Notes on using RED for Queue Management and Congestion Avoidance

Van Jacobson
van@ee.lbl.gov
Network Research Group
Berkeley National Laboratory
Berkeley, CA 94720

NANOG 13
Dearborn, MI

June 8, 1998

Acknowledgment

Most of what's described here is based on recent work done with Kathleen Nichols (`knichols@baynetworks.com`) and Kedarnath Poduri (`kpoduri@baynetworks.com`) of the Bay Architecture Lab.

There's a lot of analysis, simulation and measurement that we soon hope to turn into a paper giving an engineering and operations perspective on RED. (Kathie's in charge of this so it will probably happen.)

At NANOG 11 (Phoenix) I offered a “heads up” that a large crowd of researchers were about to say that some sort of active router queue management was necessary for the health of the Internet. That document is now out:

> **RFC2309: Recommendations on Queue Management and Congestion Avoidance in the Internet.** *B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang.* April 1998.

That was about *why*. This is about *how*.

A brief digression

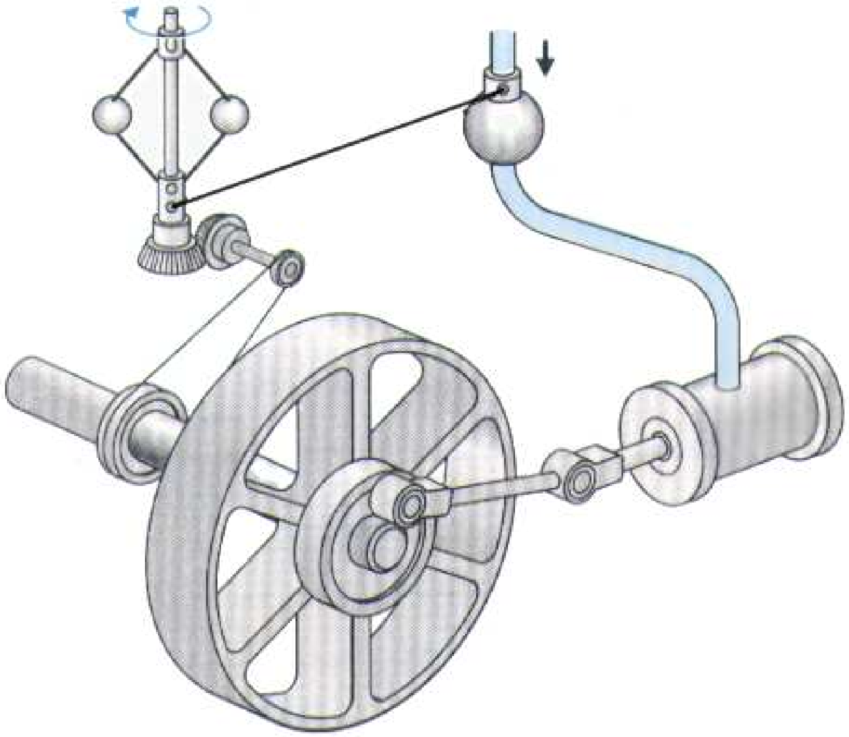
In history you may have learned that in 1794 James Watt invented the steam engine and started the Industrial Revolution. This is false.

Actually, steam engines had been around for 2000 years—Hero of Alexandria wrote about them in 200 BC—but they were only useless toys.

James Watt made them useful by inventing the Flyball Regulator.

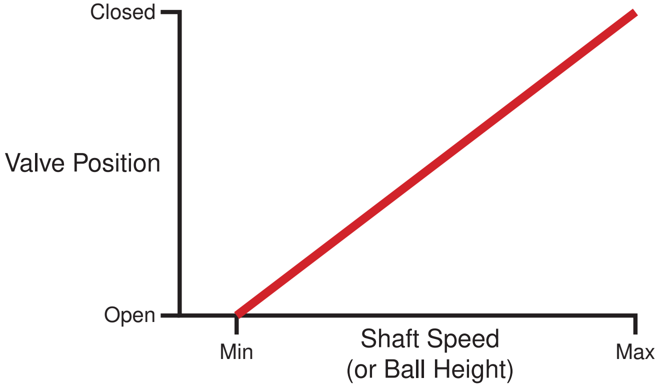
Digression (cont.)

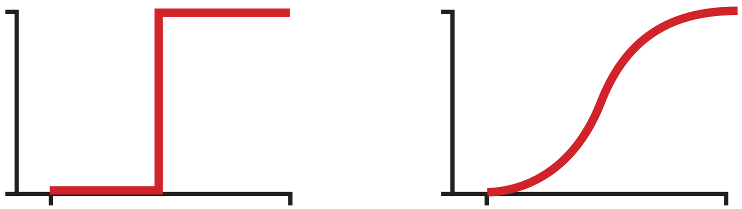
The Flyball Regulator allowed an engine to be used with real-world (changing) loads because it adjusted the steam pressure automatically based on the main drive-shaft velocity.



Digression (cont.)

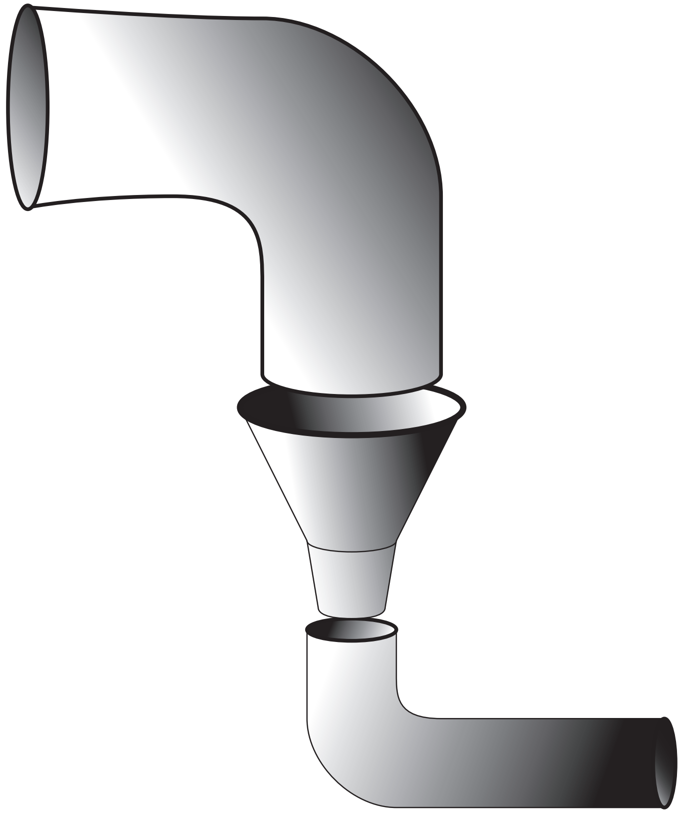
How the regulator's controlling variables affect its controlled variables is usually called the *Control Law*. To design (or understand) a regulator, understand its control law:





But because this is a closed-loop servo system, *any* control law that meets simple monotonicity and completeness constraints will work. Some just work better than others.

Getting back to networks. . .

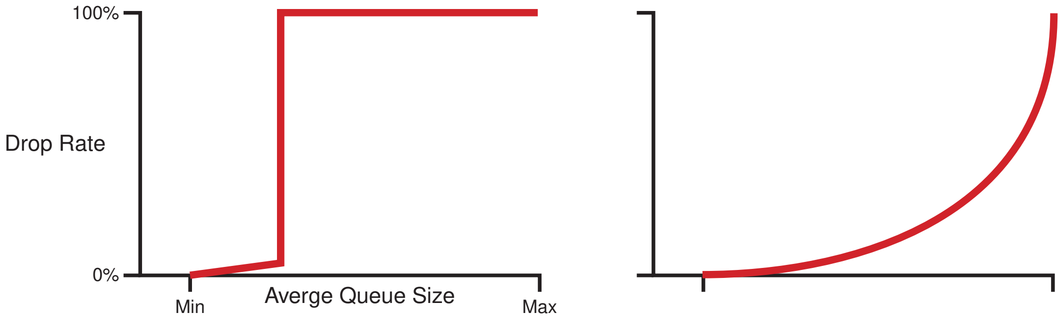


A typical congested gateway looks like a firehose connected to a soda straw through a small funnel (the output queue).

If, on average, packets arrive faster than they can leave, the funnel will fill up and eventually overflow.

RED is simple regulator that monitors the level in the funnel and uses it to match the input rate to the output (by dropping excess traffic).

As long as its control law is monotone non-decreasing and covers the full range of 0 to 100% drop rate, RED works for *any* link, *any* bandwidth, *any* type of traffic.



RED works even when the control law is bizarre. But it works really well when the control law incorporates the additional leverage caused by TCP's congestion avoidance and timeout algorithms.

Other design considerations

100ms in the life of a T1 as envisioned by a typical academic (horizontal axis is time, colored boxes are packets, white are gaps, different colors are different “flows”):

The same 100ms as observed by a network monitor:

Note that real traffic is both *very bursty* and *highly structured:*

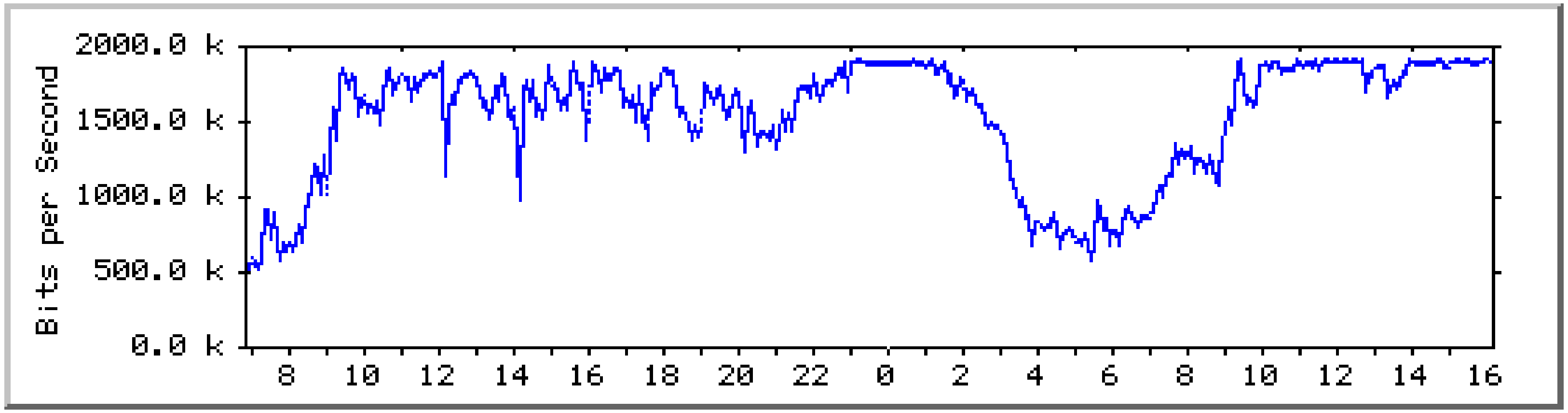
$\Rightarrow$ Have to average over relatively long time scales to find the real queue length.

$\Rightarrow$ Have to randomize drops so they’re not biased by the traffic structure.

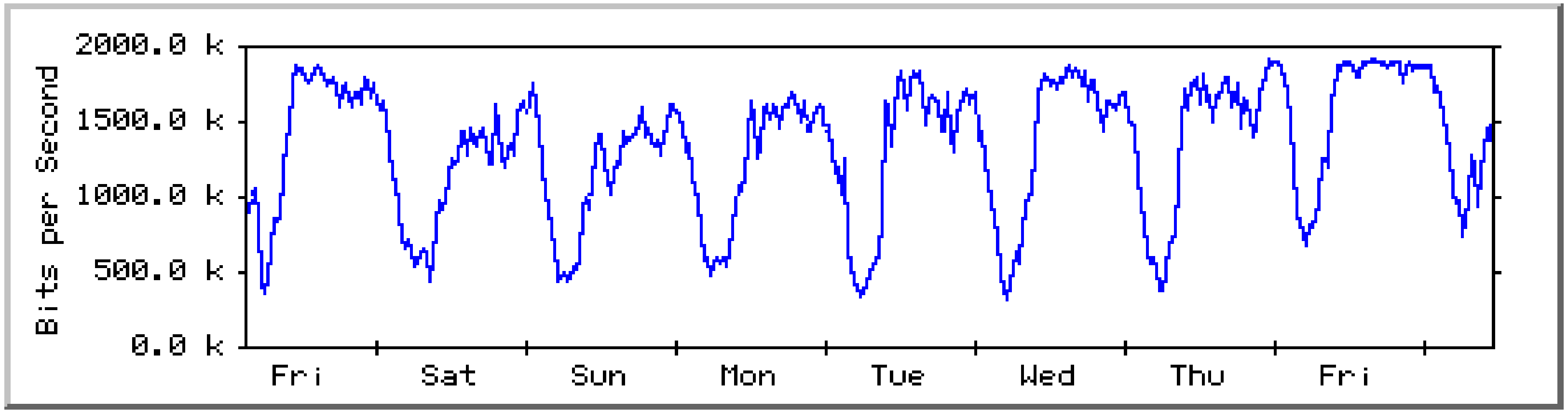
Nice theory — but does RED make any practical difference?

Data from a busy EBone E1 (2Mb/s) link courtesy of Sean Doran.

RED was turned on at 10am Friday (the 2nd “10” in the figure). Note that the utilization goes up to 100% and stays there but the FIFO queuing on Thursday hardly ever makes it up to 100%.



Maybe Thursday was just a bad day so take a look at the rest of the week:



Utilization is low because FIFO dropping interacts badly with all the structure in internet traffic:

- tends to drop multiple packets from the same conversation
- tends to drop from small users rather than big ones
- tends to drop (multiple) TCP SYNs

But RED is too complicated for a high speed router. . .

No. The entire RED algorithm is only 20 lines of C. And even that doesn't have to be done per-packet. RED is driven off long-term *average* behavior and will work just fine if run as a low duty-cycle background task.

RED requires exactly one addition to the forwarding path (in the place where an arriving packet is added to the queue):

```
if (--packetsTillDrop == 0)
    DropPacket();
```

Complication (cont.)

But the 1993 RED paper wasn't very clear on this (because of an editing error, its "efficient" implementation isn't). And implementing a distributed algorithm on the specialized multi-tasking multi-processor that forms a modern router is inherently architecture specific.

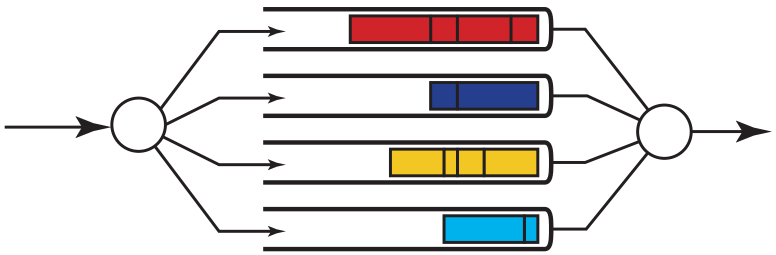
To help get RED deployed, I'd be glad work with any router vendor seriously interested in implementing it. (This offer has stood for several years. So far only Bay has taken me up on it.)

What about WRED, FRED, RIO, ARED,. . . ?

Maybe. But be suspicious. All of these are basically drop-preference schemes which means they're driven off the detailed sub-structure of a queue. Queues have a complex, time-varying structure and it is *very* hard to extract reliable information from them.

If general, the same per-packet information used to decide the drop preference in a queue:

can be used to instead select one of a set of queues:



It is almost always better to run one instance of RED over each of these queues rather than multiple instances of RED over one big queue.

Conclusion

- RED is a simple network traffic regulator based on 200 years control engineering experience.
- It works *much* better than tail-drop (or head-drop) FIFO queues.
- It works in practice, not just theory.
- A RED implementation adds about 5 instructions (or < 200 gates) to the forwarding path.

But the original RED paper was missing a lot of important operational information and suggested a control law that isn't a good match to how the Internet has evolved. The pioneering vendor implementations have suffered from our mistakes. We're currently trying to fix that.

Soon…