Oxymorons

- **Military Intelligence**
- **Plastic Glasses**
- **United Nations**
- **Business Ethics**
- **Chili**
- **Computer Science**
- **Reliable Multicast Protocol**

The concept of a general purpose Reliable Multicast Protocol is an oxymoron

- **with a reliable unicast protocol, there is inevitable fate-sharing between sender and receiver**
  - **failure means abandonment**
  - **there is an appropriate rate that avoids congestion**
  - **there is one definition of consistency**
- **with multicast, the design space is much larger**
  - **may or may not know all the receivers**
  - **receivers may join and leave "part-way though"**
  - **failure to a single receiver may or may not be important**
  - **consistency may be strict or relaxed**
  - **there may be no appropriate rate *that we can discover* that avoids congestion to all receivers**

Assumptions

- **A significant number (>100) of users participate in any session we are designing for.**
  - if possible mechanisms should cope with very large numbers (thousands to millions) of users, though this may not always be appropriate for all tools.
- **The network is heterogeneous, but that in many (most?) places our traffic is a small fraction of the total traffic.**
  - fast backbones, slow subscriber loops, etc
  - both transient and persistent congestion may occur in multiple places in the multicast tree.
  - probability of any particular packet being lost by at least one receiver is significant.

Categorisation Factors

- **Number of sources**
- **One/many/no start time**
  - **many => data sources must provide history**
  - **no => no history**
- **Realtime requirements**
- **Consistency requirements**

Non factors(?):

- **Congestion Control**
  - **necessary for all, but mechanism may vary**
- **Ordering**
  - **seems to depend on application more than category**

Categorisation of RM Applications

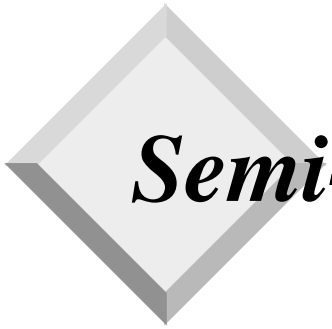
- **Bulk-data transfer**
  - **usenet news**
  - **VR dataset distribution**
- **Live-data feeds**
  - **stock prices**
  - **press news feeds**
- **Semi-reliable multicast**
  - **streaming of “loss-tolerant” (eg: audio) data**
- **Shared-data applications**
  - **shared whiteboards, shared editors**
  - **distributed dataset**
- **Hydrids**

Bulk-data transfer

- **Examples:**
  - **Usenet news feeds**
  - **Mailing list distribution**
  - **Web cache pre-load**
- **Properties**
  - **one-to-many distribution**
  - **“relaxed” real-time requirement**
  - **synchronised start time**
  - **only “file” level consistency required**
  - ***large commonality of requirements?***

Live-data feeds

- **Examples**
  - **stock prices**
  - **press news feeds**
  - **event logging**
- **Properties**
  - **one data source**
  - **no start time (no retransmission of old data)**
  - **quasi-real-time requirement**
  - **may need synchronised consistency (either everyone gets it or no-one does).**
  - ***commonality of requirements depends on consistency requirements***

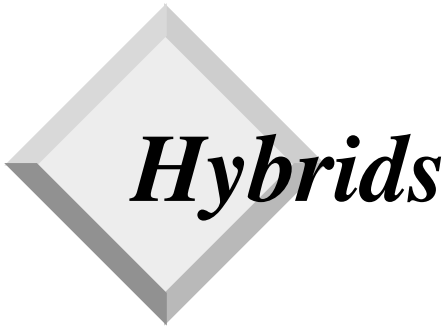


Semi-reliable multicast

- **Examples**
  - **streamed video and audio from a multimedia server**
- **Properties**
  - **one data source**
  - **single start time**
  - **data stream is being consumed in realtime**
  - **data stream may be streamed faster than realtime**
  - **limited time window for retransmission**
  - **no consistency requirement**
  - ***large commonality of requirements***

Shared-data applications

- **Examples**
  - **shared whiteboards, shared editors**
  - **conference control**
- **Properties**
  - **distributed dataset**
  - **all sites hold all (or most) data**
  - **many start times (opportunistic retransmission of old data)**
  - **changes originating from multiple sites**
  - **relaxed (?) consistency requirements**
  - **real-time requirements**
  - ***small commonality of requirements***



Hybrids

- **Example: a VR system**
  - **unreliable distribution of tracking data and audio**
  - **shared-data properties when interacting with objects**
  - **bulk-transfer properties when downloading the environment**

Shared Data Applications: 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 - deterministic behaviour

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

4 - fully interactive

– I don’t want to have to wait for locks or such like

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

Shared Data Applications: 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)

One possible 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.**

Why categorise?

- **Lack of a “one size fits all” solution.**
- **Don’t want to reinvent the wheel for every application.**

So it this taxonomy useful?

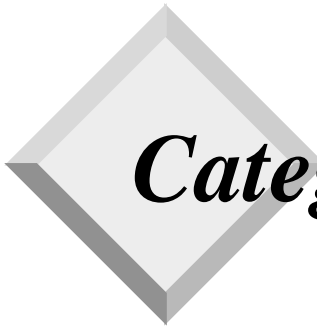
- **Depends on commonality.**
- **Depends on what the hard problem is.**

The hard problem is congestion control

- **Does this categorisation give us a handle on congestion control?**
  - **Solutions for some categories work well, but don't apply for all categories.**

The hard problem is scalability to large groups

- **Does this categorisation give us a handle on scalability?**
  - **local retransmission and FEC work much better for some categories than for others because the contsraints are easier**



Categorisation of RM Applications

- **Bulk-data transfer**
- **Live-data feeds**
- **Semi-reliable multicast**
- **Shared-data applications**
- **Hydrids**

Is anything missing from this list?

Are these sufficiently different to warrant different approaches?