Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.892 Fall 1999 **Adaptive multicast applications** November 3, 1999

Overview. IVS. RLM.

1 IVS

- Problem: how to obtain feedback from multicast group members in a scalable manner.
- Paper has an elegant solution to the *implosion* problem.
- Does not focus as much on the control loop, i.e., on what the sender does with this information.
- Some possible solutions:
  1. Probabilistic receiver response: receiver responds to source with some probability. Ideally, probability depends on size of session, which we don't know. Source not guaranteed to receive response from worst receiver within a guaranteed time.
  2. Random-delay receiver response: how to set delay? Depends on size of session. This is used by the audio tool *vat* to time the state multicasts from multiple receivers.
  3. Increasing scope search: too coarse-grained. And often, the worst receivers are farthest away.
- The IVS scheme uses a clever randomization scheme based on a sliding key scheme to achieve scalable feedback.
- How does it work?
  1. Source solicits responses from receivers in *rounds*; a sequence of rounds defines an *epoch*.
  2. In each epoch, source and receivers generate random bit-strings of length i.
  3. In each round, source sends out its key and indicates how many digits (from the left) of the key are significant.
  4. There are three types of messages sent from the source: SIZESOLICITED, STATE (implicit if SIZESOLICITED is unset) and RTTSOLICITED.
  5. If a receiver's key matches the indicated number of bits, then it replies with its current STATE (UNLOADED, LOADED, CONGESTED) to the sender. A receiver replies only if its current perceived STATE is at least as bad as what's advertised by the sender in this round.
  6. The longest (worst) round-trip delay is estimated by the sender using the *rttshft* field and setting it to some value. Q: what should this value depend on? (A: estimate of size of session.)
  7. In response, the receivers pick a random number in $(0, 2^{rttshft})$ and respond to the sender at that time. Q: how does the rtt get computed by the sender? (A: based on timestamp echo from receiver together with random delay time at receiver, which is also sent back.)

- Some issues:
  - How do receivers perceive congestion? How might receivers perceive a LOADED state? A: losses; jitter.
  - How does the sender respond? Drop rate multiplicatively, increase additively (all receivers UNLOADED causes this).
  - IVS drops rate only if some configurable fraction of the receivers are CONGESTED. Their implementation used 1.4% because that corresponded to a six-round (mean) difference between the SIZESOLICITED message and a corresponding CONGESTED response from some receiver.
- Analysis: why does it work?
  - Notation: n = number of receivers, i = length of key in bits, r_j = number of replies sent in round j and p_j = prob. that receiver replies in round j.
  - $P(r_j > 0 | r_{j-1} = 0) = (1 - (1-p)^n)(1 - 2^{j-1-i})^n$
    Here, $p_j = 2^{-i}$ if $j = 0$ and $2^{j-1}/(2^i - 2^{j-1})$ if $j > 0$.
  - Expected time to first hit small (about 6 with 16 bits for i).
  - Implosion eliminated; in each round only a small number of receivers respond each round until first few tens of hits.
- Drawbacks/critique:
  - Heterogeneity not addressed.
  - Receiver feedback from a "troubled" receiver could take a long time to reach the sender.
  - Experiments in paper hardly conclusive (but analysis is sound).

2 RLM

- Problem: how to handle rate adaptation of video transmission in the presence of bandwidth heterogeneity.
- Most previous schemes progressed essentially at the rate of the slowest receiver, penalizing better connected hosts. But if you transmitted at a high rate to satisfy well-connected receivers, you end up swamping most other receivers. *One rate does not fit all!*
- We know (e.g., IVS) how to obtain receiver feedback avoiding implosion. Is that enough? A: no, because we would only be able to go at rate of the slowest (or close-to-slowest) receiver.
- Key idea (folklore): Use layered video. One of two kinds: base layer with enhancements, or *simulcast*.
- With this, different receivers, depending on their connectivity, can pick and choose which layers they get.
- But how do you selectively forward flows in the network? There are at least two approaches to this:

1. Network support: Use priority-drop gateways. Source prioritizes data with higher layers at lower priority. When congestion happens, these packets are preferentially dropped. Q: what is claimed problem with this? A: poor incentive properties. We will explore this issue in detail next class. (Briefly, the claim is that if you do priority dropping, there's no incentive for flows to discover the sustainable rate of the network, and they will end up driving the net to congestion.)
2. Multiple multicast groups with a receiver-driven join/leave mechanism. 1-1 mapping between layer and multicast group. Keeps the network simple and can be deployed today. This paper describes how you should design the control algorithms to decide when to leave, how to probe for bandwidth and join a higher layer, etc.

- When should you leave a group (layer)? Easy, upon congestion, as detected by packet loss at receiver. When packets are lost (regardless of layer), unsubscribe to highest layer. (But only when you *do not* have notification of a join experiment for a layer $\leq$ your highest layer in progress, as explained below.)

- Tricky part is how to join higher layers, probing carefully for spare network bandwidth. The paper has an elegant end-to-end scheme to orchestrate *join experiments* by receivers to do this in a scalable and safe manner.

- Q: When should join experiments be done? A: Frequently when they are likely to succeed and seldom when they are likely to fail. This is an example of *adaptation* to congestion based on learning network characteristics.

- At each layer, every receiver has an adaptive *join timer*. Upon a successful experiment, move up a layer. Upon an unsuccessful experiment, multiplicatively backoff the join timer. Need a *detection time* to determine if a join experiment is successful or not.

- Problem: independent join experiments. Leads to poor scaling because more and more network bandwidth is used up in join experiments as the session grows in size.

- Solution:*shared learning* of experiment results.

- Before experiment, notify group that you're starting an experiment at a particular layer. Now, if within the detection time of that notification, if any other receiver sees packet losses, it attributes it to a problematic layer. In turn, it multiplicatively backs off a join experiment for that layer.

- Q: Can we also add if someone else's join experiment seems not to have caused any losses for us? A: No! (Because they may not share a congested path with us.)

- What if you have multiple join experiments outstanding? Causes problems. Reduce occurrence of this using multicast suppression for higher-layer experiments. (Suppress join experiment if you hear someone doing an experiment for a lower layer, because our higher-layer experiment may adversely interact with their lower layer one.)

- Key principles:
  - Shared multicast control.
  - Adaptation.
  - Receiver-based design.

- Simulation results are reasonable. Performance better with RED than drop-tail. Interactions with TCP slow start?
- Network implications for RLM-like protocols:
  - Receiver-consensus.
  - Fast leaves and joins in group management protocol.
  - Fairness: RLM *does not* achieve fairness in isolation.
  - Priority-drop gateways interfere with scalability of RLM, because a failed experiment for a higher layer will not be noticed by another node on the same shared congested link. This affects the effectiveness of shared learning.