Multimedia Conferencing on the Internet

Van Jacobson

Lawrence Berkeley Laboratory
Berkeley, CA 94720

SIGCOMM ’94 Tutorial
University College London
London, England
30 August 1994

A widely believed myth:

Real-time traffic like audio and video requires a connection-oriented, virtual circuit network. Datagram networks can't be used for real-time traffic because

- they don't have the state necessary to meet real-time scheduling and delivery constraints and

- IP delivery is 'best effort' so transit times are unbounded and vary wildly.

But . . .

There has been world-wide, IP-based, real-time conferencing over the Internet (via the MBone) in daily operation for the past two years.

The 10,000+ users on 1500 networks in 30 countries spanning 22 timezones will be terribly disappointed to learn that all these meetings, seminars, social events, etc., are a mass delusion.

In fact, there have been many attempts at connection-oriented approaches to conferencing and collaborative work. Most have been dismal failures. None have worked well. There are reasons for this:

- Too much burden on users — they have to know other participants and details of network topology.
- Horrible scaling — if everyone can talk and listen there are $O(n^2)$ connections for n participants.
- Unreliable — conference fails if any of the n^2 connections fail.
- Intolerant — difficult to join in-progress conference.
- Slow — goes no faster than the slowest participant.

Connection-oriented approaches make a key mistake: ‘the conference’ has no network-visible identity — a conference is just the collection of its participants.

Since core of problem is invisible to network, useful division of labor between application and network is impossible.

The network can’t help you if you don’t tell it the problem!

New Collaborative Application Model

LBL (Sally Floyd, Steve McCanne & I) with Xerox PARC, MIT and ISI have developed a new communication model for collaborative applications called **Light-weight Sessions** (a generalization of IP multicast). This model places no burden on users, has all the scalability, fault tolerance, and robustness of IP, accommodates intermittent connectivity, and delivers good performance even when participant link bandwidths are very different.

Light-weight Session building blocks are:

- IP (datagrams with best-effort delivery)
- IP Multicast
- Timing recovery via receiver adaptation
- ‘Thin’ transport layer (Dave Clark’s Application Layer Framing, ALF)

IP and
IP Multicast

Why IP?

IP is more robust than Virtual Circuits (X.25 style calls) because it made right separation of global and local:

- VC packet identifiers have local meaning so problems have to be fixed globally.

- IP packet identifiers have global meaning so problems can be fixed locally.

IP scales and works well because it has a very clear separation of roles:

- End-nodes know nothing about topology.

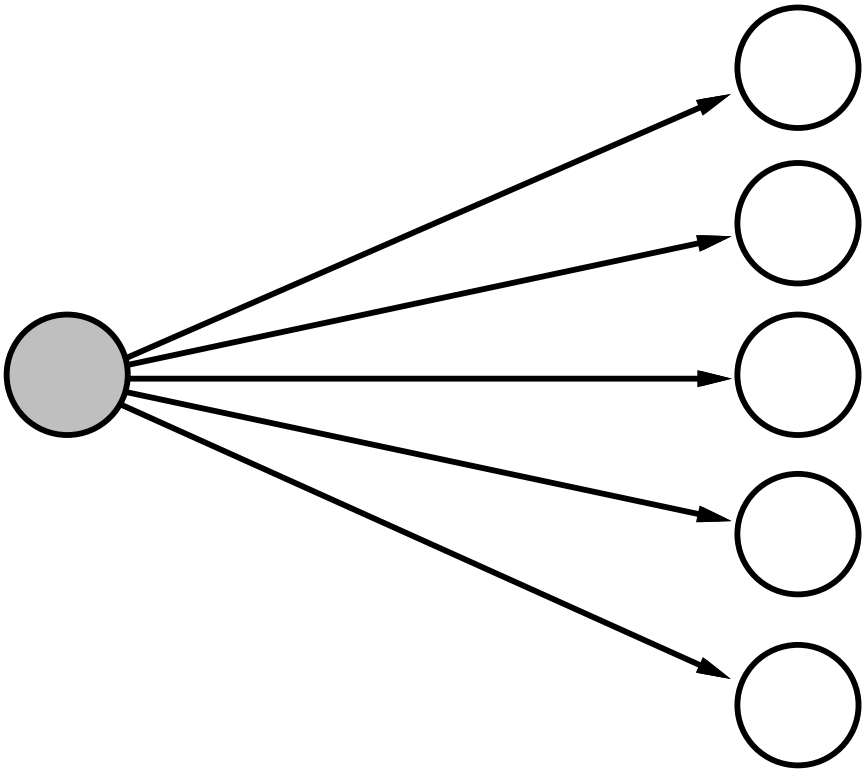
- Routers know nothing about ‘conversations’.

First item allows net to dynamically change delivery topology.

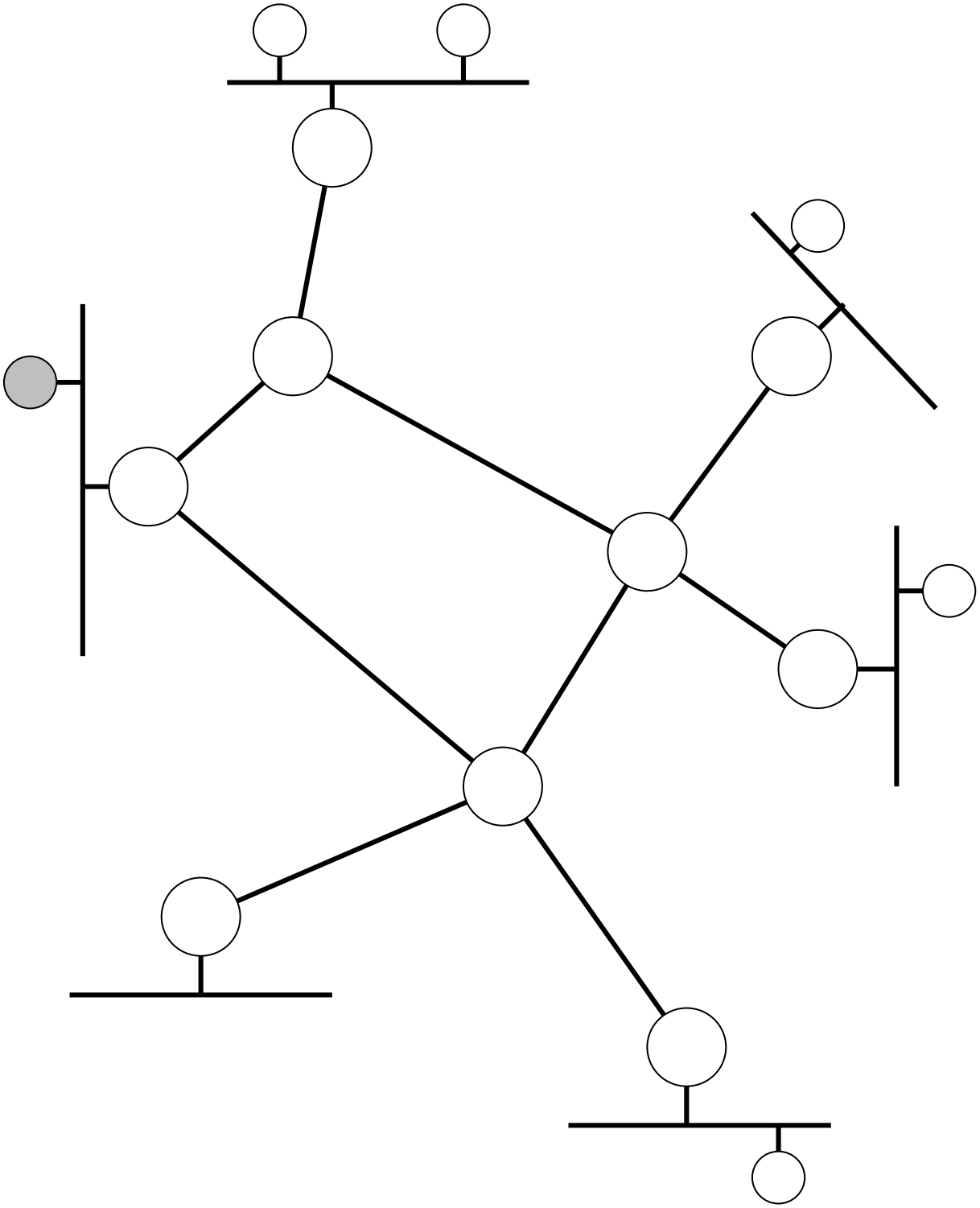
Second means that changes can be done without global coordination (no end-to-end state to move around).

But separation of roles is problem for multi-point conversations.

I.e., sender thinks the world looks like this:



When in fact the topology is:



This results in multiple copies of data being sent across one link and demand on net will increase proportional to number of receivers. I.e., we won't be able to have large conferences.

Core problem: How to do efficient multipoint distribution (i.e., at most one copy of a packet crossing any particular link) without exposing topology to end-nodes.

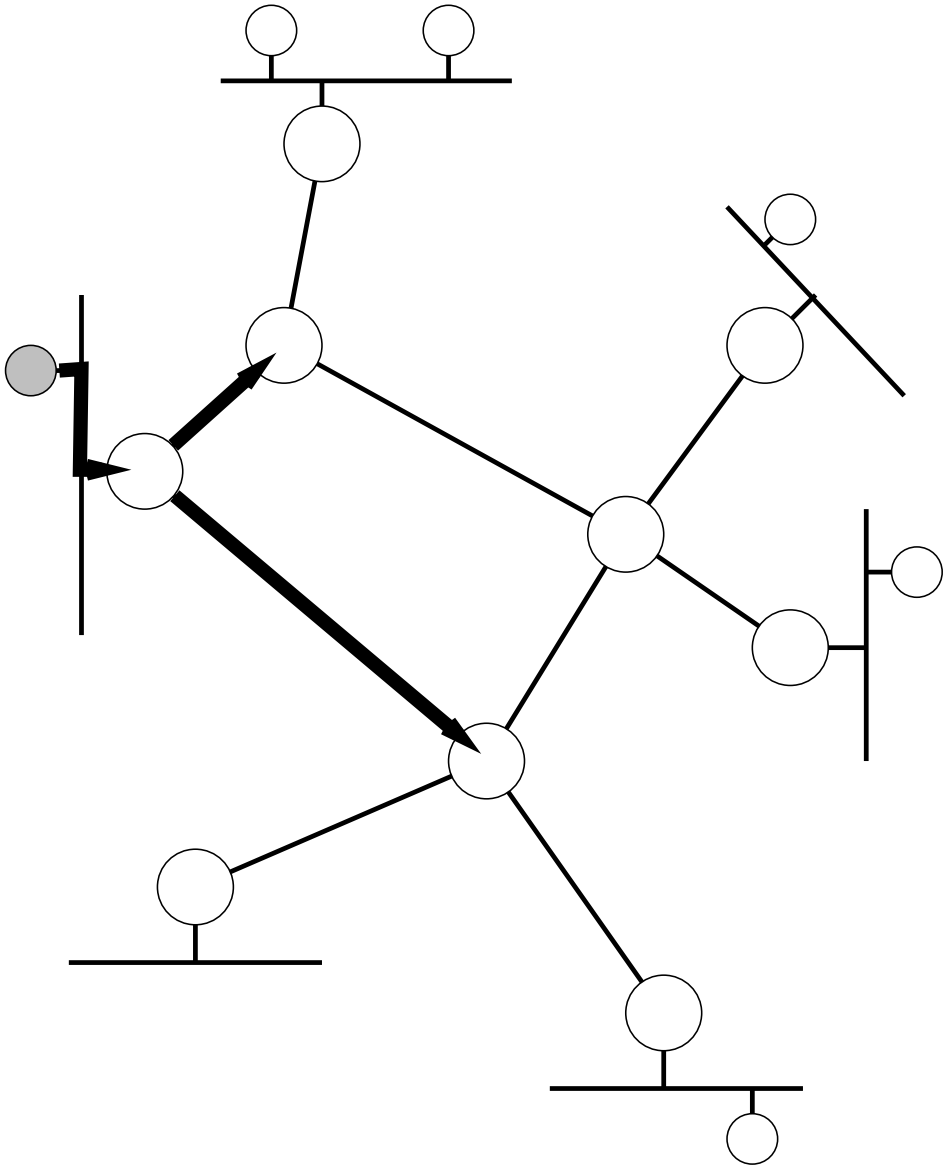
Solution: Steve Deering's IP Multicast

- Dynamically constructs efficient, shortest-path, distribution trees from sender(s) to receiver(s).

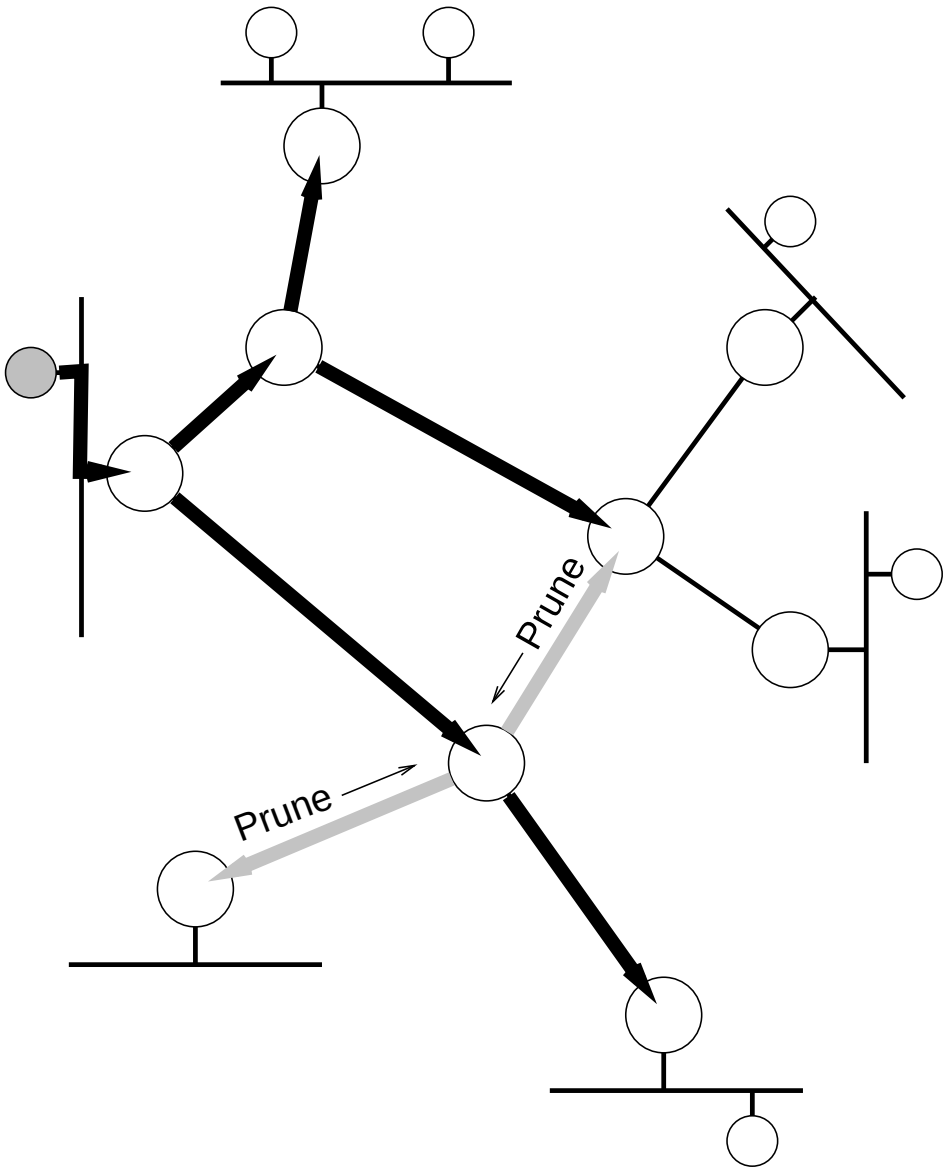
- Very simple service model:
  - Receivers announce interest
  - Senders just send
  - Routers conspire to deliver sender's data to all interested receivers.

- Preserves separation of roles and thus all the robustness of IP.

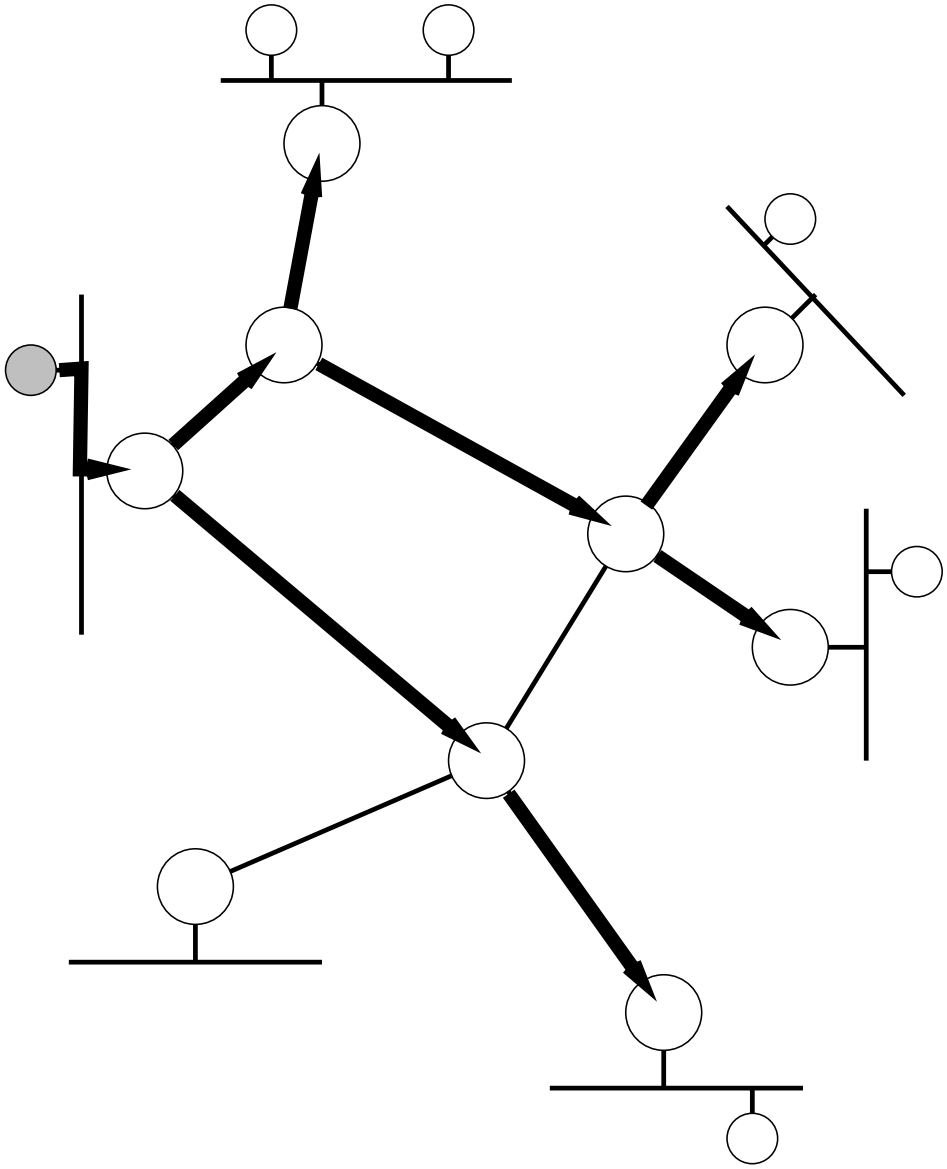
Basic model is flood-and-prune (can flood either data or ‘interests’):



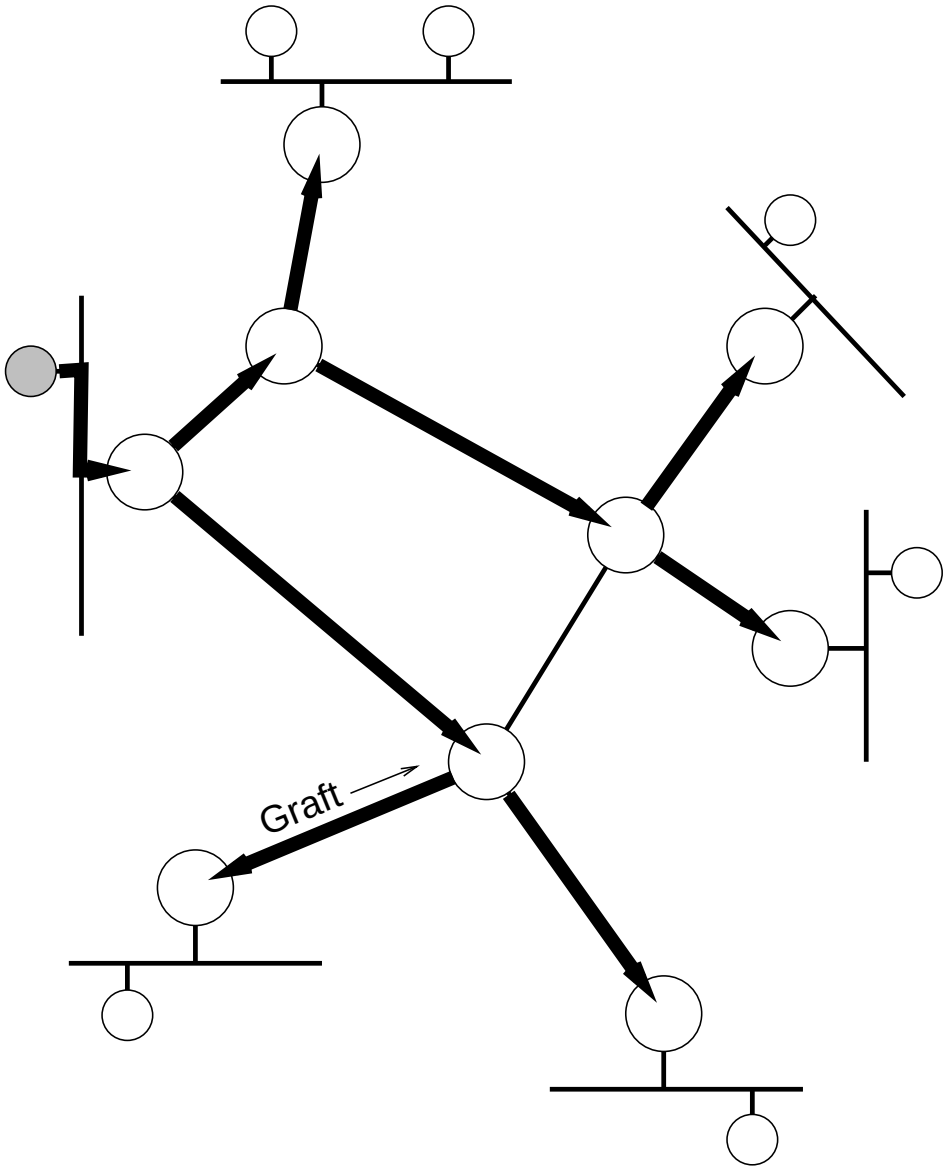
Branches where there are no members and branches not on shortest path tree get pruned off:



Resulting in a shortest-path distribution tree rooted at sender:



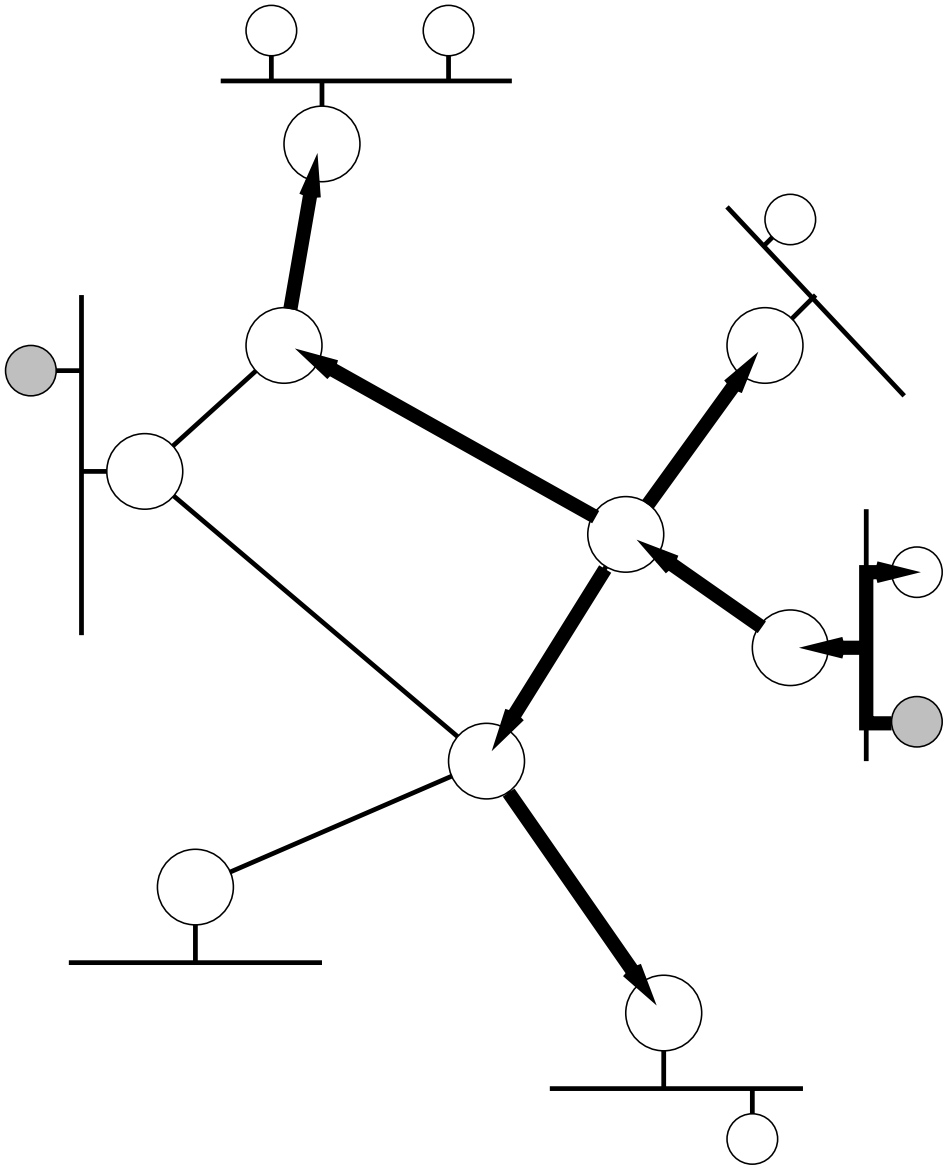
Changes in membership are handled efficiently and locally via graft/prune machinery:



Locality implies scalability:

Work of maintaining distribution tree is spread out and changes don’t propagate or concentrate.

Distribution trees are per-(sender, group) and constructed via lazy evaluation, triggered by data packets.



Note: Preceding was very rough overview. There are many ways of doing IP Multicast. They have different time, state and traffic scaling properties. They all interoperate and all preserve the robustness of IP. They all present the same, simple, service model to end-nodes. For more information, see Steve Deering's thesis and the PIM and CBT papers in this conference.

Note that IP Multicast is more than a simple, efficient, robust, delivery mechanism. It also greatly simplifies the conferencing problem for users. I.e., if we associate a ‘conference’ with a multicast address, then

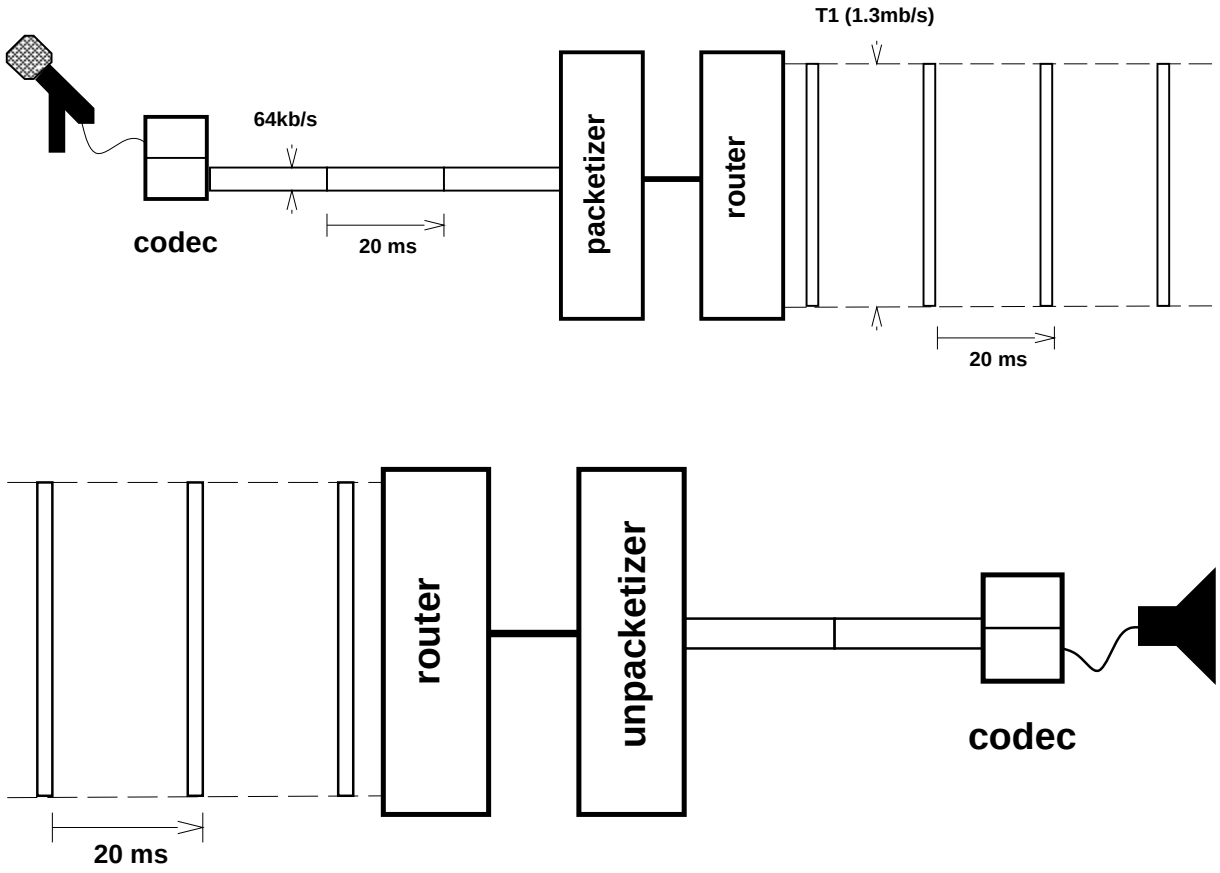
- Users can join the conference without enumerating (or even knowing) other participants.

- Users can join and leave at any time.

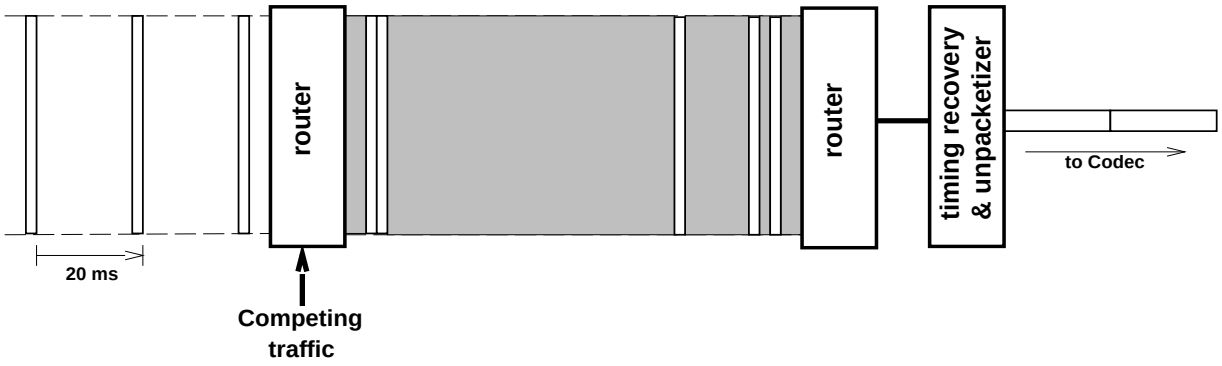
- The conference has a network visible identity so the net takes care of rendezvous, distribution and membership, not users. This means we inherit the group-size-independent scaling of multicast.

Meeting Real-time Delivery Constraints

Unlike normal data traffic, the time structure of real-time traffic has significance:



But, since the network infrastructure is shared, competing traffic can distort this time sequence:



There are three ways to deal with this:

- Don’t share. E.g., the phone system assigns a different TDM time slot to each phone conversation. But if sources are not on continuously, this wastes a lot of bandwidth.

- Schedule different traffic streams so that they don’t interfere. But this requires that every host and router adhere to schedule for anything to work. Also, distributed scheduling problems are known to be NP-hard.

- Make sender put timestamps in packets so timing is explicit rather than implicit then receiver can reconstruct timing before playing out data. This requires a few kilobytes of buffer at receiver (but memory is $30/megabyte).

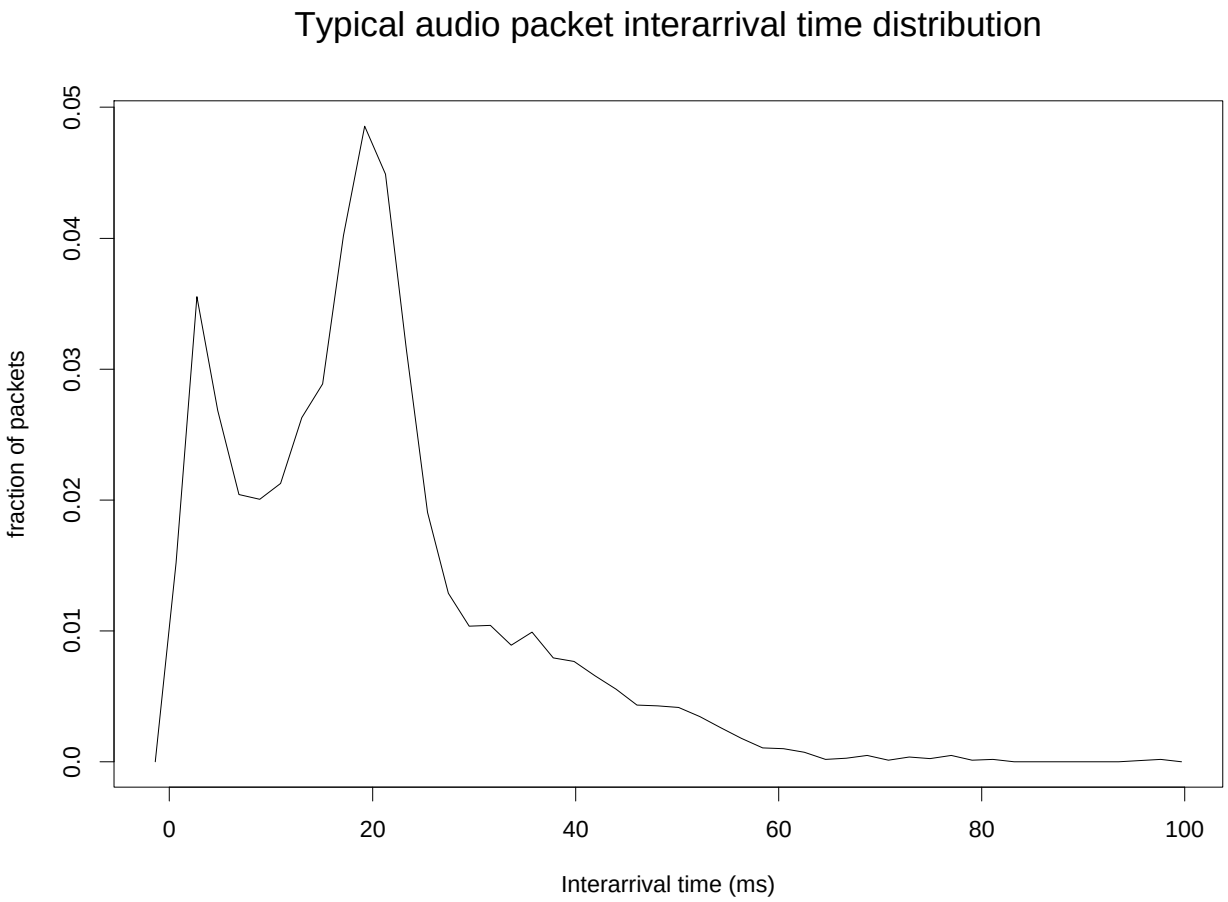
How does receiver reconstruct timing?

Note that the average arrival rate at the receiver must equal the departure rate at the sender (otherwise the network doesn't have enough capacity for the conversation and nothing will make things work).

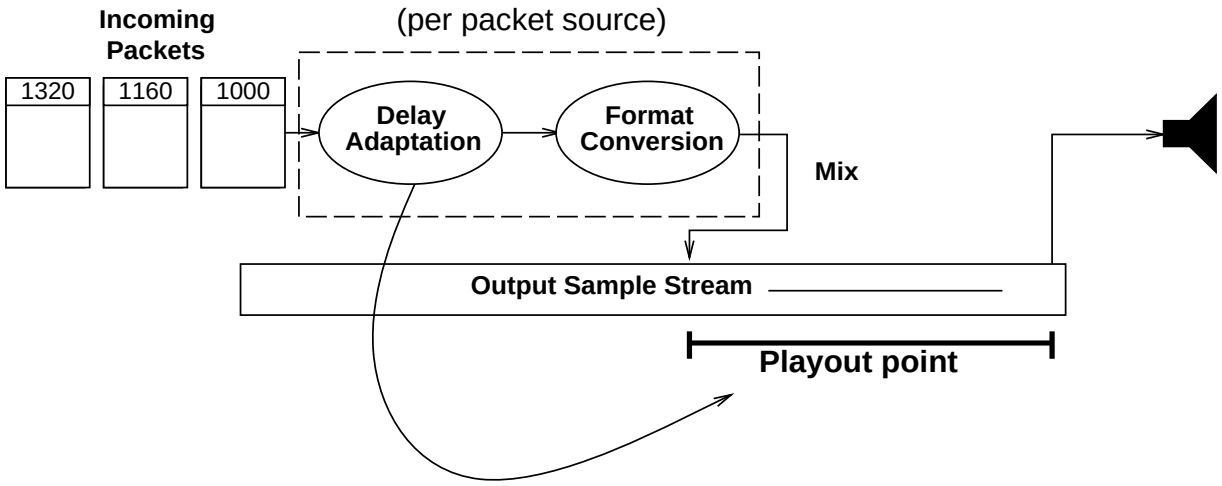
Since 1st order (mean) statistics are same at sender & receiver, the receiver only need adapt to 2nd order (variance) of arrival rate. This is easily estimated as the mean of the interarrival time.

How much buffer might be needed?

Some claim that IP makes no delivery guarantee so arrival rate variation is unbounded & required buffer is infinite. But variation is due to fluctuations in queues in intermediate routers. Queues (thus queue fluctuations) are bounded and small. Fluctuations along path also add in quadrature (partially cancel each other) so real variation is small:



So, adaptive application starts to look like:



Basic ‘playout point’ calculation is low pass filtered version of the difference between packet interarrival and interdeparture times (this is just the amount of timing distortion introduced by the network).

I.e., if T_i is the timestamp the sender put in packet i and A_i is the time that packet i arrived at the receiver then

$$\begin{aligned} V_i &= |(A_i - A_{i-1}) - (T_i - T_{i-1})| \\ &= |(A_i - T_i) - (A_{i-1} - T_{i-1})| \end{aligned}$$

and the estimated mean variation at the ith packet is

$$\begin{aligned} M_i &= (1-g)M_{i-1} + gV_i \\ &= M_{i-1} + g(V_i - M_{i-1}) \end{aligned}$$

where $g < 1$ is a smoothing constant.

Expressing the preceding in C, say there's per-source state at the receiver that holds the current playout estimate and the previous time difference. If the smoothing constant is 1/8, the playout update looks like:

```
state = findstate(packet.src);
d = now - packet.ts;
v = abs(d - state->d);
state->d = d;
state->m += v - (state->m >> 3);
```

If the timestamps are in samples, `state->m` will be the playout point (the amount to delay this packet) in samples, left shifted by 3.

For particular applications there may be additional considerations in the playout calculation.

With audio, for example, adapting the playout on every packet introduces audible gaps.

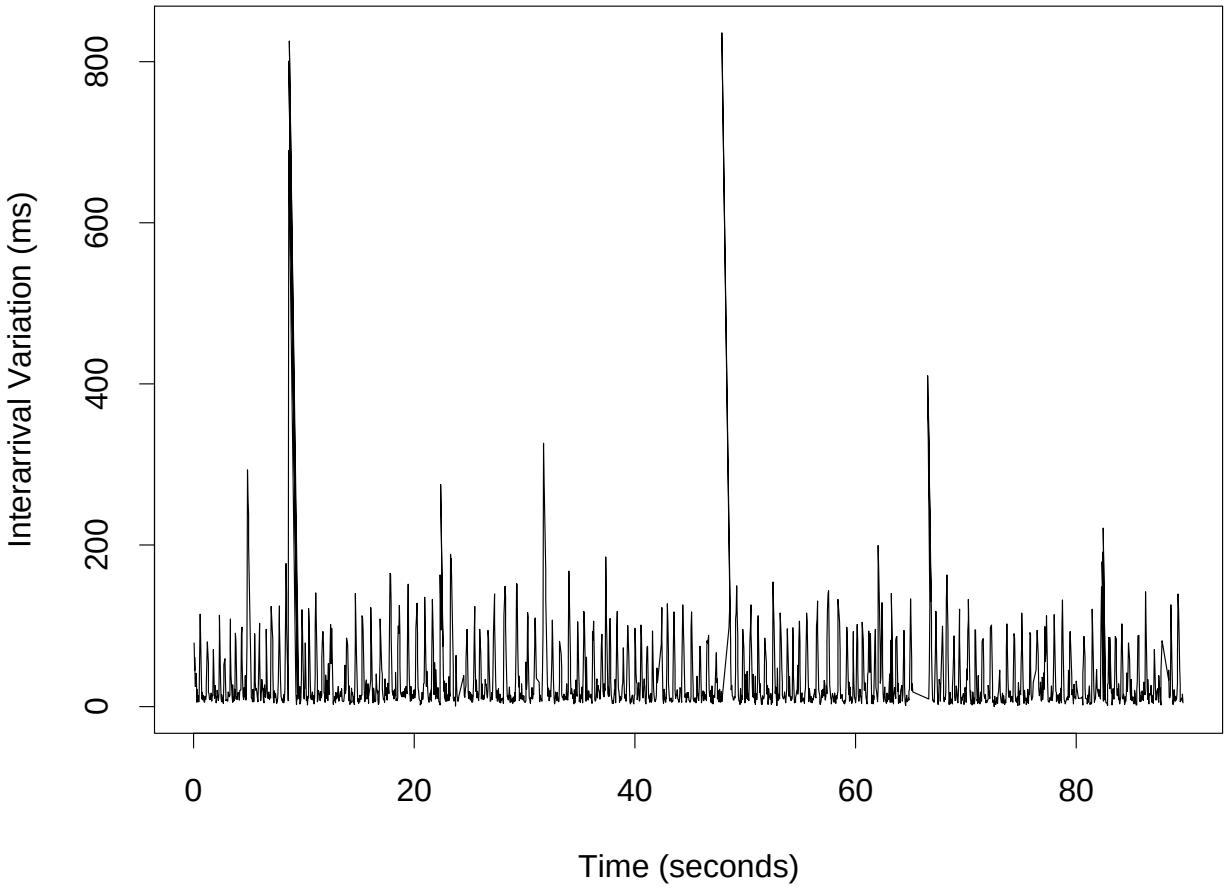
It’s better to estimate the playout on every packet but only adapt at the start of a talkspurt (so the adaptation makes an inaudible change in the length of a silence interval).

Also, different types of conference may have different playout objectives:

A receiver in an interactive meeting might be willing to tolerate slightly greater packet loss in exchange for lower playout delays and better interactivity.

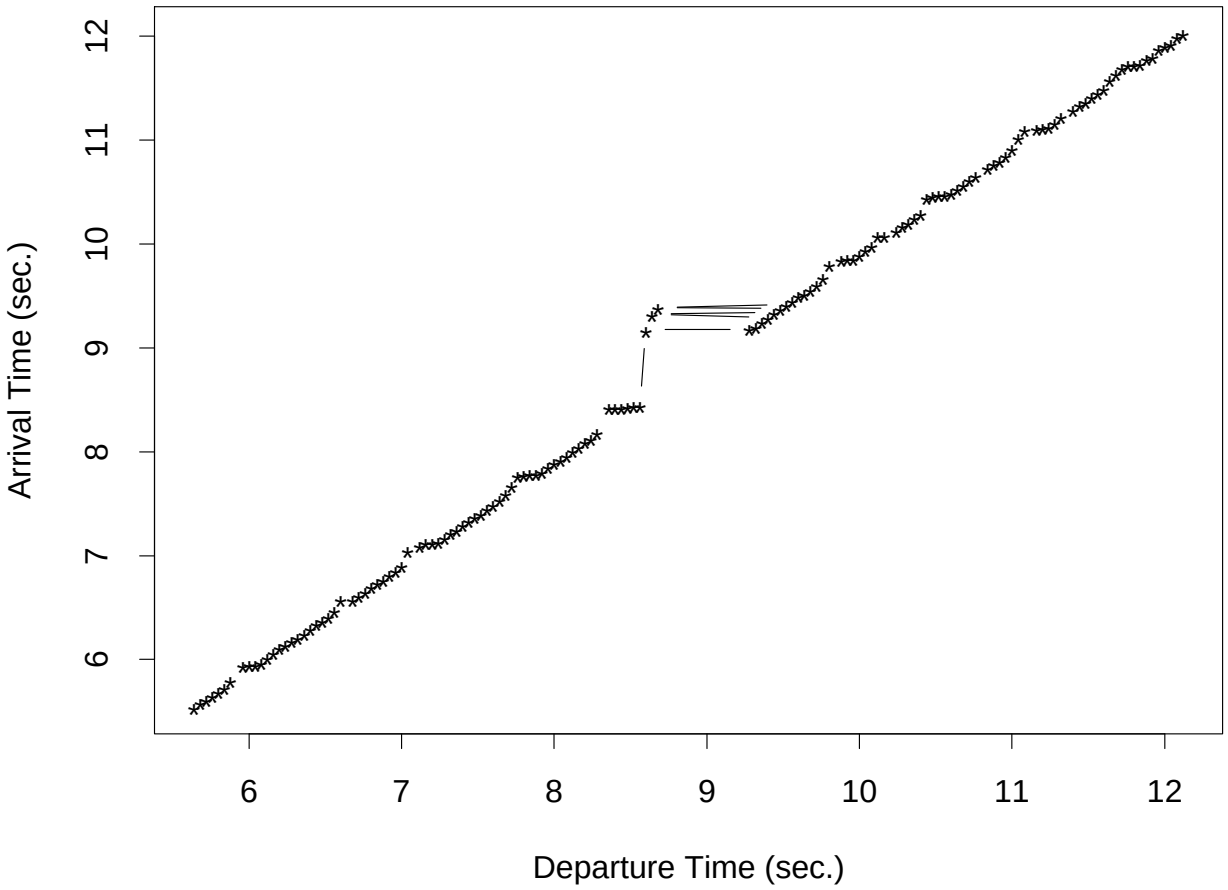
The receiver of a seminar might not care about delay and want to minimize packet loss to get the best possible reception.

Knowing the (human) receiver's objective is important because interarrival variation is not smooth so there is a large difference between the median and max. of the distribution:



(this data is for audio packets from the International Symposium on Electronic Art, originating in Helsinki, Finland, and being received in Berkeley, California.)

The structure of the variation is easier to understand if you look at input vs. output times rather than at time differences:



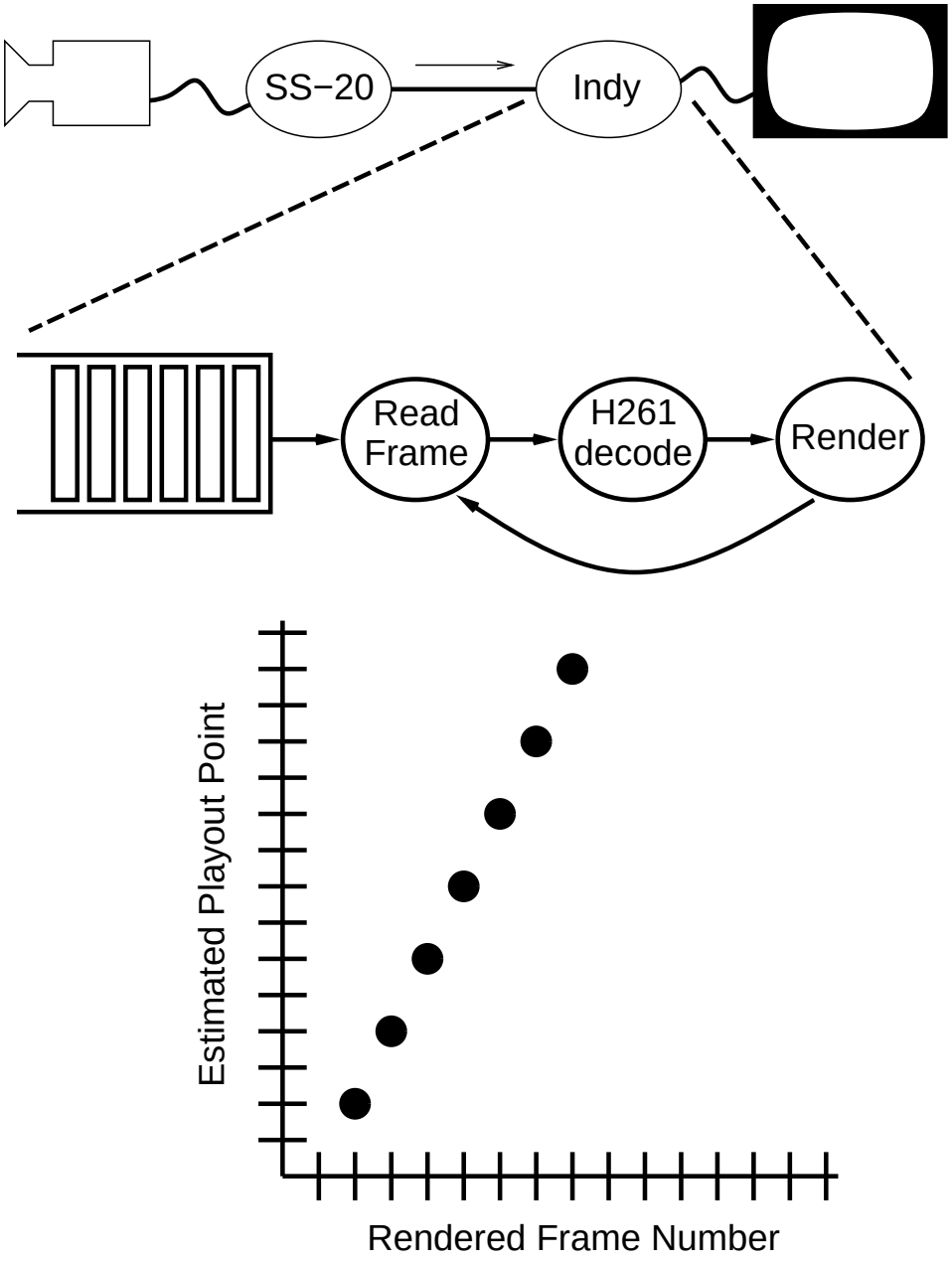
For complex audio and video compressions there may significant variation in frame-to-frame decompression times that should be included in the playout delay.

As long as A_i is consistently referenced to either start or end of packet processing cycle, calculation above will automatically include this variation in playout point calculation.

Related problem is that sender and receiver processing rates may be mismatched.

E.g., workstation vendors seem to regard anything within $\pm 10\%$ of 8KHz audio sampling rate as ‘good enough’ so actual audio rates vary wildly. If sender sources at 9K samples/sec. and receiver sinks at 7K samples per sec., playout point at receiver will increase linearly until it hits max and packets are dropped,

Similarly video send and receive rates might be mismatched. E.g., a Sun SS-20 can software encode and source H.261 video at 30 f/s but an SGI Indy can only software decode and sink H.261 video at 10 f/s. Since 3 packets arrive for every one processed, input queue rapidly fills up, delay is maximized, and very bursty drop behavior follows.


SS−20
Indy
Read
Frame
H261
decode
Render
Estimated Playout Point
Rendered Frame Number

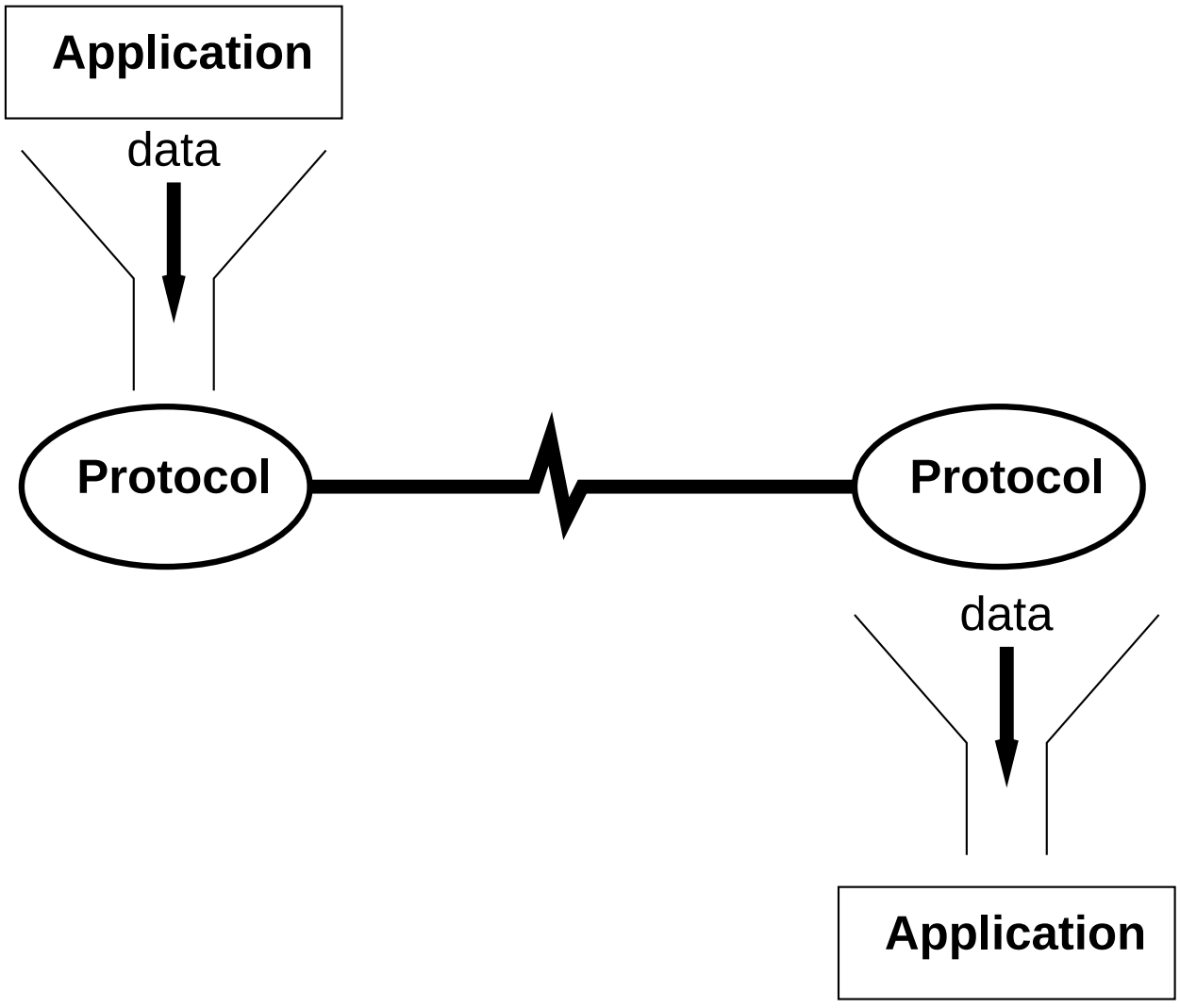
Solution is to estimate both playout point and trend in estimate (via either a Holt-Winters estimator or a 2-d Kalman filter.)

In example, trend will be two frames/frame since three frames arrive for every one that's rendered.

So, if reader discards two frames for every one it renders, there will be no steady-state backlog or delay and rendering will be as smooth as the broken hardware allows.

Application / Network Interface: ‘Thin’ Transport Layer and ALF

The world as we (used to) know it:



Real-time conferencing 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.)

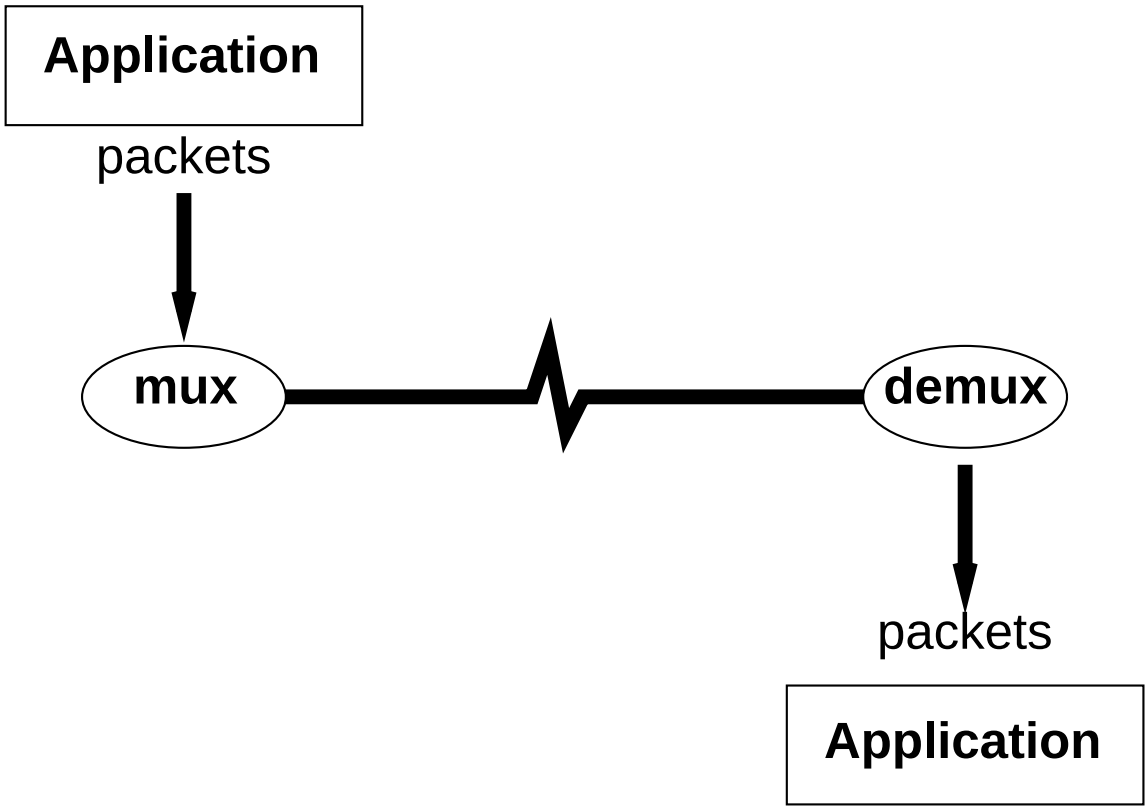
Application policy may be too involved to allow a useful application/protocol split.

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.

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



(This translates nicely into IP/UDP plus a particular application architecture.)

Basic Lightweight Sessions model

'Session' is multicast address + port.

Sites with data send it.

All sites quasi-periodically multicast '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.

An Aside on Resource Reservation

There are two schools of thought on Resource Reservation. One holds that it is necessary in any network trying to support interactive voice and video ('integrated services'). Lixia Zhang's RSVP is a model for how to do resource reservation in an IP multicast environment. The messy details are being pursued in the IETF RSVP and ISIP working groups.

The other school holds that resource reservation is unnecessary and a more efficient, adaptive, equivalent is to use layered codings to supply a range of bandwidth choices and assign the different layers to different multicast addresses. Then multicast pruning (with the added ability to prune particular sources) can be used control distribution of the layers.

Resource Reservation (cont.)

The jury is still out on these two alternatives (both are currently being prototyped) but the final answer will probably be a mixture rather than either extreme.

It's too early to predict because resource reservation and bandwidth allocation is not a technical problem — it is primarily a social problem and the answer will depend on the social and economic evolution of the Internet.

‘Session’ Packets

Early on in large-scale conferencing tests over the MBone, we found that conventional network management tools are useless for diagnosing multicast problems.

The best diagnostic seems to be to solicit reports from the all the receivers about how well they’re receiving the session’s data (report contains packets lost and interarrival variation — see RTP Internet Draft).

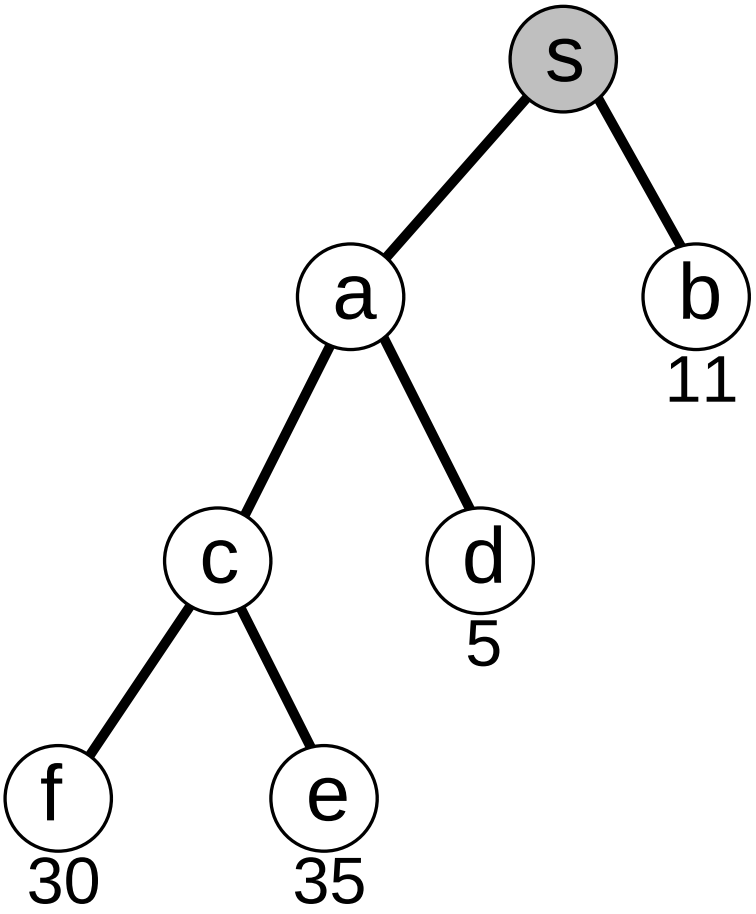
‘Session’ Packets (cont.)

These reports are multicast so any participant can answer questions like “is everyone getting lousy audio or just me?”

They also can be used by adaptive senders to tune their rate to changing network conditions (see Ian Wakeman’s paper in this conference).

‘Session’ Packets (cont.)

Can also be combined with multicast topology info to construct a sink tree for fault isolation:



‘Session’ Packets (cont.)

Since conferences can be arbitrarily large, don’t want session packet traffic to increase with number of participants.

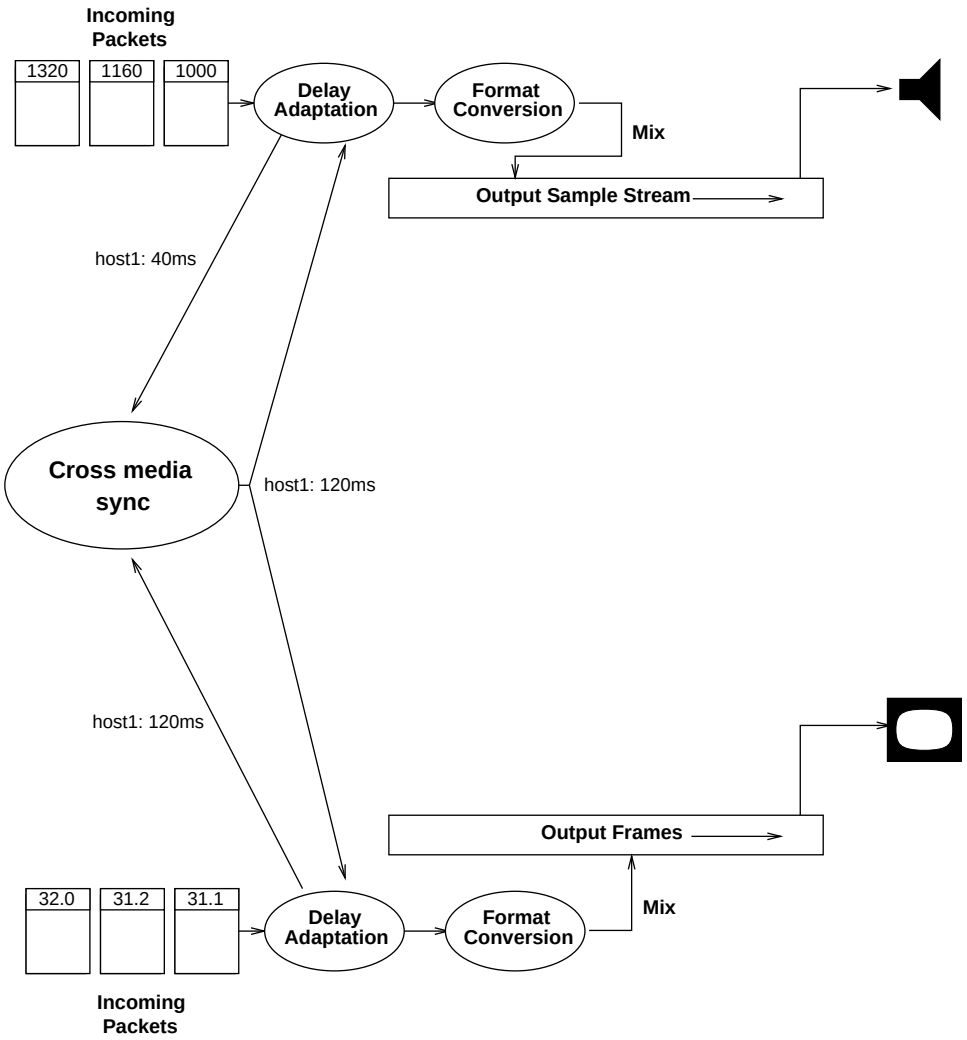
Solution is to say that total rate of session packets is fixed then let participants compute their local send rate based on knowing target rate and number of participants.

E.g., if rate target is 10 session msgs / sec. and there are 5 participants, each one should send an average of 2 msgs / sec. (but randomize send times between 250 and 750 ms so that aggregate traffic doesn’t synchronize).

We pick target rate as small fraction of data bandwidth (e.g., 5% of 64kb/s audio session) so total ‘control’ traffic always negligible compared to data traffic.

‘Session’ Packets and Synchronization

Where it’s desirable to do cross-media synchronization (e.g., audio and video), just need to use info in session packets from each media stream to drive playout delay adaptation machinery:



‘Session’ Packets and Synchronization (cont.)

Note that this requires certain semantics on timestamps in data and session packets: Sender must timestamp each media such that when timestamps are aligned, media will be in sync.

I.e., timestamp marks when acoustic wave hits mike & photons hit the CCD, not when applications first saw the bits.

Also requires that it be possible to match up participant identifiers from each of the separate media.

(RTP spec gives detailed description of data & machinery to do this.)

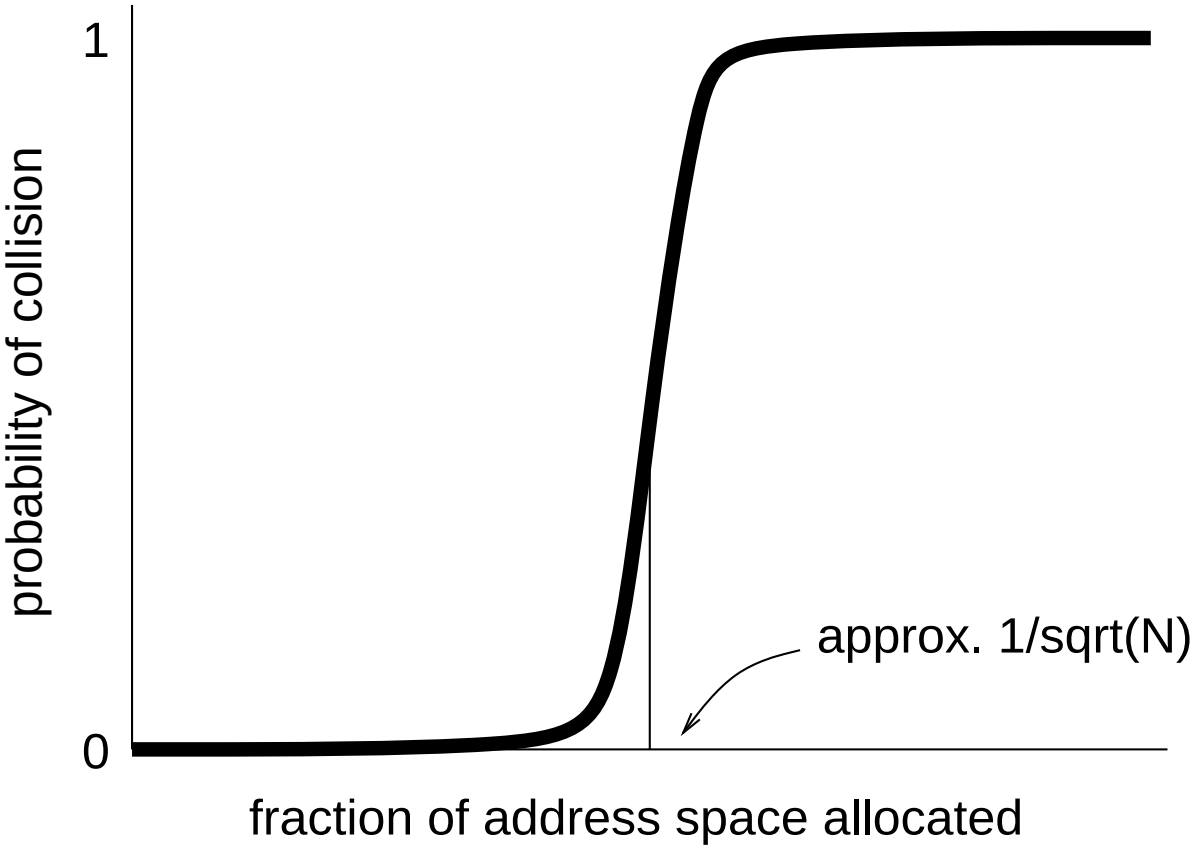
Multicast Address Allocation

Since ‘sessions’ are identified by multicast address(es), we need to do dynamic, distributed allocation of addresses.

There have been a number of academic proposals made on how to solve this problem. (Some of them have a rather tenuous grip on reality.)

Multicast Address Allocation (cont.)

A good solution requires that the probability that two different users will allocate the same address (collide) be very small. This means that behavior will be dominated by the statistics of the 'Birthday Problem':



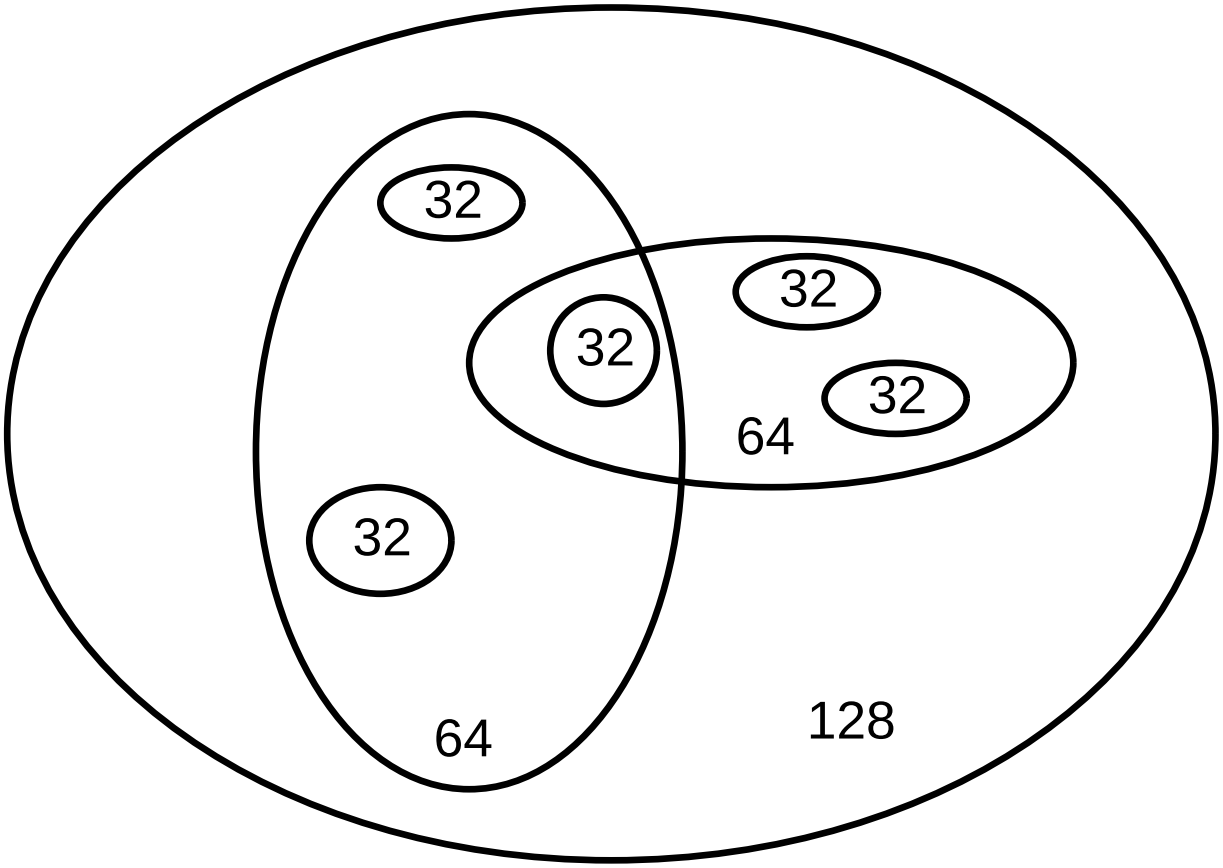
Multicast Address Allocation (cont.)

Basically the behavior is that when allocating from an address space of size N, if the number of addresses allocated is $< \sqrt{N}$ then the probability of collision is negligible but when there are more than $\sqrt{N}$ addresses allocated, the probability of collision is 1.

Since the IPv4 multicast address space is $\approx 2^{28}$, about 2^{14} (16K) addresses can be dynamically allocated before running into problems.

Multicast Address Allocation (cont.)

16K is clearly a small limit but you also have to take into account **Scopes** (different sessions have different ranges)



and **Lifetimes** (most sessions have a relatively short life).

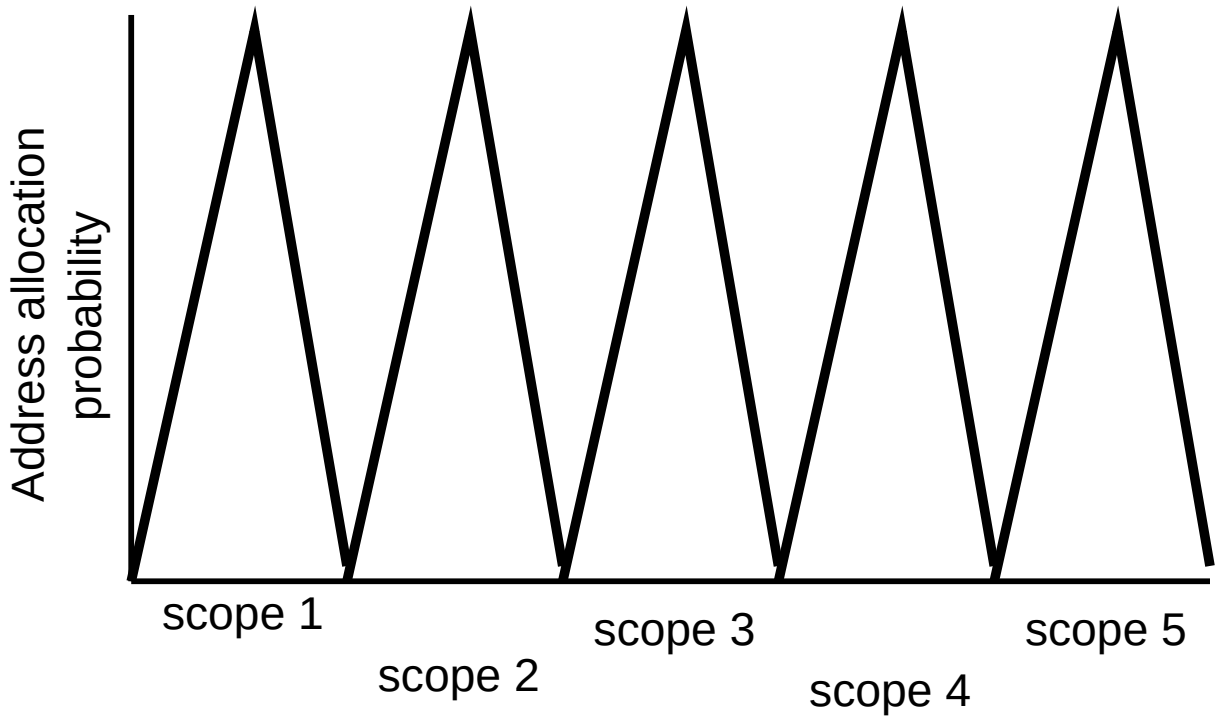
Multicast Address Allocation (cont.)

Two sessions can only conflict if they overlap in both space and time. So problems really occur with 16K simultaneously active sessions visible at some place in the topology. This is not such a small limit.

Sd's assignment algorithm ('informed, partitioned, random allocation') explicitly accounts for scope and lifetime. It's loosely based on an algorithm for dynamic, distributed, trunk allocation done at Bell Labs in the '60s.

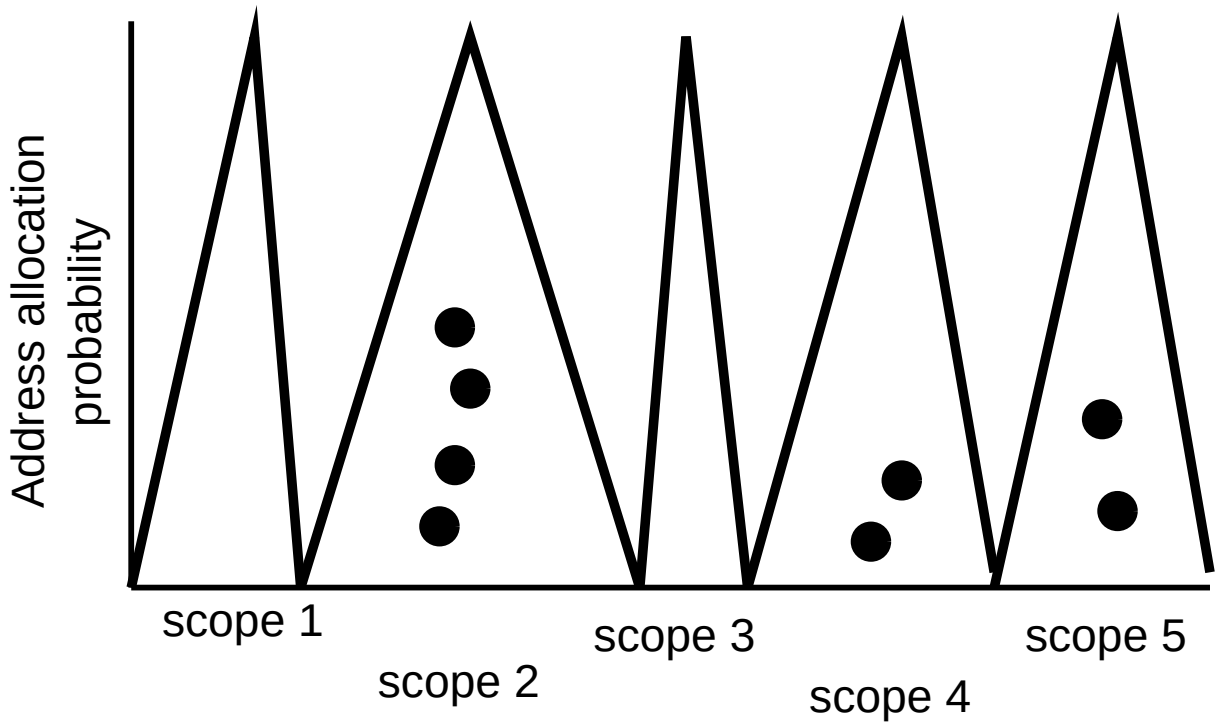
Multicast Address Allocation (cont.)

The address space is uniformly partitioned into regions for each scope:



Multicast Address Allocation (cont.)

As address allocations are made (either by the local sd or announced by remote sds or address discovery daemons), they cause the associated partition to expand, compressing its neighbors:



The result is a dynamic equilibrium where the fraction of the address space available for assignment in each partition tunes itself to the number of active addresses in that partition.

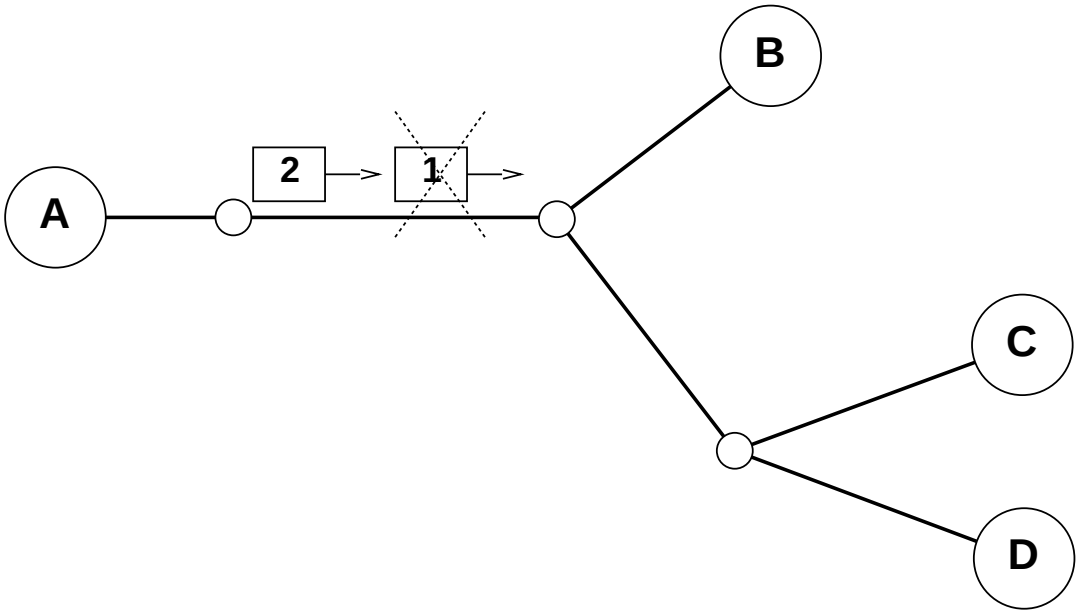
A ‘Social’ Note on Address Allocation

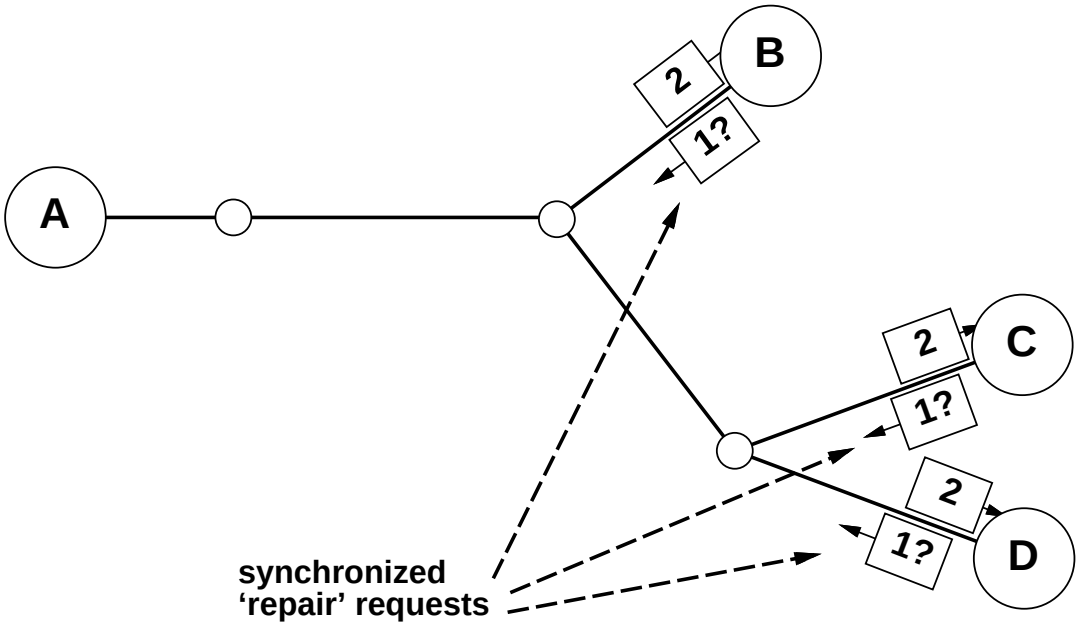
If building a dynamic address allocation tool (like sd), remember that people have great incentive to create sessions (they want to announce their event) but no incentive to delete sessions afterwards.

Unless you make deletion automatic (i.e., **require** start **and end** times for every allocation) space will fill up with obsolete, unused sessions and dynamic allocation will become impossible.

Reliable Multicast

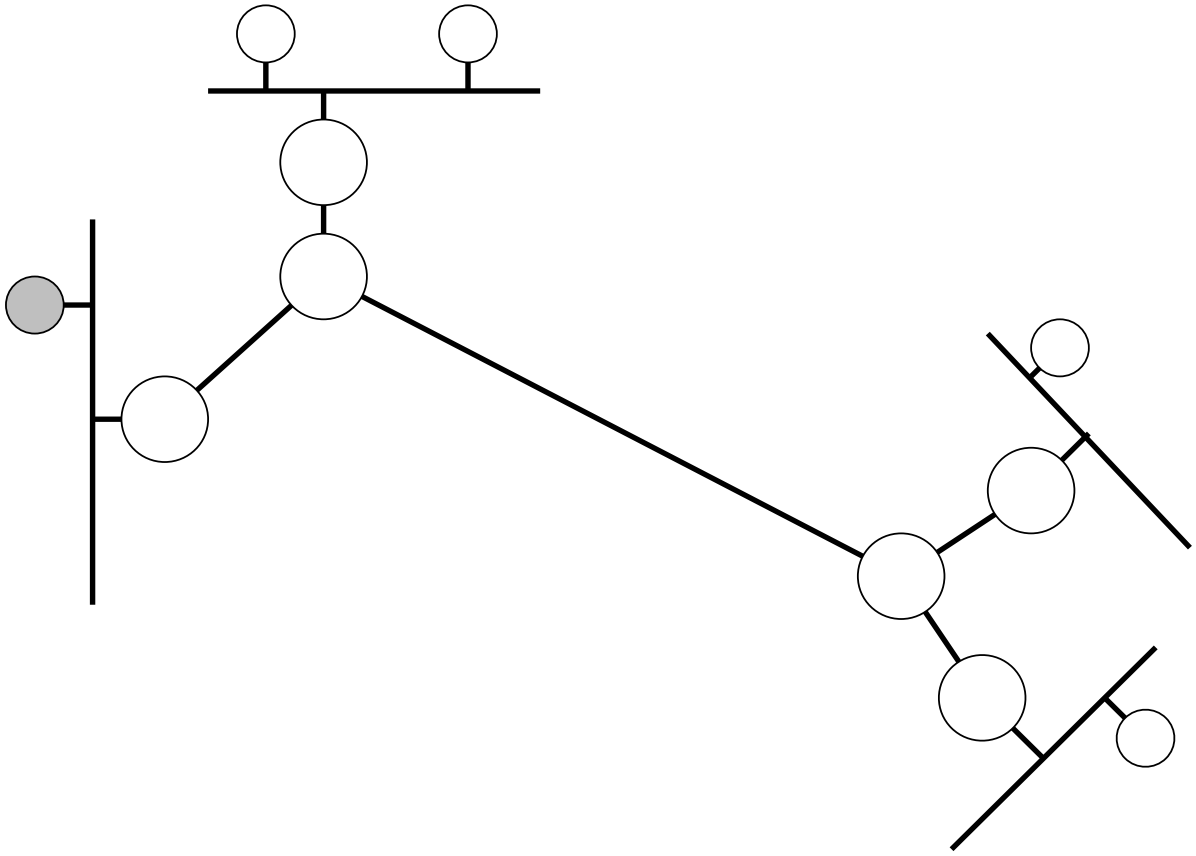
‘Reliable’ delivery is complicated by ‘ack implosion’ problem:





If we multicast repair request and response, can use those multicasts to suppress duplicates.

This requires adding some random jitter before requesting or responding.



To maximize response time, want host closest to point of failure to request repair.

This requires each host estimate its distance (in time) from every sender.

Closer hosts will pick a smaller randomization interval than distant hosts.

Also want host closest to requestor to respond with repair data.

Implies response timers work similarly to request timers.

Since anyone should be able to supply data, also implies that data identity be distinct from sender identity. (This is very different from conventional protocol like TCP where dialog always in terms of peer-peer communication state.)

Avoiding ack implosions (receiver)

1. If new hole detected, allocate ‘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)

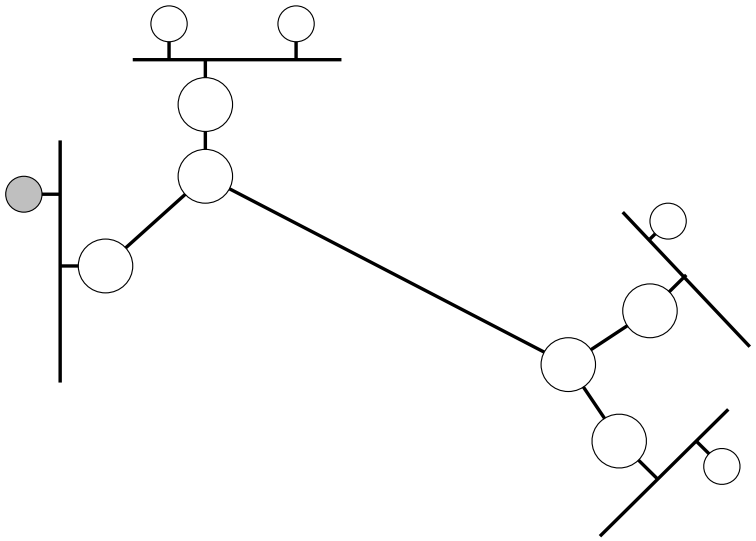
If repair request received for data that you have

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 protocols, not layered models or one-size-fits-all 'real-time' protocols.

- use multicast for everything
  - allows distribution problem diagnosis and allows participants 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.