How to Kill the Internet

Van Jacobson

Lawrence Berkeley Laboratory
Berkeley, CA 94720

SIGCOMM ’95 Middleware Workshop
Cambridge, MA
28 August 1995

The Internet was designed to survive a nuclear war.

It lives up to its design.

How might you kill it?

Easy — Invent the Web.

Web traffic is destroying the Internet.

Not because it's popular — there are benign popular protocols — but because the application-level protocols are abysmal.

With a lot of effort and cooperation, we may be able to fix the Web without changing its user interface.

But it would be nice if application designers didn't make this painful set of mistakes again.

With 25 years of Internet experience, we've learned exactly one way to deal with exponential growth:

Caching.

Data has to find 'local' sources near consumers rather than always coming from the place it was originally produced.

But the number of caches has to grow as fast as the Internet, exponentially.

How do we architect for *lots* of caches?

Configuration is a major problem.

Experience shows that anything that must be configured will be misconfigured.

Web clients need hostname of their local cache.

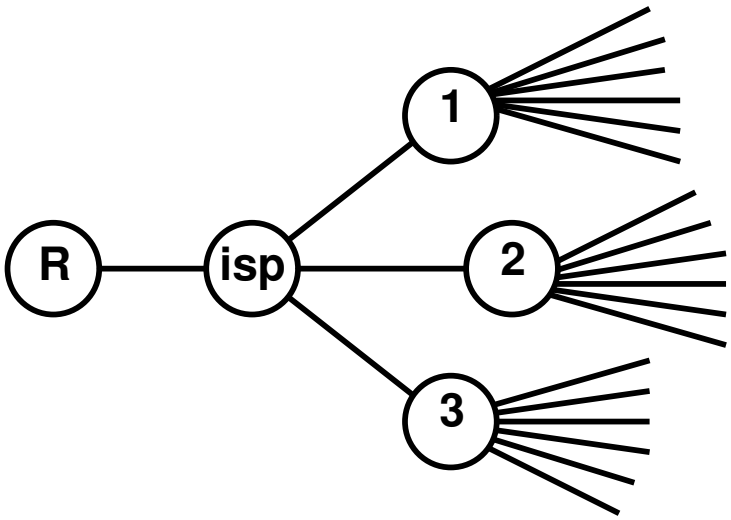
$\Longrightarrow$ Client rendezvous with local cache should be automatic.
I.e., use multicast.

For any source of popular data, want a cache distribution tree rooted at that source with leaves near everyone who wants the data.

But nothing in the Internet is static: sources, receivers and distribution topology all change continuously.

$\Longrightarrow$ Cache hierarchy should be
self-configuring and adaptive.
I.e., use multicast.

Near root of distribution tree, it's very likely data requested by one cache will soon be needed by another.



$\Longrightarrow$ Should use efficient, multi-point distribution for data.
I.e., use multicast.

Using multicast implies that we stop thinking of communication as ‘conversations’:

> Instead of asking *X* to send you *Y,* simply ask for *Y.*

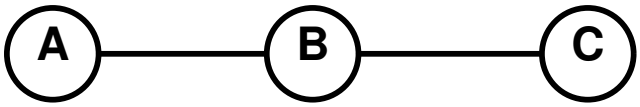
(This is a profound change at the protocol level — Dave Clark’s ALF).

Applications and data (data naming) have to be designed with *scalable* caching in mind.

This implies either explicit lifetimes or generation numbers otherwise bottlenecks re-appear due to cache-coherency protocol.

(The average transaction is small so
a validity check $\approx$ a data xfer.)

The Internet has grown because IP's default behavior is transitive and a site has to do extra work to limit transitivity.



Need the same attitude in application design.

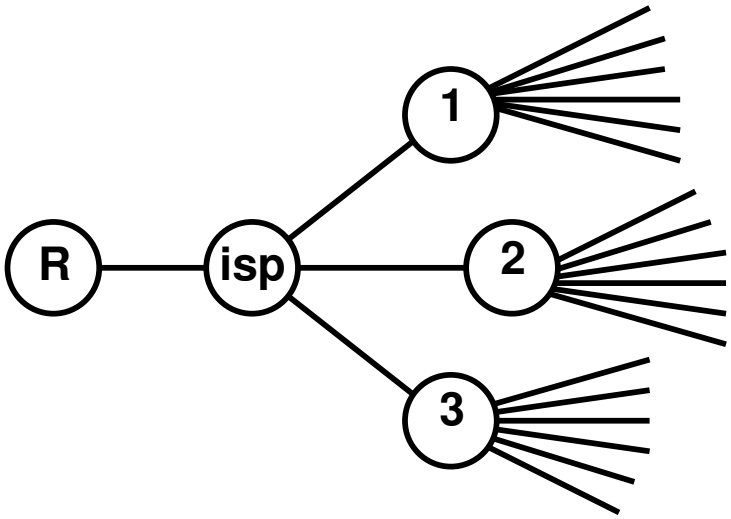
I.e., every reader is a cache (becomes a source of the data read) and users have to work to turn this off.

Large-scale caching makes it much more likely that data is corrupted or subverted.

$\Longrightarrow$ Need cheap, universal, data integrity and authentication machinery.

Item size vs. packet size mismatch might require machinery with some new properties.

Need a billing architecture that encourages rather than discourages caching.



$\Longrightarrow$ Receiver-pays, *not* sender-pays.