Reliable Multicast

Mark Handley

Information Sciences Institute,

University of Southern California (in Boston)

Oxymorons

- **Military Intelligence**
- **Plastic Glasses**
- **United Nations**
- **Business Ethics**
- **Chilli**
- **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**

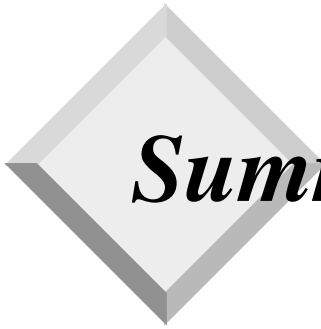
- **If you try to design an API to a general purpose reliable multicast protocol, you need the API to express all the semantics of the assorted reliability and failure modes**
- **Such an API ends up so complicated that most of the application is *beneath* the API!**
- **This leads us to the design principle of ALF - Application Level Framing**

Application Level Framing

- **Thin transport layer**
- **Data is framed on the network in units that have semantic importance for the application**
- **Failure modes and reliability are then the responsibility of the application, which is really the only place where these can be handled appropriately**

Network Behaviour

- **Before designing applications or protocols, examine the behaviour of the network on which they are to be used.**
- **mtrace:**
  - can trace a route through the network are report hop-by-hop loss rates
- **RTP/RTCP**
  - can monitor reception of video data to observe loss patterns
  - can monitor RTCP reception reports to observer loss correlation



Summary: Packet Loss

Congestive Loss

- transient in properly provisioned networks
- persistent in under-provisioned networks
- packet losses are effectively independent if...

- **Burst Loss**
  - so-called “Cisco Disease”
  - routing changes
- **Partitionings**
  - link/router failures
  - routing errors

Summary: General Aims

- ***survive*** **partitionings**
- ***prevent*** **burst loss**
- ***adapt*** **to reduce congestive loss where possible**

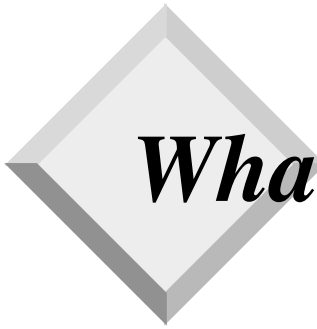
  or

  tolerate **it where not.**

This talk is mostly about tolerating loss and surviving partitionings.

Assumptions

- **IP multicast for efficient data distribution and as a rendezvous mechanism.**
- **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.



What about reservations?

They should help, but...

...lets see how far we can get without them...

NTE - a shared text editor for the Mbone

- **If we can't design a general purpose reliable multicast protocol, how about a special purpose reliable multicast application?**
- **LBL's wb shared whiteboard showed that this could be done in 1993**
- **In 1994, we'd been using wb for a year or so, and discovered that although it's a good whiteboard, we also needed a shared text editor for the MICE project meetings.**
- **I thought it would only take a few weeks...**

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

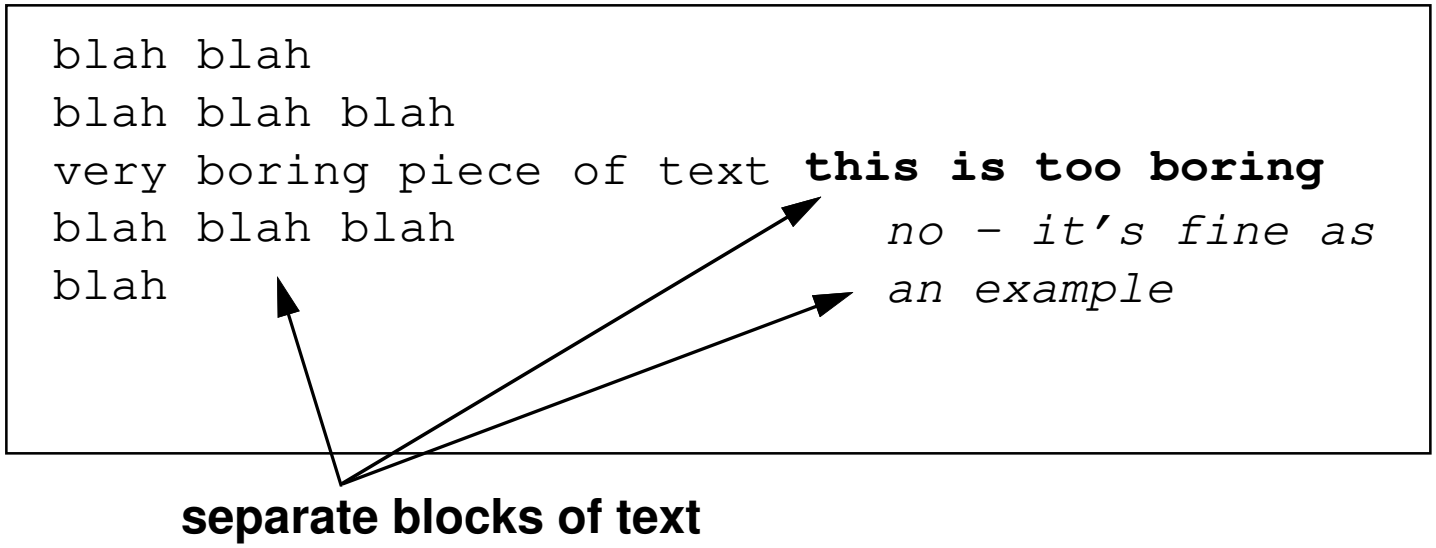
The problem is that these requirements are contradictory given unreliable networks.

Realistic User Requirements

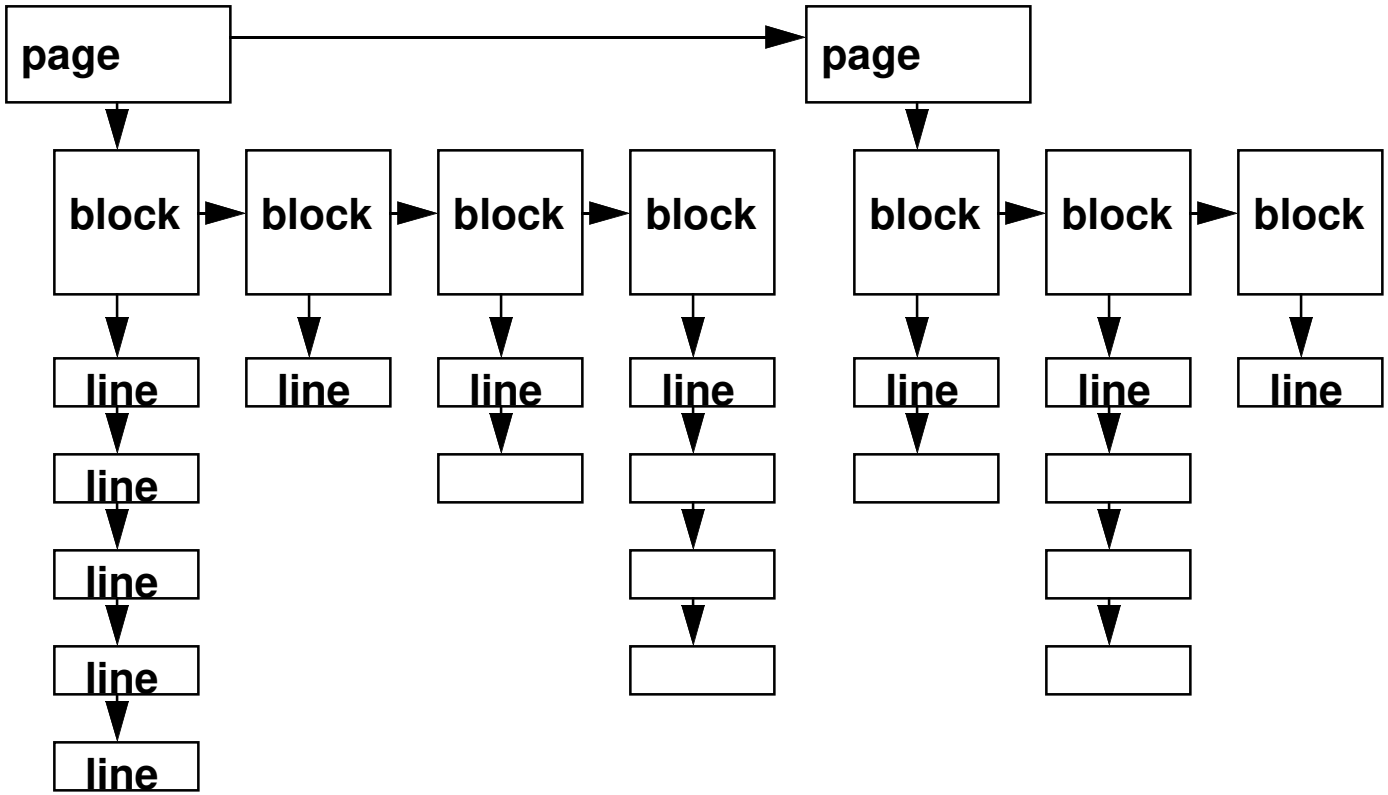
- **Many people should be able to manipulate the same data object over time.**
  - wb doesn't allow this, but we must do for a shared editor
- **Eventual consistency**
  - probably can't compromise on this
- **Fully interactive**
  - most systems compromise this in the presence of loss
- **Deterministic behaviour**
  - if I typed it I want it to stay typed... *unless*....
  - ***what happens if we relax this requirement a little?***

Nt design

Collaboration tends to dictate an editor that resembles a drawing package in that different users want to be able to work simultaneously and to add comments or annotations that are separate from the text being worked on.



Hierarchy of Objects



Each object is sent in a separate packet, has a globally unique name, and references the objects it is linked to.

Nt design

block and line based

↓

line is our ADU

↓

send whole line for each change

↓

gives great redundancy

for little additional overhead

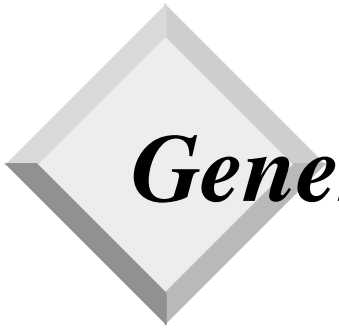
↓

but now how do we do reliability?

Redundancy and reliability

if you keep typing most loss is naturally corrected, and the meaning of what you're saying is conveyed without the need for any retransmissions

- thus we don't need (or want) all lost packets to be retransmitted if they'll be naturally replaced
- we don't want the retransmission scheme to be too fast
- we do want it to be bounded and predictable in the time it takes
- most importantly, we do want eventual consistency



General Idea

timestamp each change to a line and send the whole line, line-id and timestamp each time the line is changed.

if an incoming object has a later timestamp than our current version, the incoming version is correct.

needs 3 mechanisms:

- clock synchronisation
- an announcement mechanism
- a nack-implosion avoidance mechanism

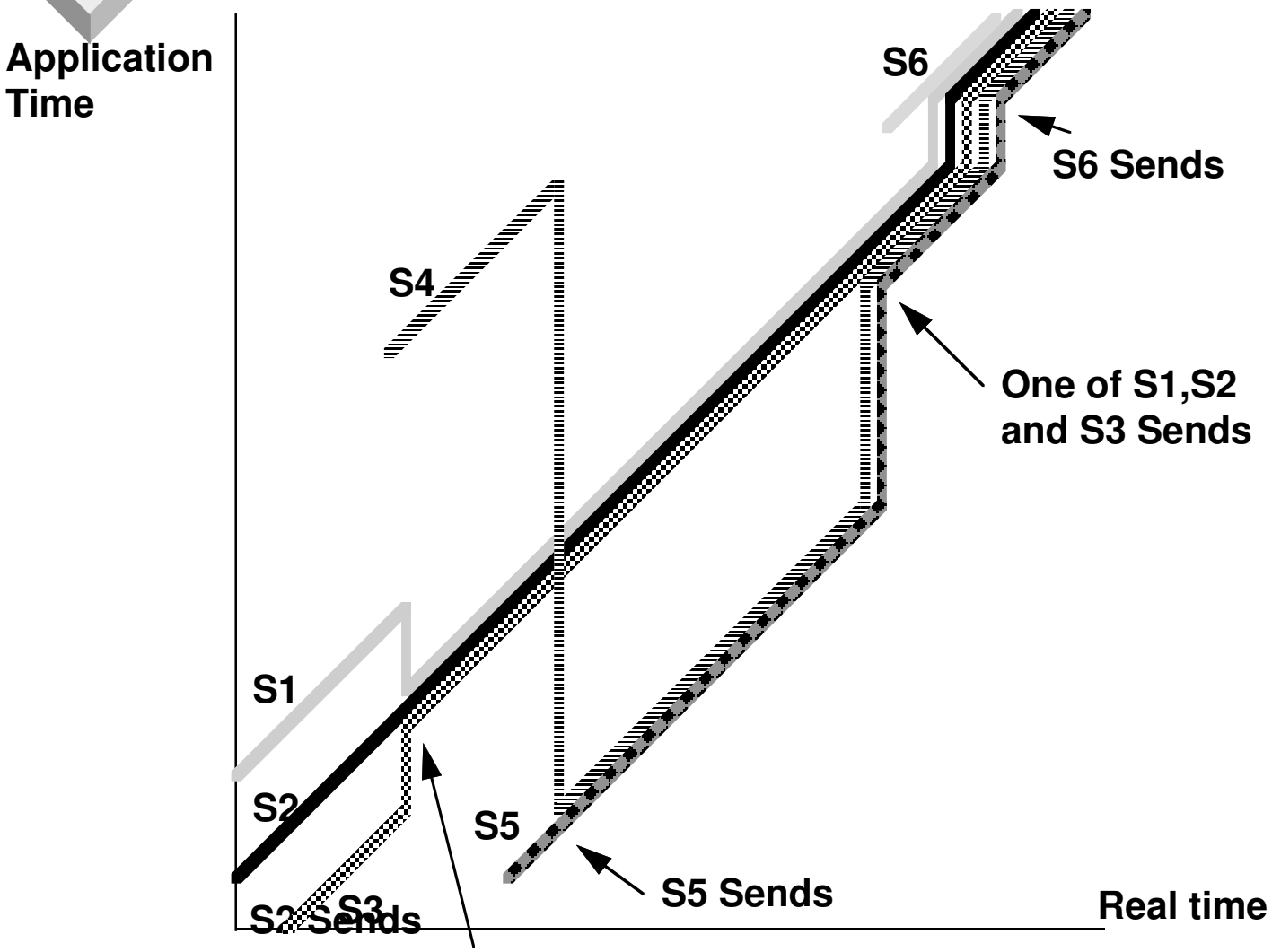
Clock Synchronisation

- **For our purposes, the basic requirements are that a clock never moves backward due to any adjustment, and that all clocks converge on some (arbitrary) time.**
- **Given that all changes are multicast, and that all packets contain timestamps, we have a simple mechanism for clock synchronisation**

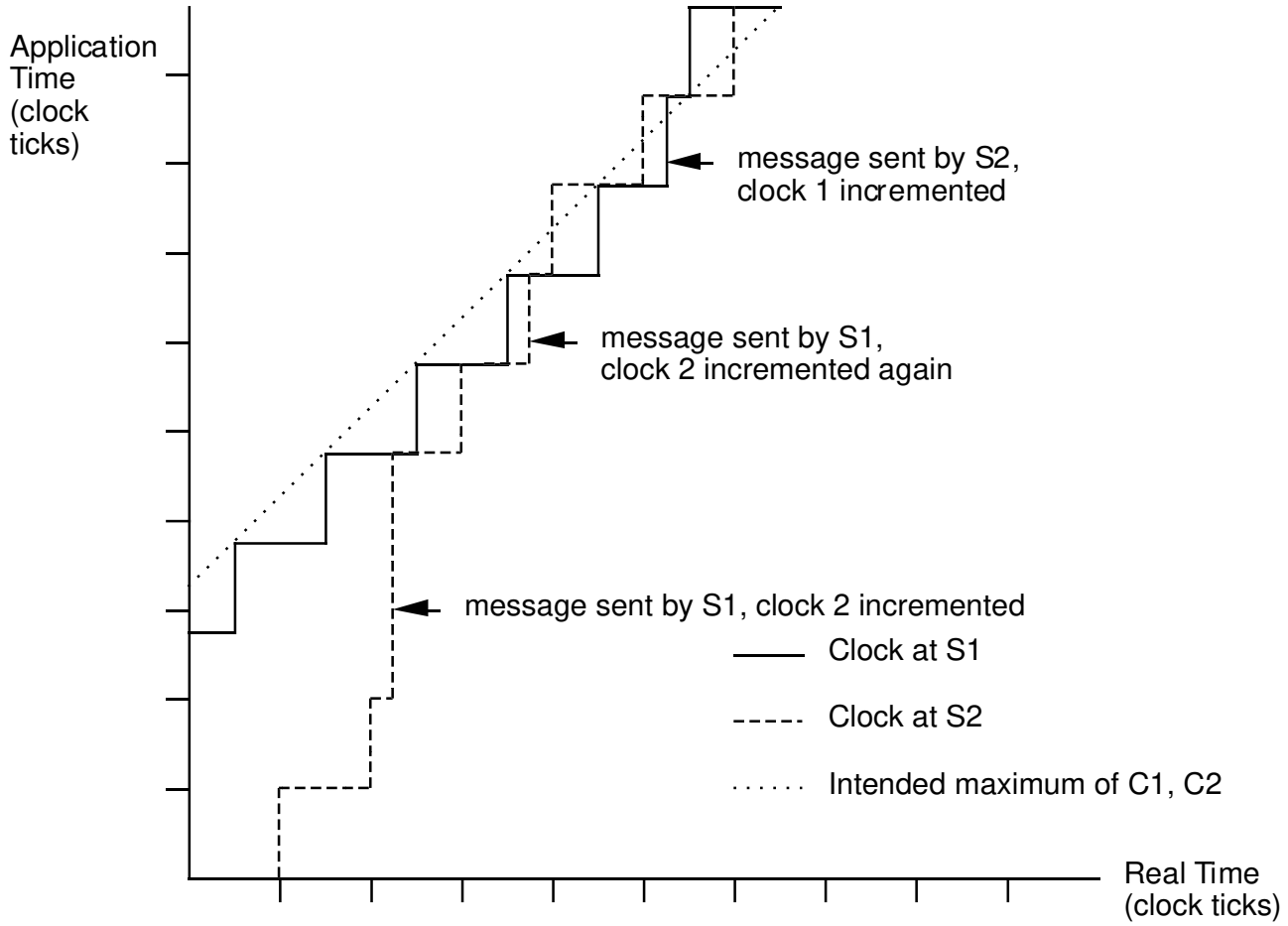
Clock Sync

- **if a site has not sent any data and receives data, it sets its app clock to the timestamp in the packet.**
- **if a site has not received any data and needs to send data, it sets its app clock to its own real-time clock.**
- **if a site then receives a message with a timestamp greater than its current app clock time, it increases its app clock to equal that in the received message.**

Clock sync


Application Time
S6
S6 Sends
S4
One of S1,S2 and S3 Sends
S1
S2
S5
S5 Sends
S2 Sends
S3
Real time

Clock Granularity Problem



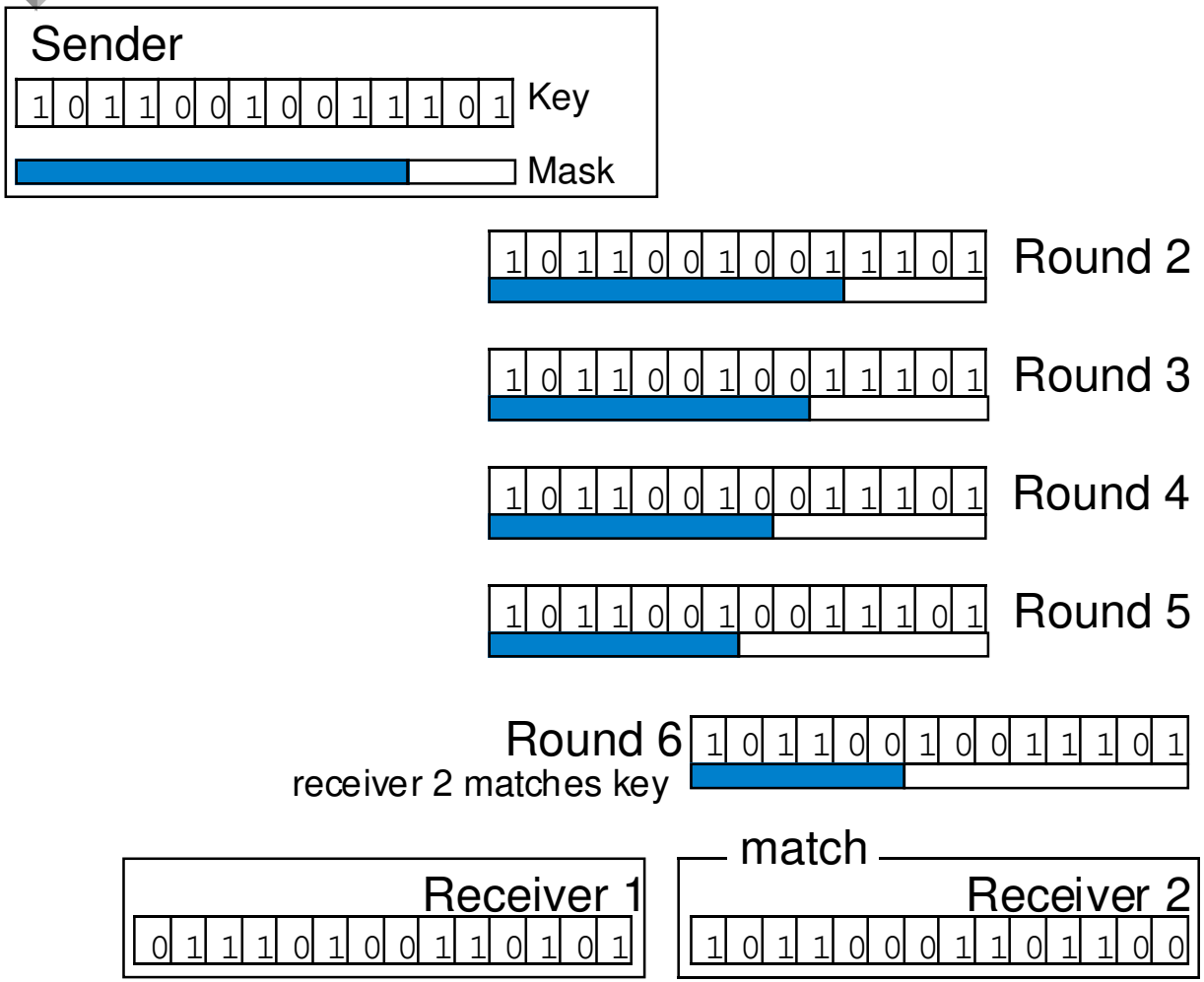
Announcement Mechanism

- **This is the mechanism by which sites discover that they have an out-of-date version of an object or a missing object.**
  - though missing objects are usually noticeable by references from other objects.
- **Two main situations where we need this:**
  - losing the last of a sequences of changes to a line.
  - recovery from network partitioning
- **We could distribute this mechanism**
  - have sites each list one recently modified object
- **We could "centralise" it**
  - elect a site to list all the objects that recently changed

Nack-implosion avoidance

- **This mechanism may not be optimal for general use, but it pre-dates any description of SRM, it is novel, and it works!**
- **Also it can be combined with the announcement mechanism.**
- **It is based on the sliding-key scheme devised by Ian Wakeman (see Sigcomm ‘94) for scalable congestion control for video.**

Sliding Keys



Sliding Key Mechanism

- **Avoids NACK implosion problem**
- **O(logN) delay before being able to make a request**
  - depending on whether others got loss too.
- **Simple to implement**
  - requires minimal tuning to work well.
  - SRM requires much more tuning and RTT matrix estimation.
- **Works well when combined with an announcement mechanism**
- **Would I do it this way now?**
  - Perhaps. Possibly I'd use an SRM based scheme...

There's much more...

- **Checksum mechanism for solving long-term partitioning.**
- **Constraints on line generation and deletion.**
  - basically, ordering of lines in a block can only be modified by creation of new lines.
  - deleted lines are left in place but no longer visible.
  - this allows many inconsistencies to be automatically corrected and the rest to be detected and then corrected by an explicit mechanism.
- **User-locking of blocks**
  - constrained to ensure deterministic behaviour

Lessons

- **Nt works well**
  - noticeable inconsistencies are rare (even with high loss)
  - human protocols rapidly evolve to avoid conflict
- **There are many reliable multicast schemes**
  - one shoe doesn’t fit all
- **Application specific clock-sync can be simple**
  - though being able to assume NTP would be better
- **Redundancy greatly complicates reliability!**
- **ALF applies to time as well as space**

Summary and Observations

- **ALF can and should be interpreted quite broadly.**
- **ALF applies to time as well as space.**
- **Although ALF is a design principle that can lead to tools that perform well, it's difficult to draw guidelines or make generalisations from many ALF applications.**
- **Designing collaboration tools from the network constraints upwards seems to result in network tools that perform better and therefore satisfy users better than designing them from the user requirements down.**