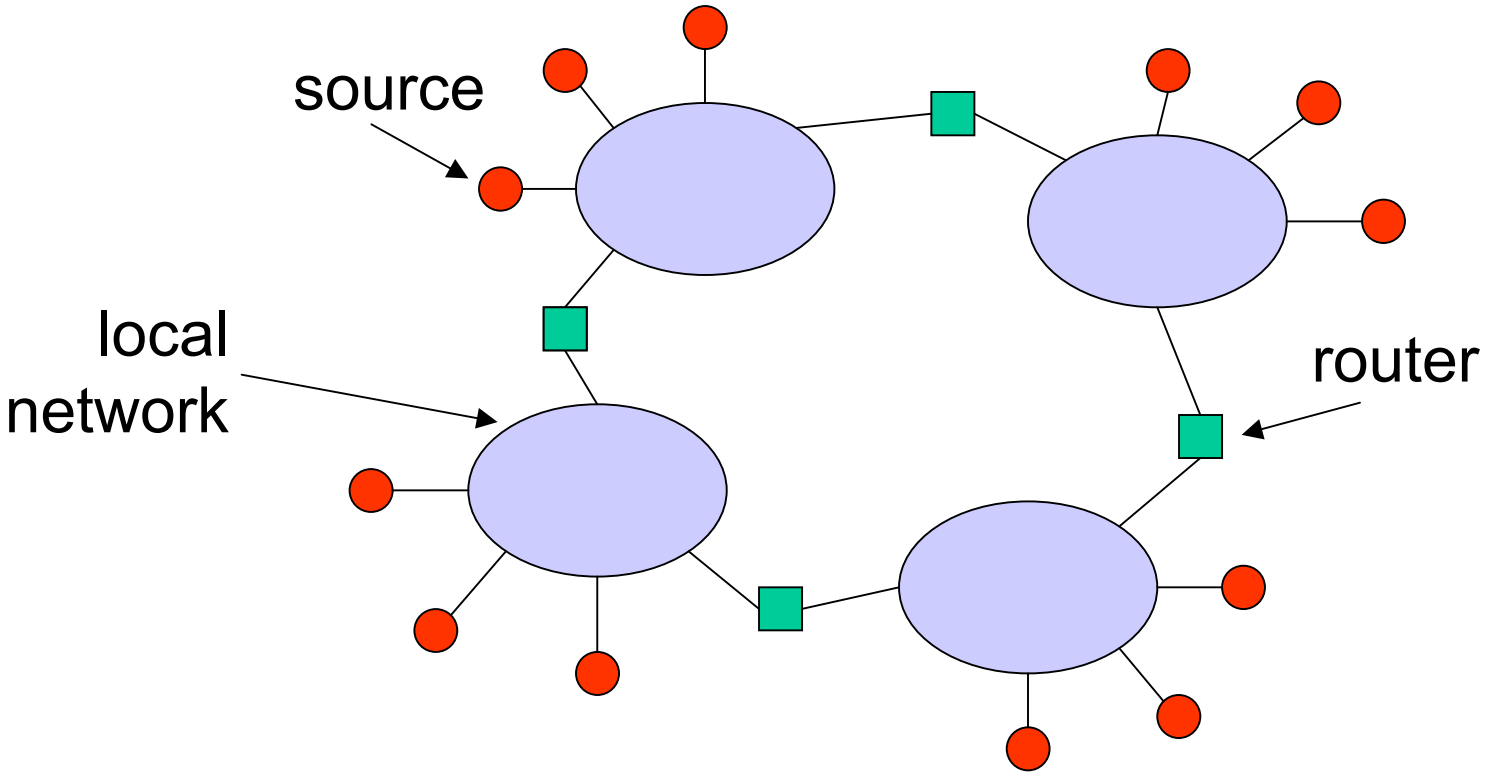
$$\rho = \sum_{i=1}^{n} r_{i,i}$$ ..... # impossible assignments

$$V = \left[\binom{n}{2} + 1 \right] \rho + \sigma$$ ..... Lyapunov function

Every rule application decreases *V*.

Pf: A rule is applied if it decreases ρ, or if it keeps ρ the same and decreases γ.

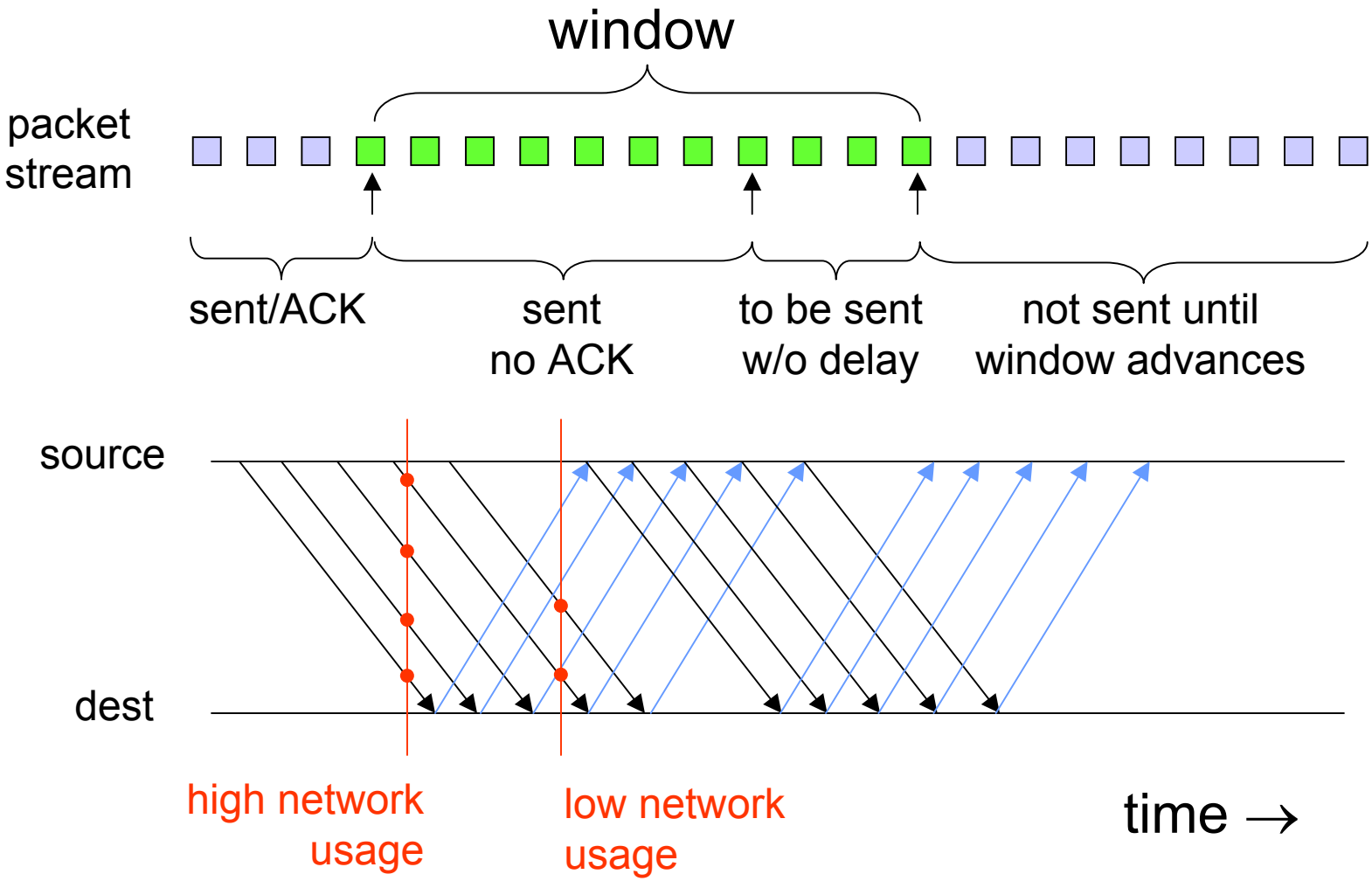
Internet Congestion Control



Goals:

- Reliable transport of data (need ACKs)
- **Fairness**: Each source transmits at same rate
- **Usage**: Network is used at full capacity

Basic TCP, Source Side



How does a source choose the window size? (Vegas, Reno)

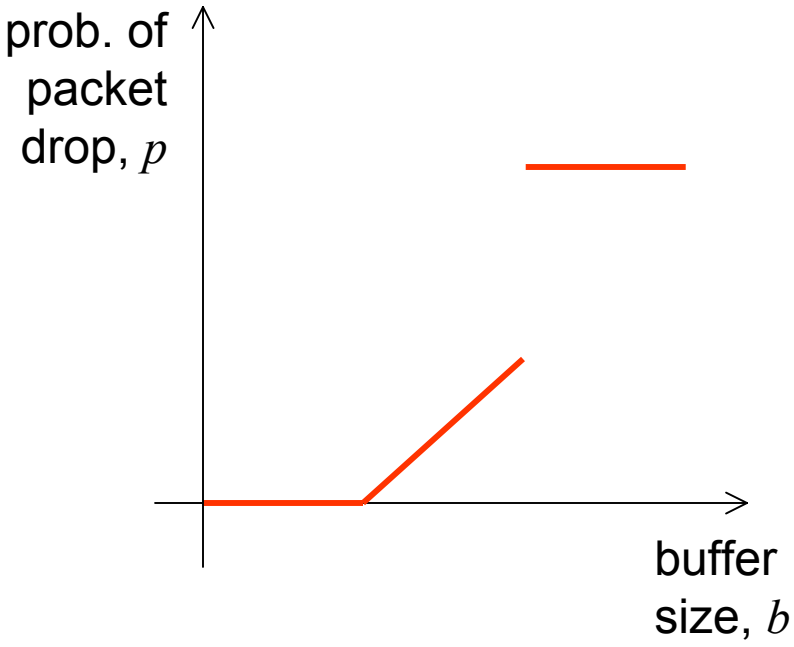
Feedback from the Router

- The router needs a way of communicating the load back to the source.

- The basic way to do this is to drop packets.

- If the source gets no ACK after a reasonable period of time (2*RTT), it determines that the link is congested.

- Of course, this information is <u>delayed</u> because it has to travel through the network too.



AQM = Active Queue Management (we'll assume $p=b$).

What To Do with the Feedback?

TCP algorithms have two main phases:

slowstart -- window size increase by 1 with each ACK until a threshold (w_{max}) is reached.

congestion avoidance -- resize window dynamically based on feedback from the router (dups).

<u>TCP Reno (roughly)</u>

```
congestion mode:
  if ACK
    w ← w + 1/w
  if Loss
    w_max = 0.5 w
    clear window
    goto slowstart
```

<u>TCP Vegas (roughly)</u>

```
congestion mode:
  set w to be proportional
  to the estimated queuing
  delay (which goes like Δb),
  otherwise like Reno.
```

How can we determine the performance of these control laws?

A Continuous Model

(One link, n sources)

Source rates (packets/s):

$$x_i, \text{ for } i = 1, \ldots, n$$

Source rate control:

$$\dot{x}_i = F(x_i, b, \dot{b})$$

Aggregate flow through link:

$$y = \sum_{i=1}^{n} x_i$$

Link dynamics:

$$\dot{b} = y - c$$

link capacity

More realistic source rate control:

$$\dot{x}_i(t) = F(x_i(t), b(t - \tau^b(t)), \dot{b}(t - \tau^b(t)))$$

delay due to network

Try 0: A Simple Approach

Q: Why not have all rates set to c/n?
A: c and n may not be known, or may change!

In general, constant rates may guarantee fairness, but will under or over utilize the network.

Try 1: Something Like TCP Vegas

If $y < c$, increase, else decrease:

$$\dot{x}_i = -y + c = -\dot{b}$$

The equilibrium is thus any $(x_1,..., x_n)$ such that

$$\sum_{i=1}^{n} x_i = c$$

The set of all such points is a **neutrally stable submanifold** of R^n (1 negative and n-1 zero eigenvalues).

Thus, the algorithm is <u>not fair</u>, although it does use the entire capacity of the link.

Try 2: More Like TCP Vegas

Add an element of greed:

$$\dot{x}_i = -y + c + \frac{k}{x_i} = -\dot{b} + \frac{k}{x_i}$$

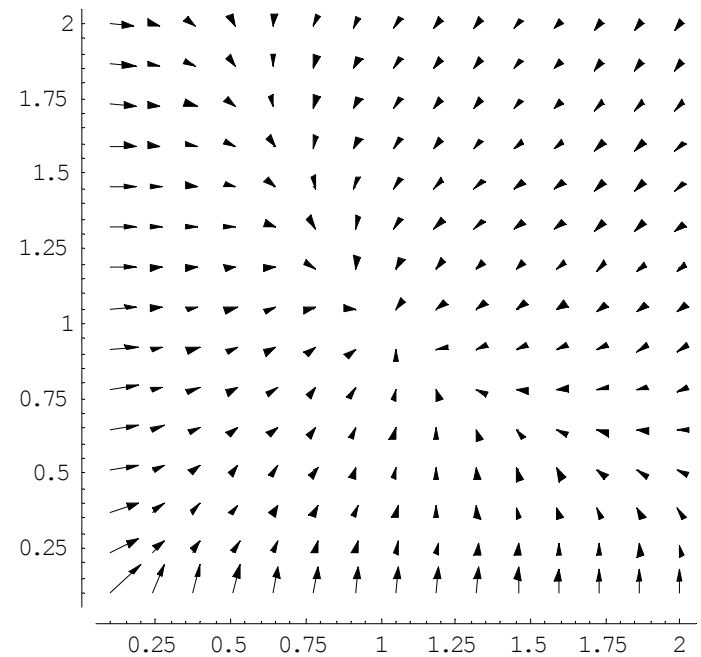
Solving the above for 0, gives

$$x_i^* = \frac{c + \sqrt{c^2 + 4kn}}{2n}$$

Linearizing around this point gives eigenvalues:

$$-n - \frac{k}{(x_i^*)^2} \text{ with multiplicy } 1$$

$$-\frac{k}{(x_i^*)^2} \text{ with multiplicy } n-1$$



n=2 case

How to Choose k

Consider

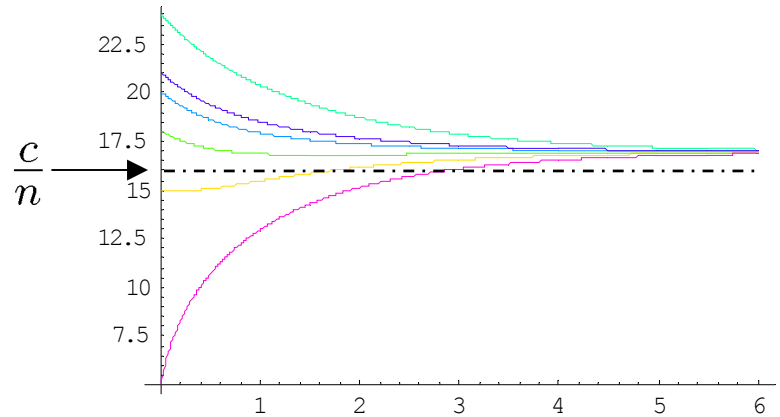
$$x_i^* = \frac{c + \sqrt{c^2 + 4kn}}{2n}$$

Setting $k=0$ gives $x^*=c/n$, but then we get Try 1 again!

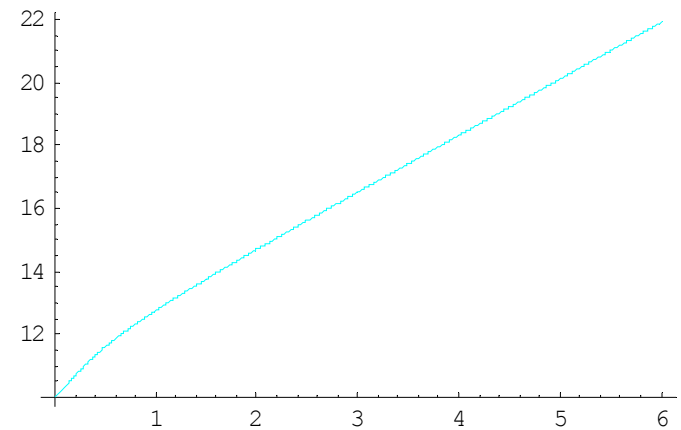
Setting $k>0$, gives $x^*>c/n$ so that $y>c$.

Thus, b grows without bound (until b_{max} is reached, where a different model holds).

So we get fairness and **slightly more than full capacity**!



rates



buffer size

Try 3: Something Like Reno

A different approach:

$$\dot{x}_i = \underbrace{-b\frac{x_i^2}{2}}_{\text{exponential backoff}} + \underbrace{(b_{max} - b)}_{\text{additive increase, } k \text{ is the buffer size}}$$

If $b=0$, slowly increase x_i.
If $b>0$, quickly decrease x_i.

Equilibrium (stable):

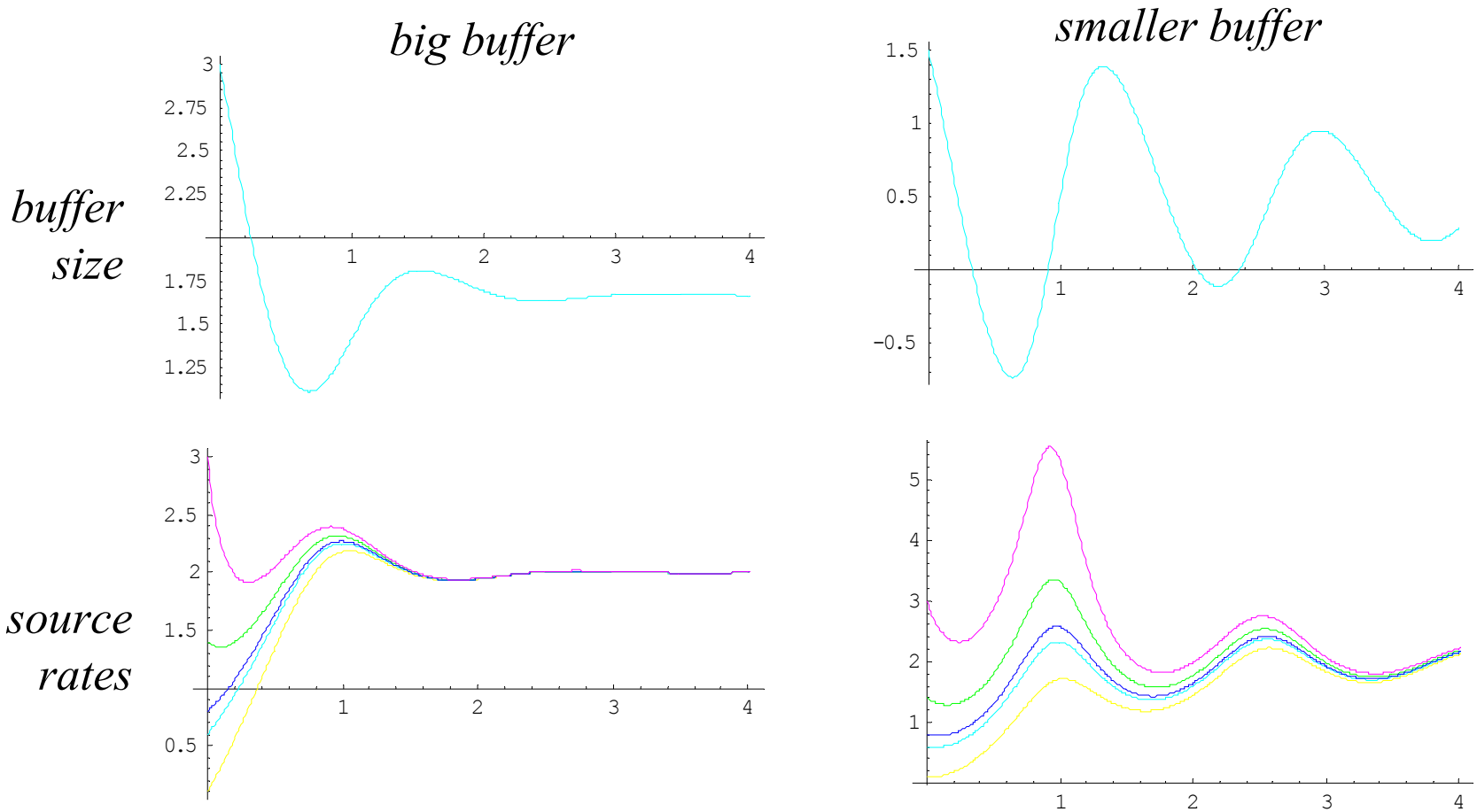
$$x_i^* = \frac{c}{n}$$

It's fair and uses full capacity!

$$b^* = \frac{b_{max}}{1 + \frac{c^2}{2n^2}}$$

Equilibrium buffer size is feasible!

Behavior of Reno

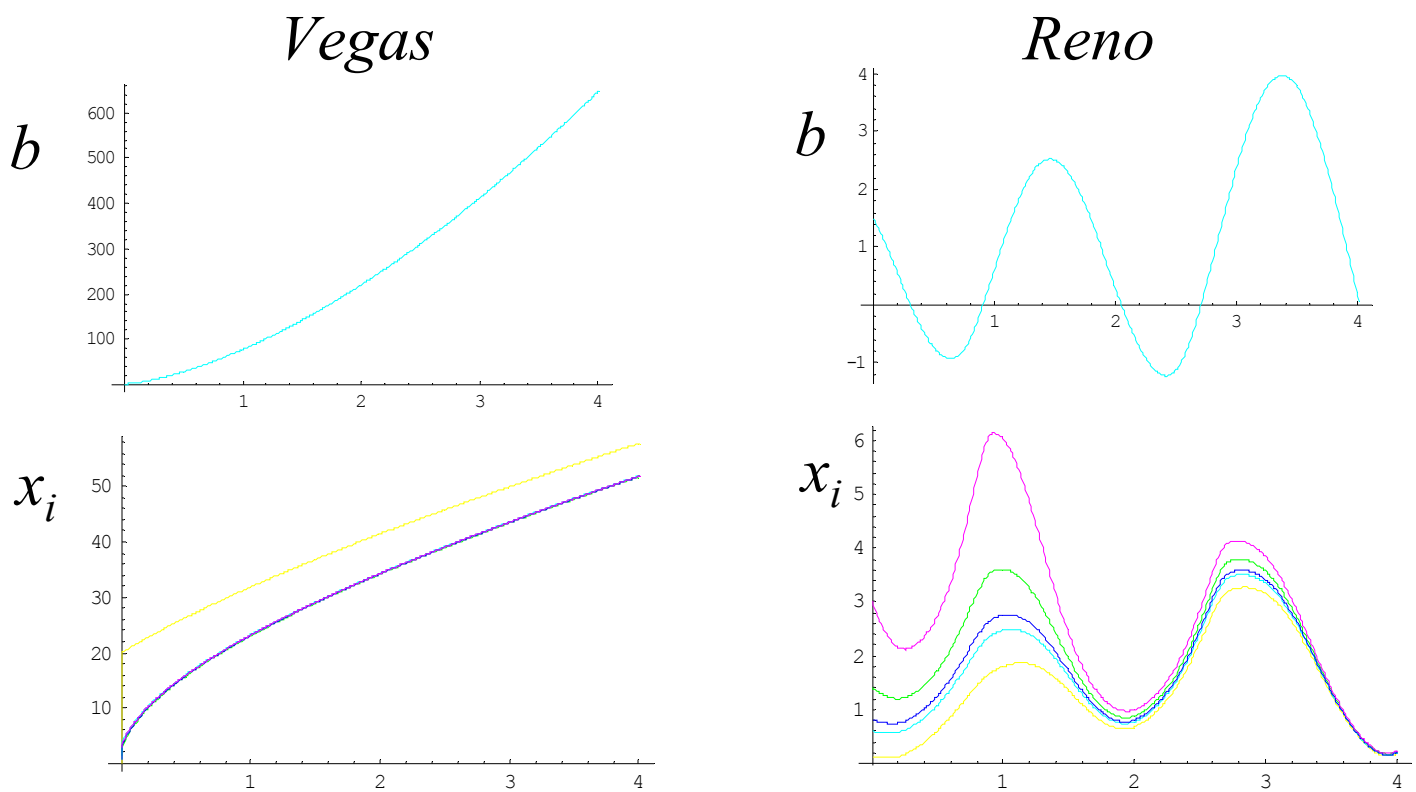


TCP Vegas was actually developed to address the oscillatory behavior of TCP Reno.

Effects of Delay

We model the delay in terms of buffer size and capacity.

$$\tau(t) = \frac{b}{c} \qquad \dot{x}_i = F(x_i, b(t-\tau), \dot{b}(t-\tau))$$



For More Information

Current research in congestion control uses more advanced techniques (such as economic models). See for example

Ki Baek Kim (`http://cisl.snu.ac.kr/~kkb`)

- My office mate: Thanks for the help!

Prof. Steven Low (`http://netlab.caltech.edu`)

FAST = Fast AQM Scalable TCP
(Ultrascale networking project)

Conclusions

Control and control-like ideas are common in CS.

We looked at two examples.

- Self stabilization
- Congestion control

What other roles are there for control techniques in CS?

- Robust software?