A Fairer, Faster Internet Protocol

By **Bob Briscoe**

ILLUSTRATION: QUICKHONEY

The Internet is founded on a very simple premise: shared communications links are more efficient than dedicated channels that lie idle much of the time.

And so we share. We share local area networks at work and neighborhood links from home. And then we share again—at any given time, a terabit backbone cable is shared among thousands of folks surfing the Web, downloading videos, and talking on Internet phones.

But there's a profound flaw in the protocol that governs how people share the Internet's capacity. The protocol allows you to seem to be polite, even as you elbow others aside, taking far more resources than they do.

Network providers like Verizon and BT either throw capacity at the problem or improvise formulas that attempt to penalize so-called bandwidth hogs. Let me speak up for this much-maligned beast right away: bandwidth hogs are not the problem. There is no need to prevent customers from downloading huge amounts of material,

so long as they aren't starving others.

Rather than patching over the problem, my colleagues and I at BT (formerly British Telecom) have worked out how to fix the root cause: the Internet's sharing protocol itself. It turns out that this solution will make the Internet not just simpler but much faster too.

You might be shocked to learn that the designers of the Internet intended that your share of Internet capacity would be determined by what your own software considered fair. They gave network operators no mediating role between the conflicting demands of the Internet's hosts—now over a billion personal computers, mobile devices, and servers.

The Internet's primary sharing algorithm is built into the Transmission Control Protocol, a routine on your own computer that most programs run—although they don't have to. TCP is one of the twin pillars of the Internet, the other being the Internet Protocol, which delivers packets of data to particular addresses. The two together are often called TCP/IP.

Your TCP routine constantly increases your transmission rate until packets fail to get through some pipe up ahead—a tell-tale sign of congestion. Then TCP very politely halves your bit rate. The billions of other TCP routines around the Internet behave in just the same way, in a cycle of taking, then giving, that fills the pipes while sharing them equally. It's an amazing global outpouring of self-denial, like the "after you" protocol two people use when they approach a door at the same time—but paradoxically, the Internet version happens between complete strangers, even fierce commercial rivals, billions of times every second.

The commercial stakes could hardly be higher. Services like YouTube, eBay, Skype, and iTunes are all judged by how much Internet capacity they can grab for you, as are the Internet phone and TV services provided by the carriers themselves. Some of these companies are opting out of TCP's sharing regime, but most still allow TCP to control how much they get—about 90 percent of the 200 000 terabytes that cross the Internet each second.

This extraordinary spirit of global cooperation stems from the Internet's early history. In October 1986, Internet traffic persistently overran available capacity—the first of a series of what were called congestion collapses. The TCP software of the day continued to try to retransmit, aggravating the problem and causing everyone's throughput to plummet for hours on end. By mid-1987 Van Jacobson, then a researcher at Lawrence Berkeley National Laboratory, had coded a set of elegant algorithms in a patch to TCP. (For this he received the IEEE's prestigious Koji Kobayashi Computers and Communications Award in 2002.)

Jacobson's congestion control accorded well with the defining design principle of the Internet: traffic control is consigned to the computers around the edges of the Internet (using TCP), while network equipment only routes and forwards packets of data (using IP).

The combination of near-universal usage and academic endorsement has gradually elevated TCP's way of sharing capacity to the moral high ground, altering the very language engineers use. From the beginning, equal rates were not just "equal," they were "fair." Even if you don't use TCP, your protocol is considered suspect if it's not "TCP-friendly"—a cozy-sounding idea meaning it consumes about the same bit rate as TCP would.

Sadly, an equal bit rate for each data flow is likely to be extremely unfair, by any realistic definition. It's like insisting that boxes of food rations must all be the same size, no matter how often each person returns for more or how many boxes are taken each time.

Consider a neighborhood network with 100 customers, each of whom has a 2-megabit-per-second access line connected to a single shared 10 Mb/s regional link. The network provider can get away with such a thin shared pipe because most of the customers—let's say 80 of the 100—don't use it continuously, even over the peak period. These people might think they are constantly clicking at their browsers and getting new e-mail, but their data transfers might be active perhaps only 5 percent of the time.

However, there are also 20 heavy users who download continuously, perhaps using file-sharing programs that run unattended. So at any one moment, data is flowing to about 24 users—all 20 heavy users, and 4 of the 80 light ones. TCP gives 20 shares of the bottleneck capacity to the heavy users and only 4 to the light ones. In a few moments, the 4 light users will have stepped aside and another 4 will take over their shares. However, the 20 heavy users will still be there to claim their next 20 shares. They might as well have dedicated circuits!

It gets even worse. Any programmer can just run the TCP routine multiple times to get multiple shares. It's much like getting around a food-rationing system by duplicating ration coupons.

This trick has always been recognized as a way to sidestep TCP's rules—the first Web browsers opened four TCP connections. Therefore, it would have been remarkable if this ploy had not become more common.

A number of such strategies evolved through innocent experimentation. Take peer-to-peer file sharing—a common way to exchange movies over the Internet, one that accounts for a large portion of all traffic. It involves downloading a file from several peers at once. This parallel scheme, sometimes known as swarming, had become routine by 2001, built into such protocols as BitTorrent.

The networking community didn't immediately view connecting with many machines as a circumvention of the TCP-friendliness rule. After all, each transfer used TCP, so each data flow "correctly" got one share of any bottleneck it encountered. But using parallel connections to multiple machines was a new degree of freedom that hadn't been thought of when the rules were first written. Fairness should be defined as a relation between people, not data flows.

Peer-to-peer file sharing exposed both of TCP's failings. First, a file-sharing program might be active 20 times as often as your Web browser, and second, it uses many more TCP connections, typically 5 or even 50 times as many. Peer-to-peer thus takes 100 or 1000 times as many shares of Internet bottlenecks as a browser does.

Returning to our 100 broadband customers: if they were just browsing the Web and exchanging e-mail, each would get nearly the full benefit of a 2 Mb/s access pipe—if 5 customers were active at a time, they'd just squeeze into the 10Mb/s shared pipe. But if even 20 users started continuous parallel downloading, the TCP algorithm would send everyone else's bit rate plummeting to an anemic 20 kilobits per second—worse than dial-up! The problem

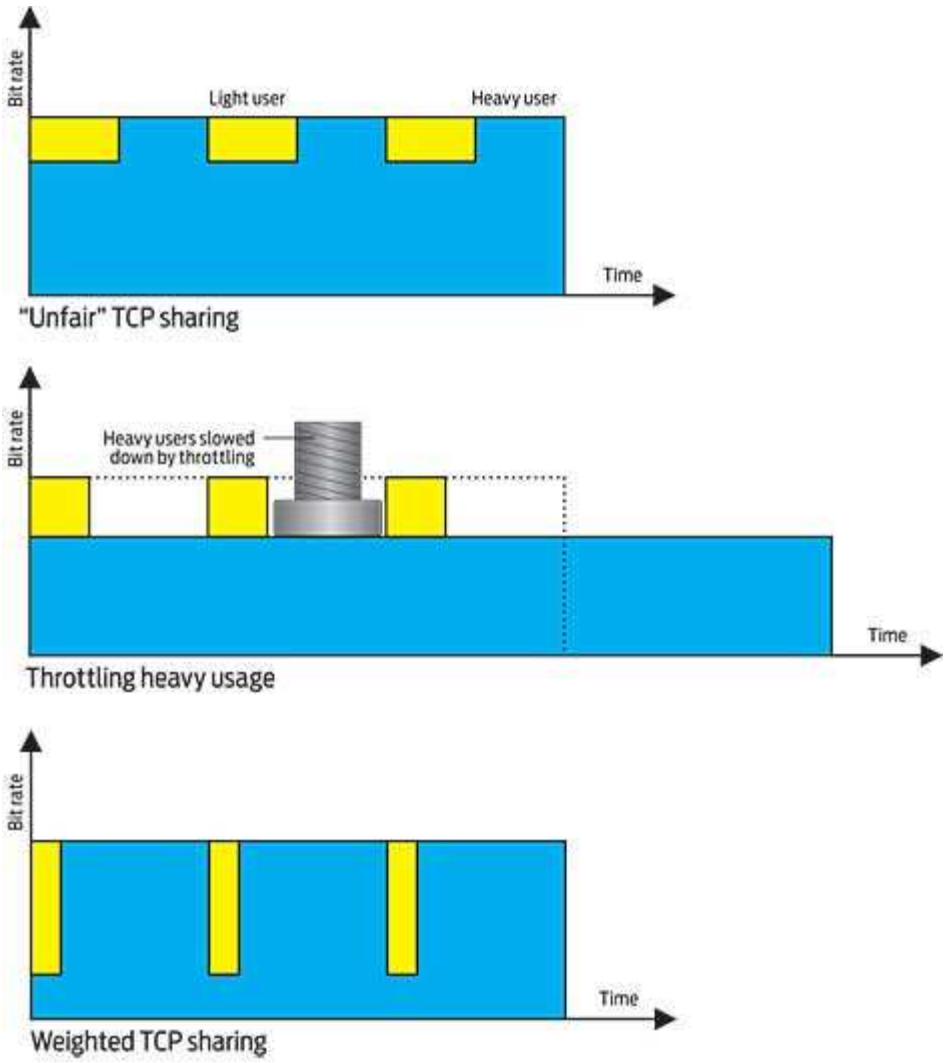
isn't the peer-to-peer protocols; it's TCP's sharing rules.

Why can't the service provider simply upgrade that stingy 10 Mb/s shared pipe? Of course, some upgrades are necessary from time to time. But as a general approach to the problem of sharing, adding capacity is like throwing water uphill.

Imagine two competing Internet service providers, both with this 80:20 mix of light and heavy users. One provider quadruples its capacity; the other doesn't. But TCP still doles out the upgrader's capacity in the same way. So the light users, who used to have a measly 20 kb/s share, now get a measly 80 kb/s—still barely better than dial-up. But now the 80 light users must pay substantially more for four times the long-distance capacity, which they hardly get to use. No rational network operator would upgrade under these conditions—it would lose most of its customers.

But there is plenty of evidence that Internet service providers are continuing to add capacity. This is partly explained by government subsidies, particularly in the Far East. Equivalently, weak competition, typical in the United States, allows providers to fund continued investment through higher fees without the risk of losing customers. But in competitive markets, common in Europe, service providers have had to attack the root cause: the way their capacity is shared.

Network providers often don't allow TCP to give all the new capacity straight to the heavy users. Instead they impose their own sharing regimes on their customers, thus overriding the worst effects of TCP's broken regime. Some limit, or "throttle," the peak-time bit rate of the peer-to-peer customers. Others partition the pipe to prevent heavy users encroaching on lighter ones. Increasingly, the share of Internet capacity you actually get is the result of this tussle between TCP and the service providers' allocation schemes.



THROTTLE THIS: Throttling tries to correct today's TCP system [left] by clamping down on heavy users [center], but the technique misses a trick. With weighted TCP sharing [right], light users can go superfast, so they finish sooner, while heavy users slow only fleetingly, then catch up. All this can be done without any prioritization in the network.

There's a far better solution than fighting. It would allow light browsing to go blisteringly fast but hardly prolong heavy downloads at all. The solution comes in two parts. Ironically, it begins by making it easier for programmers to run TCP multiple times—a deliberate break from TCP-friendliness.

Programmers who use this new protocol to transfer data will be able to say "behave like 12 TCP flows" or "behave like 0.25 of a TCP flow." They set a new parameter—a weight—so that whenever your data comes up against others all trying to get through the same bottleneck, you'll get, say, 12 shares, or a quarter of a share. Remember, the network did not set these priorities. It's the new TCP routine in your own computer that uses these weights to control the number of shares it takes from the network.

At this point in my argument, people generally ask why everyone won't just declare that they each deserve a huge weight. The answer to the question involves a trick that gives everyone good reason to use the weights sparingly—a trick I'll get to in a minute. But first, let's check how this scheme ensures the lightning-fast browsing rates I just promised.

The key is to set the weights high for light interactive usage, like surfing the Web, and low for heavy usage, such as movie downloading. Whenever these uses conflict, flows with the higher weighting—those from the light users—will go much faster, which means they will also finish much sooner. Then the heavy flows can expand back to a higher bit rate sooner than otherwise. This is why the heavy flows will hardly take any longer to complete. The weighting scheme uses the same strategy as a restaurant manager who says, "Get those individual orders out right away, then come serve this party of 12." But today's Internet has the balance of the weights exactly the other way around.

That brings us to the second part of the problem: how can we encourage everyone to flip the weights? This task means grappling with something that is often called "the tragedy of the commons." A familiar example is global warming, where everyone happily pursues what's best for them—leaving lights on, driving a big car—despite the effect this may have on everyone else through the buildup of carbon dioxide and other greenhouse gases.

On the Internet, what matters isn't how many gigabytes you download but how many you download when everyone else is trying to do the same. Or, more precisely, it's the volume you download weighted by the prevailing level of congestion. Let's call this your congestion volume, measured in bytes. Think of it as your carbon footprint for the Internet.

As with CO2, the way to cut back is to set limits. Imagine a world where some Internet service providers offer a deal for a flat price but with a monthly congestion-volume allowance. Note that this allowance doesn't limit downloads as such; it limits only those that persist during congestion. If you used a peer-to-peer program like BitTorrent to download 10 videos continuously, you wouldn't bust your allowance so long as your TCP weight was set low enough. Your downloads would draw back during the brief moments when flows came along with higher weights. But in the end, your video downloads would finish hardly later than they do today.

On the other hand, your Web browser would set the weights high for all its browsing because most browsing comes in intense flurries, and so it wouldn't use up much of your allowance. Of course, server farms or heavy users could buy bigger congestion quotas, and light users might get Internet access with a tiny congestion allowance—for a lower flat fee.

But there's a snag. Today Internet service providers can't set congestion limits, because congestion can easily be hidden from them. As we've said, Internet congestion was intended to be detected and managed solely by the computers at the edge—not by Internet service providers in the middle. Certainly, the receiver does send feedback messages about congestion back to the sender, which the network could intercept. But that would just encourage the receiver to lie or to hide the feedback—you don't have to reveal anything that may be used as evidence against you.

Of course a network provider does know about packets it has had to drop itself. But once the evidence is destroyed, it becomes somewhat tricky to hold anyone responsible. Worse, most Internet traffic passes through multiple network providers, and one network cannot reliably detect when another network drops a packet.

Because Internet service providers can't see congestion volume, some limit the straight volume, in gigabytes, that each customer can transfer in a month. Limiting total volume indeed helps to balance things a little, but limiting congestion volume does much

better, providing extremely fast connections for light users at no real cost to the heavy users.

My colleagues and I have figured out a way to reveal congestion so that limits can be enforced. We call it "refeedback" [see "

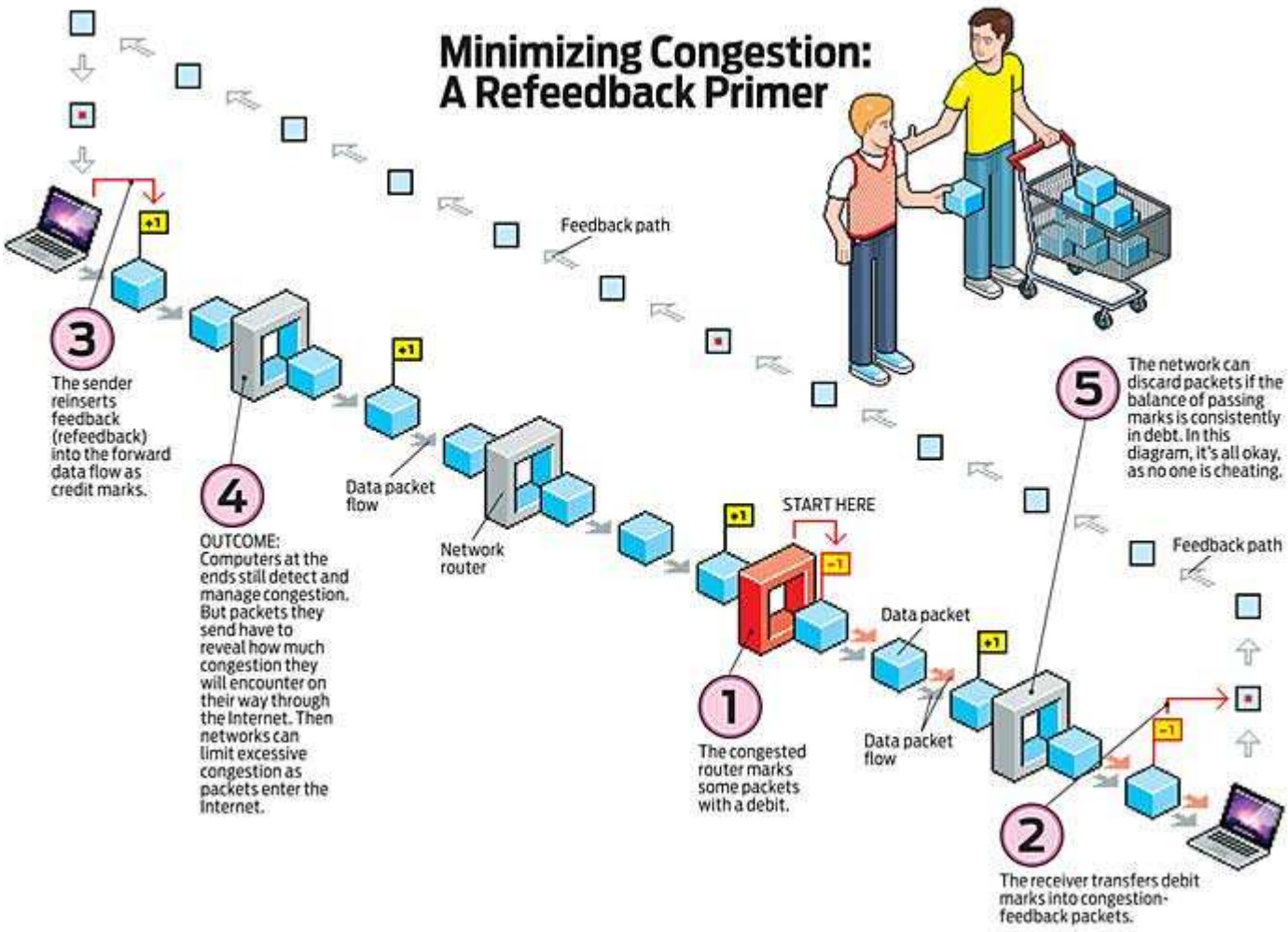


ILLUSTRATION: QUICKHONEY

." Here's how it works. Recall that today the computers at each end of an exchange of packets see congestion, but the networks between them can't. So we built on a technique called Explicit Congestion Notification—the most recent change to the TCP/IP standard, made in 2001. Equipment that implements that change marks packets during impending congestion rather than doing nothing until forced to drop them. The marks—just a change in a single bit—let the network see congestion directly, rather than inferring it from gaps in the packet stream. It's also particularly neat to be able to limit congestion before anyone suffers any real impairment.

Although the 2001 reform reveals congestion, it is only visible downstream of any bottleneck as packets leave the network. Our scheme of refeedback makes congestion visible to the upstream network before it enters the Internet, where it can be limited.

Refeedback introduces a second type of packet marking—think of these as credits and the original congestion markings as debits. The sender must add sufficient credits to packets entering the

network to cover the debit marks that are introduced as packets squeeze through congested Internet pipes. If any subsequent network node detects insufficient credits relative to debits, it can discard packets from the offending stream.

To keep out of such trouble, every time the receiver gets a congestion (debit) mark, it returns feedback to the sender. Then the sender marks the next packet with a credit. This reinserted feedback, or refeedback, can then be used at the entrance to the Internet to limit congestion—you do have to reveal everything that may be used as evidence against you.

Refeedback sticks to the Internet principle that the computers on the edge of the network detect and manage congestion. But it enables the middle of the network to punish them for providing misinformation.

The limits and checks on congestion at the borders of the Internet are trivial for a network operator to add. Otherwise, the refeedback scheme does not require that any new code be added to the network's equipment; all it needs is that standard congestion notification be turned on. But packets need somewhere to carry the second mark in the "IP" part of the TCP/IP formula. Fortuitously, this mark can be made, because there is one last unused bit in the header of every IP packet.

In 2005, we prepared a proposal documenting all the technical details and presented it to the Internet Engineering Task Force (IETF), the body that oversees Internet standards.

At this point, the story gets personal. Because I had set myself the task of challenging the entrenched principle of TCP-friendliness—equality of flow rates for all TCP connections—I decided to talk only about the change to IP, omitting any mention of weighted TCP. Instead I played up some other motivations for adding refeedback to IP. I even showed how refeedback could enforce equal flow rates—pandering to my audience's faith while denying my own. But I just looked like yet another mad researcher pushing a solution without a problem.

After a year of banging my head against a wall, I wrote an angry but—I trust—precise attack on the dogma that equal flow rates were "fair." My colleagues got me to tone it down before I posted it to the IETF; evidently I'd softened it enough at least to be invited to present my ideas at a plenary session in San Diego late in 2006. The next day, a nonbinding straw poll of the large audience showed widespread doubt about using TCP-friendliness as a definition of fairness. Elwyn Davies of the Internet Architecture Board e-mailed me, saying, "You have identified a real piece of myopia in the IETF."

I was hardly the first to challenge these myths. In 1997 Frank P. Kelly, a professor at the University of Cambridge, put together some awe-inspiringly elegant and concise mathematical arguments to prove that the same weighted sharing would maximize the value that users get from their Internet throughput. However, to create the right incentives, he proposed varying the prices charged for the packets as they were received, and everyone balked. People like to control, in advance, what they will pay.

Objections to Kelly's pricing scheme blinded the Internet community to all the other insights in his work—particularly the message that equalizing flow rates was not a desirable goal. That's why my team built the refeedback mechanism around his earlier ideas—to limit congestion within flat fees, without dynamic pricing.

Everyone's subsequent obsession with bandwidth hogs, and thus with volume, is also misdirected. What matters is congestion volume—the CO2 of the Internet.

Meanwhile, our immediate task is to win support in the Internet community for limiting congestion and for a standards working group at the IETF to reveal the Internet's hidden congestion. The chosen mechanism may be refeedback, but I won't be miffed if something better emerges, so long as it makes the Internet as simple and as fast.

About the Author

BOB BRISCOE describes how to ease Internet congestion by remaking the way we share bandwidth in "A Fairer, Faster Internet" [p. 42 He says the problem isn't bandwidth hogs but the Internet's sharing protocol itself. Briscoe is chief researcher at BT's Networks Research Centre, in England. He is working with the Trilogy Project to fix the Internet's architecture.

To Probe Further

Details on Internet fairness and the refeedback project can be found at http://www.cs.ucl.ac.uk/staff/bbriscoe/projects/refb/.