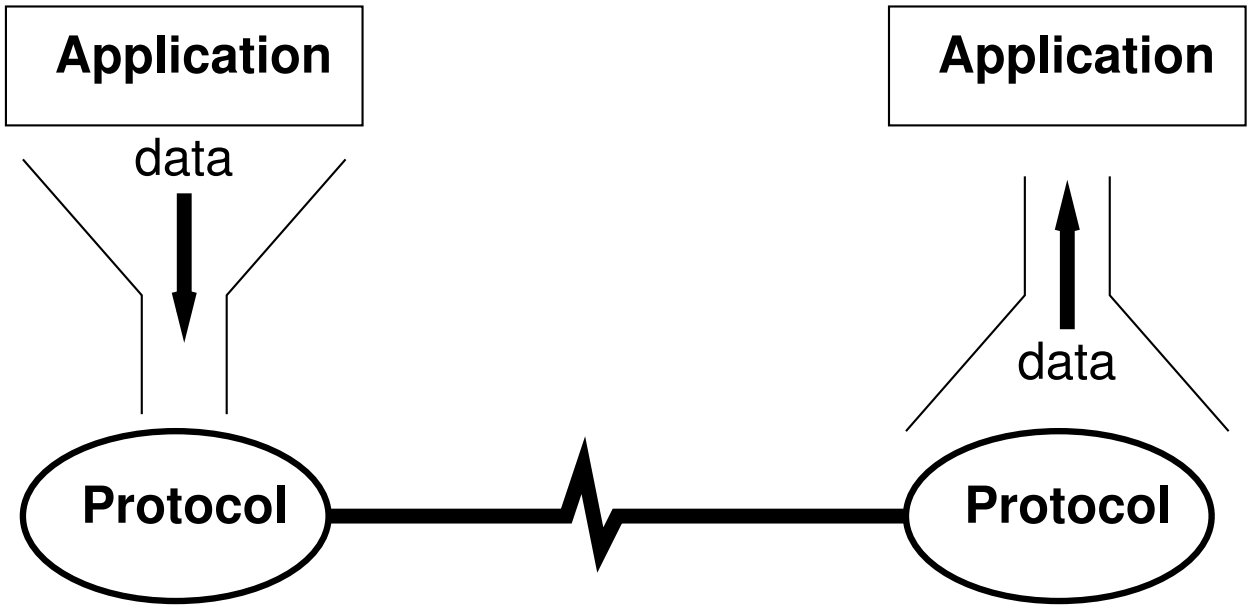
Lightweight Sessions — A new architecture for realtime applications and protocols

Van Jacobson
Steve McCanne
Sally Floyd

Lawrence Berkeley Laboratory
Berkeley, CA 94720

Networkshop '93
Melbourne, Australia
30 November, 1993

The world as we know it



Real-time traffic changes the rules

Application requirements may make ‘protocol’ a no-op.

E.g., audio sender wants no flow control & unreliable delivery.

Audio receiver does resequencing as side-effect of playout adaptation.

Changing the rules (cont.)

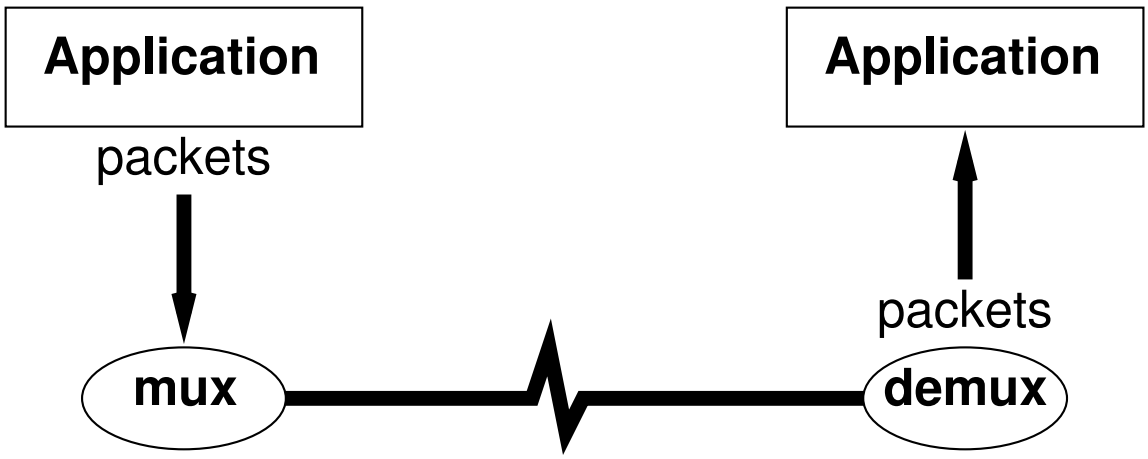
Application policy may be too involved to allow a useful application/protocol split.

E.g., ‘whiteboard’ send policy is (priority order):

- ‘repair’ responses for current page
- new data for current page
- ‘repair’ responses for other pages

Telling a ‘protocol’ what data to send is much more complicated than simply building a packet in the app.

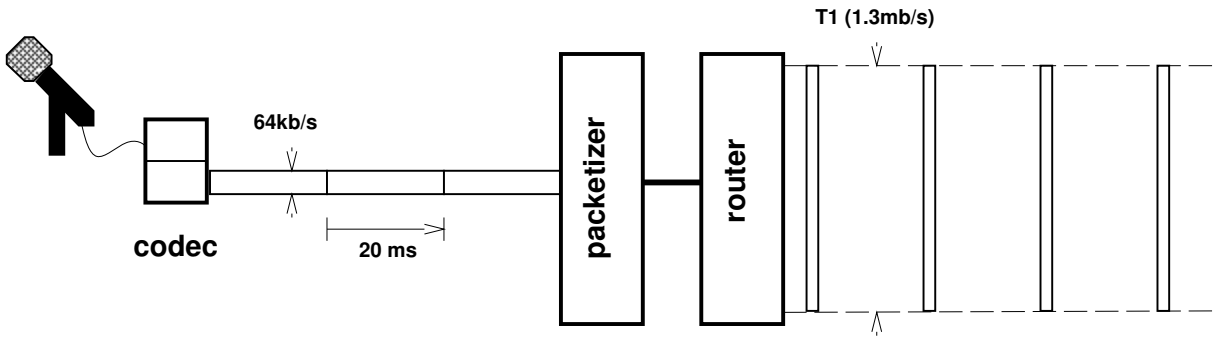
Dave Clark suggested a better networking model a couple of years ago: Application Layer Framing (ALF):



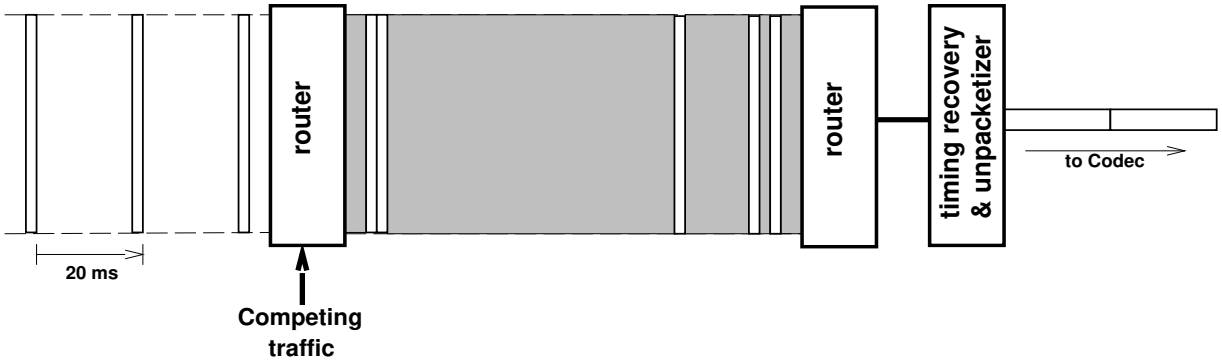
But he left out a few minor details:

- How do you adapt to changing network conditions?
- How do you do multipoint delivery?
- How do you do reliable multipoint delivery?

Basic problem: network distorts time sequence



Basic problem (cont.)



Adapting to network conditions

Several possibilities:

1. Can prohibit distortion

2. Can make network adapt to traffic

3. Can make receiver adapt to network

(1) only works for phone calls (low peak/avg ratio);

(2) requires we scrap Internet then solve NP-hard distributed scheduling problem;

(3) requires a few kilobytes of buffer in receivers.

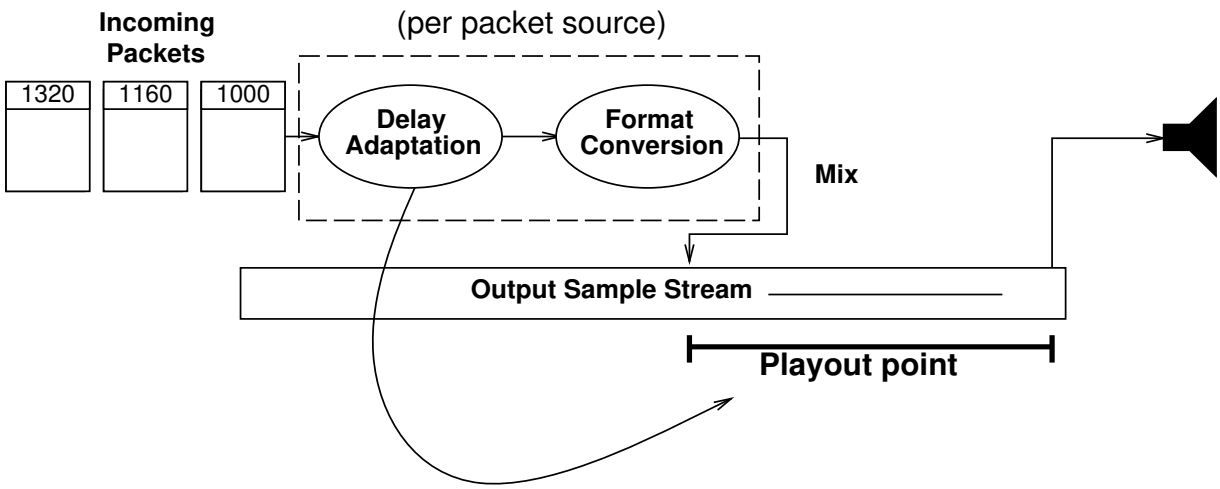
Memory costs $30 a megabyte. We do (3).

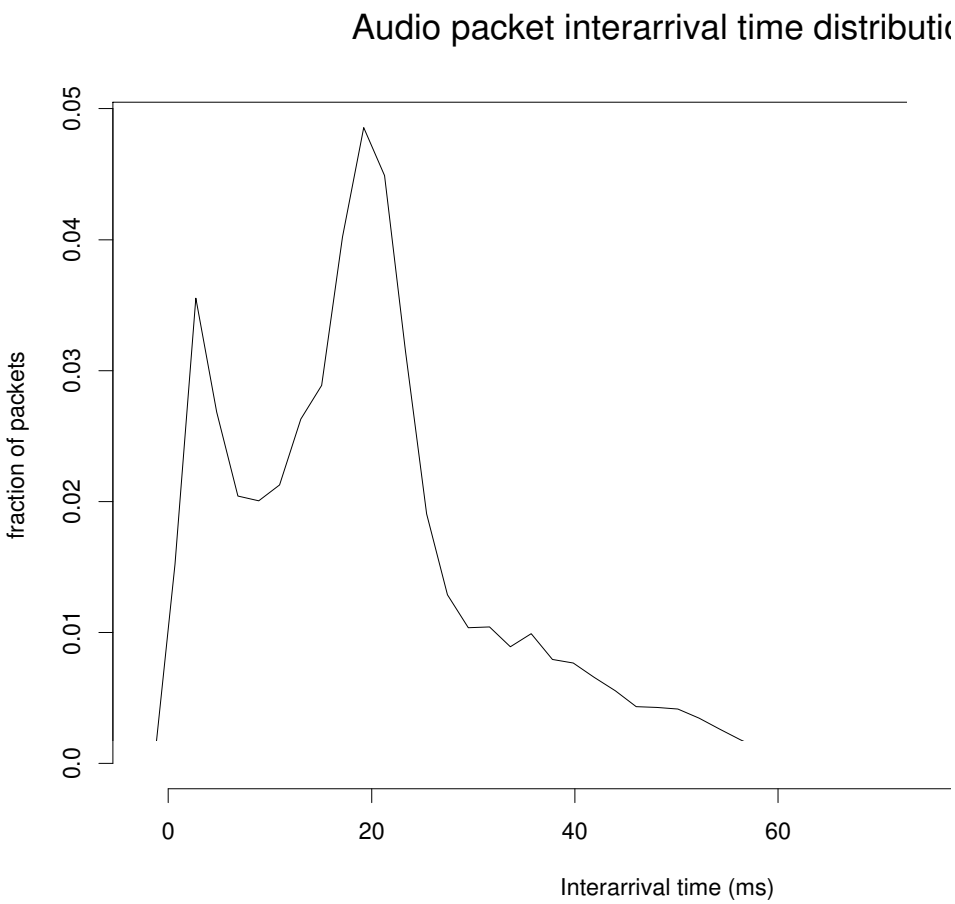
Adapting to network conditions (cont.)

Senders put explictit ‘time’ information in packet stream.

Receivers use this to:

- isolate effect of network on packet stream, then
- estimate ‘playout point’ delay that will remove effect of network.


Incoming
Packets
(per packet source)
1320
1160
1000
Delay
Adaptation
Format
Conversion
Mix
Output Sample Stream
Playout point


Audio packet interarrival time distributi
fraction of packets
0.0
0.01
0.02
0.03
0.04
0.05
0
20
40
60
Interarrival time (ms)

Multipoint delivery

Steve Deering provided most of the answer (IP multicast).

Lixia Zhang et.al. provided major missing piece (receiver initiated reservations & RSVP).

We added two small pieces:

- Conventional net management tools don't debug multicast distribution. Multicast sessions must include diagnostic information.

- For things to scale beyond a few participants, all communication must be multicast.

IP multicast communication model:

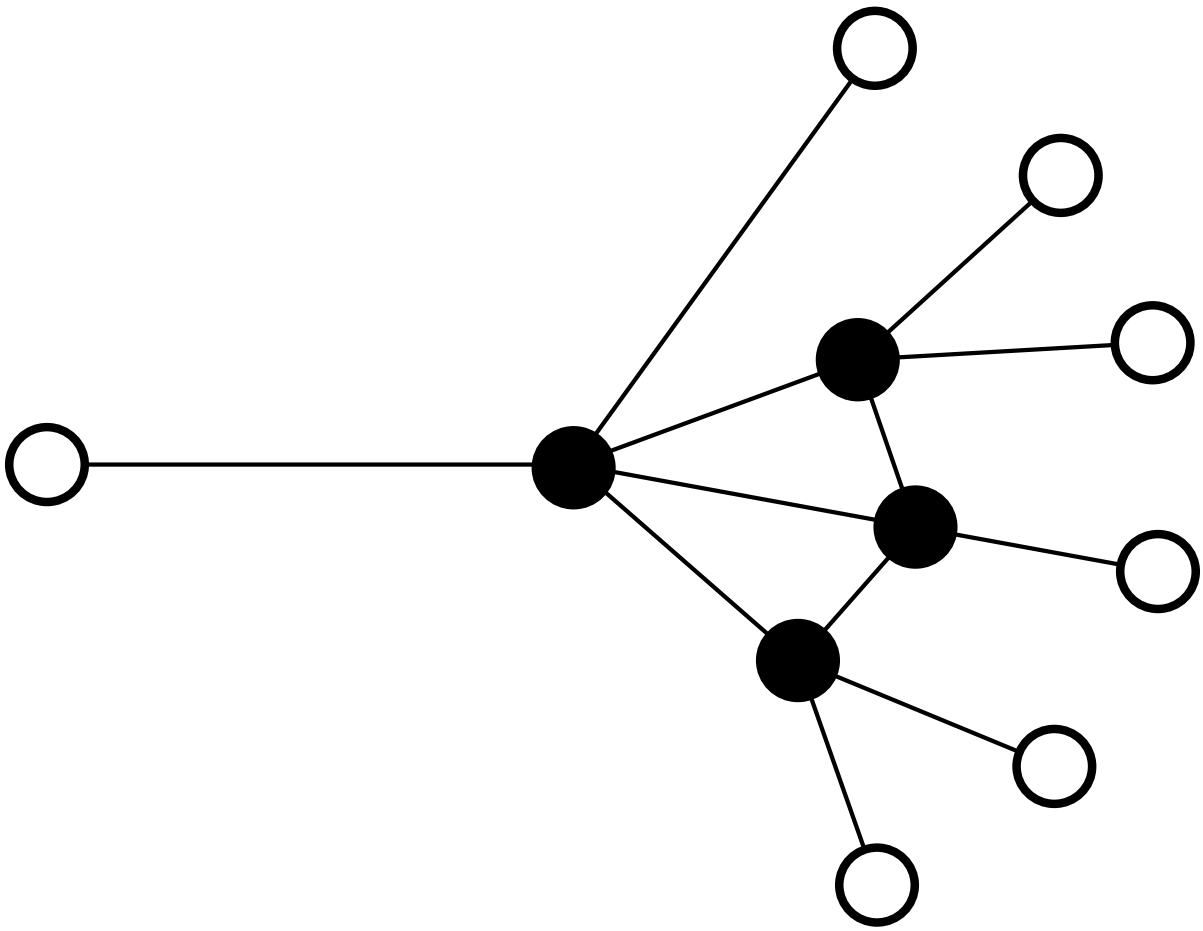
- Datagrams with best effort delivery.
- Very lightweight setup: senders just send, receivers announce interest, network delivers data to interested receivers.
- No need to know participants in advance (or ever).
- Inherits all the robustness of IP.

IP multicast (cont.)

- Works in the face of intermittent connectivity.
- Works if large bandwidth disparity between sites — doesn't slow down to least common denominator.
- Scales to arbitrary number of sites with no O(n) degradation in bandwidth or setup time.

Problem: Resource reservation for multicast data (sender doesn't know topology or participants).

Solution (from Lixia Zhang): receiver initiates reservation (RSVP).



Basic Lightweight Sessions model

'Session' is multicast address + port + random number.

Sites with data send it.

All sites quasi-periodically send 'session' packets containing:

- identity
- reception reports
- synchronization info

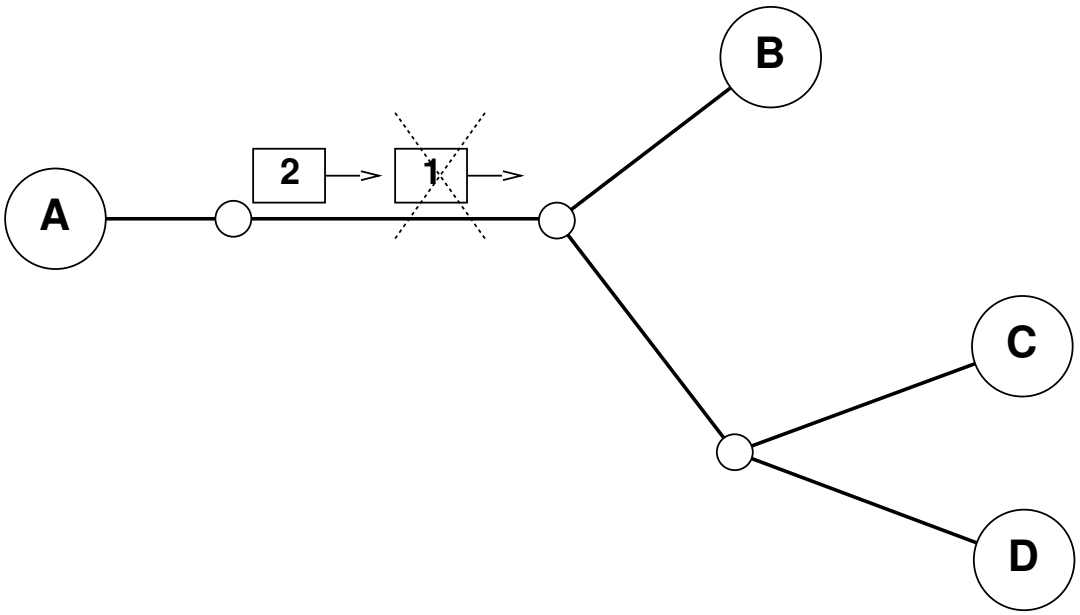
LWS Design Space

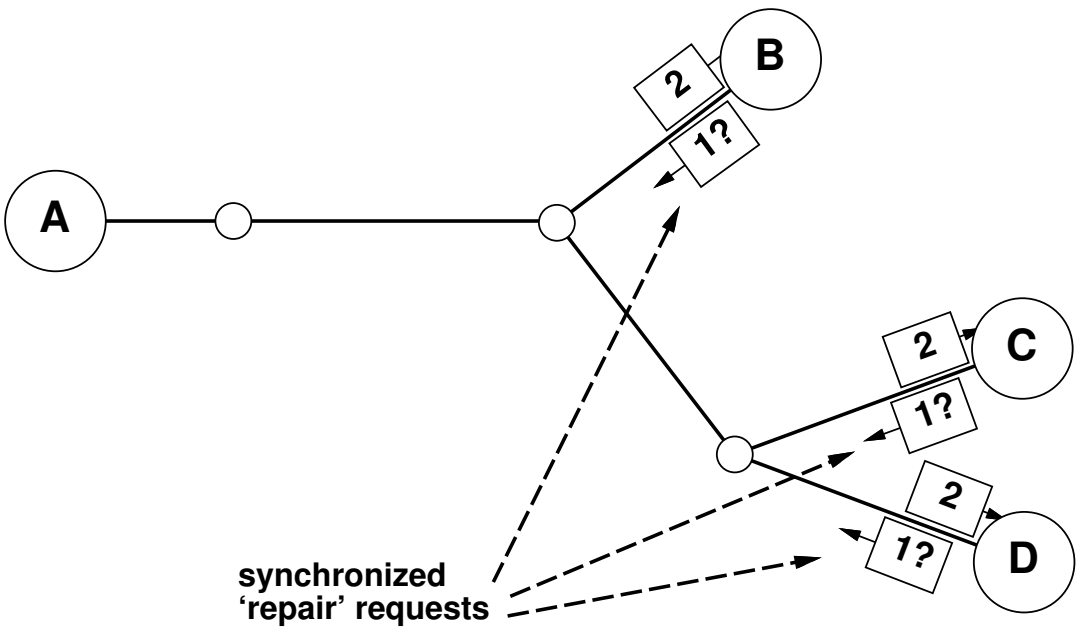
Rate adapts?	Tolerates Loss?	
	some ok	no
no	audio	
some	video	wb

Note there is no traffic characterization — no peak or average rates, no delay or jitter bounds, no burst sizes, etc.

Characteristic of this model is that things work. You selectively do resource reservation/setup if you need them to work better.

‘Reliable’ delivery is complicated by ‘ack implosion’ problem:





Avoiding ack implosions (receiver)

1. If new hole detected, allocate new ‘repair’ structure & set its request interval i to p (prop time from sender to you).

2. delay random time between i and $2i$ before requesting repair.

3. If hole filled, cancel repair.

4. If someone else requests repair, double i & go to step (2).

5. If timer fires, multicast repair request, then double i and go to step (2).

Avoiding ack implosions (sender)

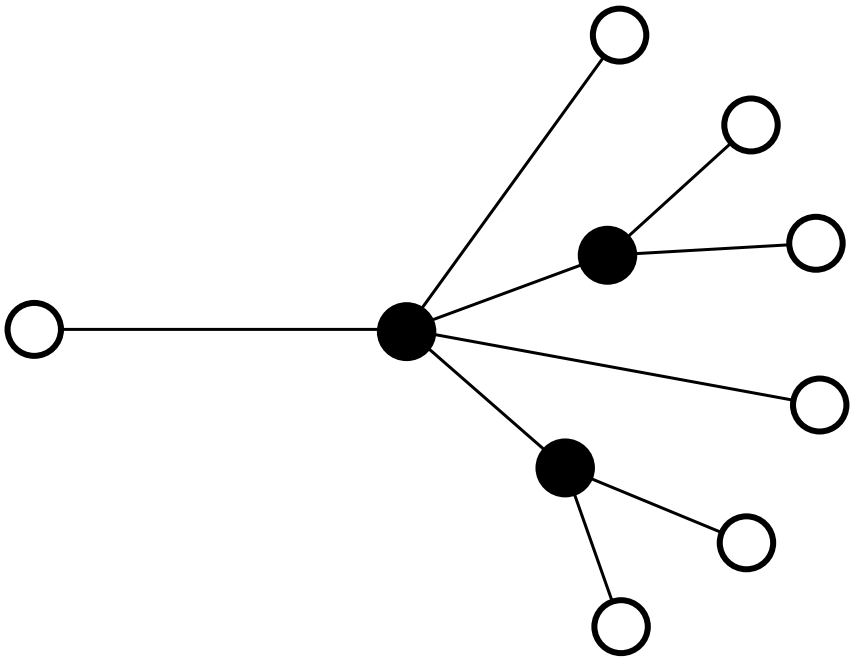
If repair request received for data that you have

1. delay random time between 0 and p (where p is prop time from requestor to you)

2. If someone else fills hole, cancel repair.

3. If timer fires, multicast repair data.

Also need to deal with two different failure modes: near sender (root of distribution tree) and on particular branches:



Mostly this involves adding scope control to previous algorithm.

Lessons we've learned

- use application specific protcols, not layered models or one-size-fits-all 'real-time' protocols.

- use multicast for everything
  - allows distribution problem diagnosis and allows partcipants to rate adapt
  - avoids ack/response implosions
  - improves repair response time & reduces bandwidth demand

Lessons we've learned (cont.)

- inject randomness to prevent correlated traffic
- distinguish data sources from packet sources
- design around simplex communication and distributed concensus. E.g., receivers can chose to ignore a sender but can't stop him.

Things we have to do:

- Write and field diagnostic tools
- Write and field some initial ‘coordination’ tools (floor control and synchronization).
- Make reference implementations (sources) available.
- (list is infinite – remainder omitted).