Reliable Multicast - Where are we and where are we going?

Mark Handley

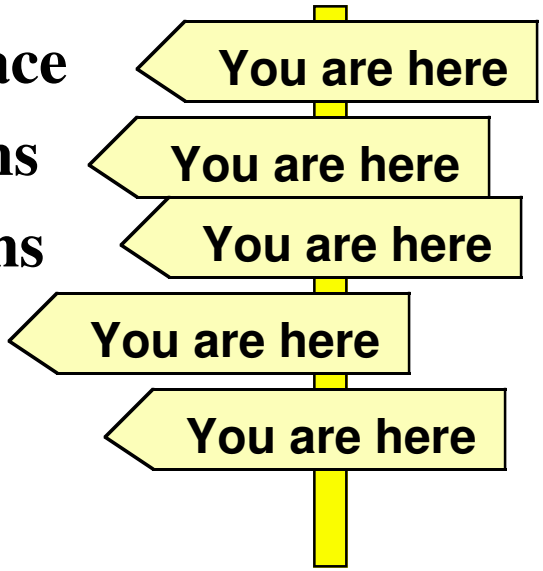
USC/ISI (North East)

An Outline for RM Research

- **Constrain the design space**
- **Identify the key problems**
- **Identify possible solutions**
- **Build testbeds**
- **Iterate**

An Outline for RM Research

- **Constrain the design space**
- **Identify the key problems**
- **Identify possible solutions**
- **Build testbeds**
- **Iterate**



Constrain the Design Space

The idea of a general purpose Reliable Multicast Protocol is an oxymoron.

- **Too many conflicting design goals**

So what problem are we trying to solve?

- **Lets just concentrate on one class of applications today.**

Distributed Data Shared Applications

Applications such as wb or NTE where the data set is held by all sites.

Architecture is symmetric peer-to-peer.

- **no site is (a priori) more important than others**

Missing data can be re-generated when required by the application if retransmission is needed

- **no problem with a sender needing to know when it can flush sent data from its transport protocol send buffer.**

Would like the application to scale to 1000s of participants.

- **may or may not care about millions of *receivers***

So what are the general properties that we end up with for such an application?

Surprise surprise - they look remarkably like SRM.

- ***NACK* based**
- ***ALF* style**
- ***Session messages* convey current state so missing data can be discovered.**
- **Receivers *request* missing data to be retransmitted**
- **NACK *implosion avoidance* mechanism.**

But not necessarily completely like SRM

- **Discovering data is missing is not so simple if you allow multiple sites to modify shared objects.**
- **Consistency may be hard too if you don’t choose to use locks or tokens.**
- **Can use redundancy to reduce the need for retransmissions.**
- **Can use FEC to repair loss.**
- **SRM doesn’t say anything about congestion control**

What are the problems?

- **NACK implosion avoidance**
  - **ideally no more than one NACK per loss**
- **Minimise retransmission requests and replys**
  - **spatially uncorrelated loss and broken minority of receivers can result in large proportion of packets requiring retransmission**
- **Congestion Control**
- **Consistency Control**
  - **allowing multiple people to modify an object over time**
- **Object naming**
- **ALF**

NACK implosion avoidance

SRM provides a good solution

- **scaling is a problem though**

we can modify basic scheme in several ways:

- **adaptive receiver delays to elect "representatives" of groups of receivers suffering loss.**
- **choose from non-uniform delay**
- **use other application-level distinctions to tune timers**

There are completely different solutions too:

- **NTE's sender-based sliding key scheme**
- **this may be useful if you don't have a multicast back channel as is the case with satelite multicast schemes.**

Solutions: Minimise retransmission requests and replies

- **Exploit natural application level redundancy.**
  - **eg, lines of text in NTE**
- **Incorportate additional packet level FEC schemes.**
  - **see "Parity" paper in Sigcomm 97**
- **Perform local retransmission for persistently problematic links.**
  - **this may also be key to congestion control mechanisms**

Probably we should do *all* of these.

Natural Application Level Redundancy

- **This occurs when linked events occur in rapid succession and we want to transmit them all immediately for usability reasons.**

  Examples: typing in an editor, drawing in wb
- **The events generated are typically small, packetisation overhead is large.**
- **Send the whole "next-level" hierarchical object (eg line of text rather than character, freehand curve rather than line-segment) in every packet.**
- **So long as the receiver can tell it doesn't need the individual events there is no need for retransmission as the later events obsolete earlier events.**
  - **In NTE, the reciever doesn't even know the earlier events occured as there are no sequence numbers, only timestamps.**

Packet level FEC

(see Kurose at al, Sigcomm 97)

Most useful for "bulk" transfer:

- **Sender sends a set of packets**
- **Receiver(s) sends the number of packets lost**
  - **can weight SRM request timer by number of packets lost**
- **Sender computes this number of parity packets and sends them.**
  - **use a variation on Reed-Solomon erasure codes.**
- **The *same parities repair different lost packets* at different receivers.**

Local Retransmission

- **Lixia?**

Congestion Control

This seems to be THE HARD PROBLEM in RM

- **No general solution exists (yet).**
  - **probably easier for true bulk transfer RM apps**
- **To some extent, DD applications may have a "natural data rate" corresponding to the drawing in wb or typing in NTE.**
  - **Should we be attempting congestion control on this traffic?**
- **Then there are the "bulk" transfer parts such as downloading postscript in wb or a new document in NTE.**
  - **There is no natural rate (would like to go as fast as possible)**
  - **We must perform some form of congestion control on this traffic.**
  - **How fast is "fair" and how to discover it?**

TCP friendly congestion control for Reliable Multicast

Can we use a modified TCP congestion control scheme? Many Problems:

- **no ACK clock**
  - **need a rate-based variant of TCP mechanisms**
- **delayed feedback**
  - **adaptive timers to elect representatives suffering worst representative loss *rate* which respond fast**
- **uncorrelated loss**
  - **send parities on demand instead of retransmissions**
- **lowest common denominator performance at best**
  - **spilt off particularly bad sites for local *transmission* using regular TCP to their representative which re-multicasts the data. (assumptions about tree here!)**

TCP friendly congestion control for Reliable Multicast

Use the equation for expected TCP throughput to derive the equivalent TCP throughput from the loss rate of the worst receiver.

- **Many pitfalls:**
  - **assumptions of the basic equation**
    - » **random loss**
  - **stability of resulting systems**
  - **makes high TCP induced loss rates a self-fulfilling prophecy.**

Consistency Control: User Requirements

1 - many people should be able to manipulate the same data object over time.

2 - eventual consistency

- **– everyone should end up with the same dataset**

3 - fully interactive

- **– I don't want to have to wait for locks or such like**

4 - deterministic behaviour

- **– if I typed it I want it to stay typed...**

The problem is that these requirements are contradictory given unreliable networks.

Consistency Control: Realistic User Requirements

- **Many people should be able to manipulate the same data object over time.**
  - **– wb doesn't allow this**
- **Eventual consistency**
  - **– probably can't compromise on this**
- **Fully interactive**
  - **– most systems compromise this in the presence of loss**
  - **– (eg anything containing locks!)**
- **Deterministic behaviour**
  - **– relaxation can lead to good performance and interesting temporary consistency problems. (eg NTE)**

Relaxed Consistency in NTE

- **Timestamp all data and provide application clock synchronisation.**
- **Take the latest version of the object.**
- **Strict rules on the data set and its manipulation to ensure that inconsistencies can never occur in a manner that is locally undetectable.**
  - **deleted objects must remain *in place* in the data set.**
- **Sometimes you can get in a consistent mess!**
  - **tend to delete and re-insert a lot to prevent this.**

A hybrid approach

- **Design an application level framework that permits different objects to relax different constraints.**
- **Limited state transition possible even for the same object.**
- **Can build this on top of something like SRM but it's not completely divorced from the transport protocol because there are explicit rules as to who can do what with a particular object.**

Object Naming

- **closely related to other parts of the problem/solution**
  - **discovering inconsistencies using the minimal amount of information in session messages**
- **NTE used a flat namespace**
  - **this was probably a mistake**
  - **cannot infer informtion from references to the object that helps with deciding priorities.**
- **probably should contain information related to the application data model**
  - **this will vary from application to application.**
  - **probably it will be hierarchical.**
- **versioning as part of the name?**
  - **probably not a good idea in the general case.**

ALF is a problem?

It is for *generalisation* and *code reuse*.

- ***No layer abstraction* means everything ends up in the application.**
- **Application *semantics* influence everything.**
- **Generalisation and performance seem contradictory in many aspects of RM**
  - **eg: exploiting application level redundancy**
  - **eg: consistency mechanisms**

Code reuse is the Holy Grail of software engineering.

- **Perhaps MASH comes close?**

Summary

We have a set of ingredients.

- **although we'd like more on the congestion control shelf.**

More than one type of DD cake is possible right now.

Perhaps it's time for the RM group to discuss recipies?