# *NTE: A Scalable Shared Text Editor for the Mbone*

**Mark Handley**

*Information Sciences Institute,*

*University of Southern California*

# *Network Behaviour*

- **Before designing applications or protocols, examine the behaviour of the network.**
- **Our analysis which influenced the design of NTE is available from:**

  **http://north.east.isi.edu/~mjh/mbone.ps.gz**
- **Key conclusion: in a multicast distribution tree, there are many points of congestion, some severe, most not. These result in receivers seeing significant spatially uncorrelated packet loss.**
- **Standard retransmission-based schemes will resend most packets.**

# *A Multicast Distribution Tree*

*NASA Shuttle Video, March 1996*

# *Oxymorons*

- **Military Intelligence**
- **Plastic Glasses**
- **Chili**
- **Reliable multicast**

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

- **late joiners need application support**
- **failure modes are application dependent**
- **consistency is application dependent**
- **response to congestion is application dependent**

**Application Level Framing (Clark and Tennenhouse, Sigcomm 1990) provides a "design philosophy" that helps resolve these problems.**

# *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.**

# *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

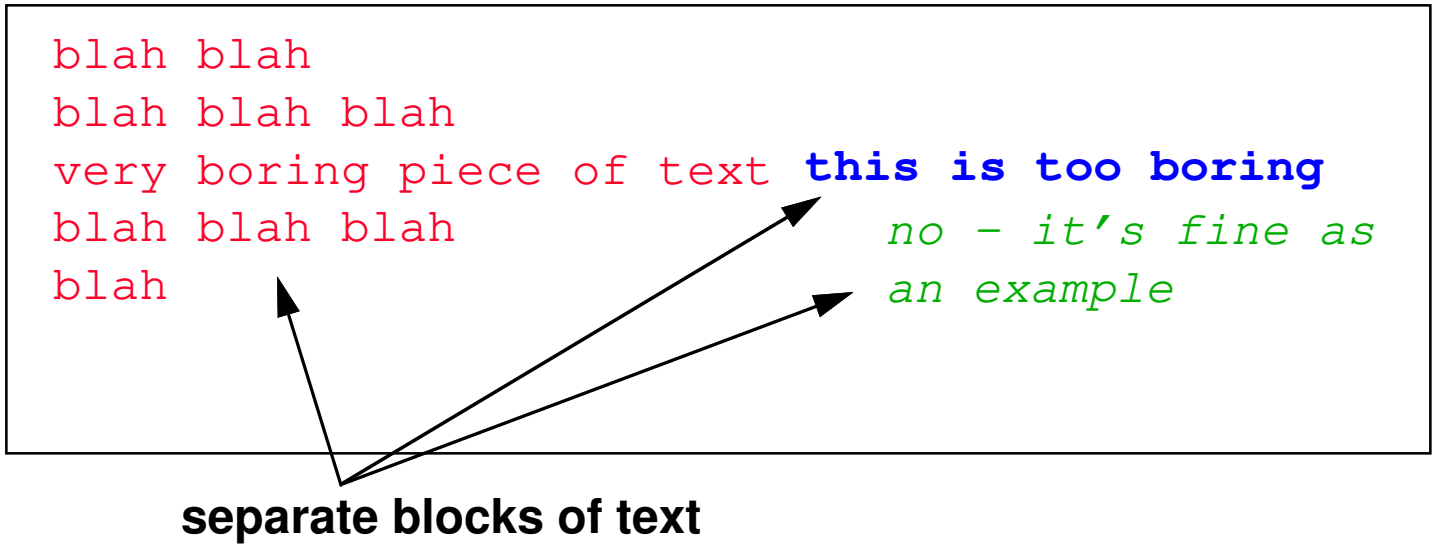
***The problem is that these requirements are contradictory given unreliable networks.***

# *Realistic User Requirements*

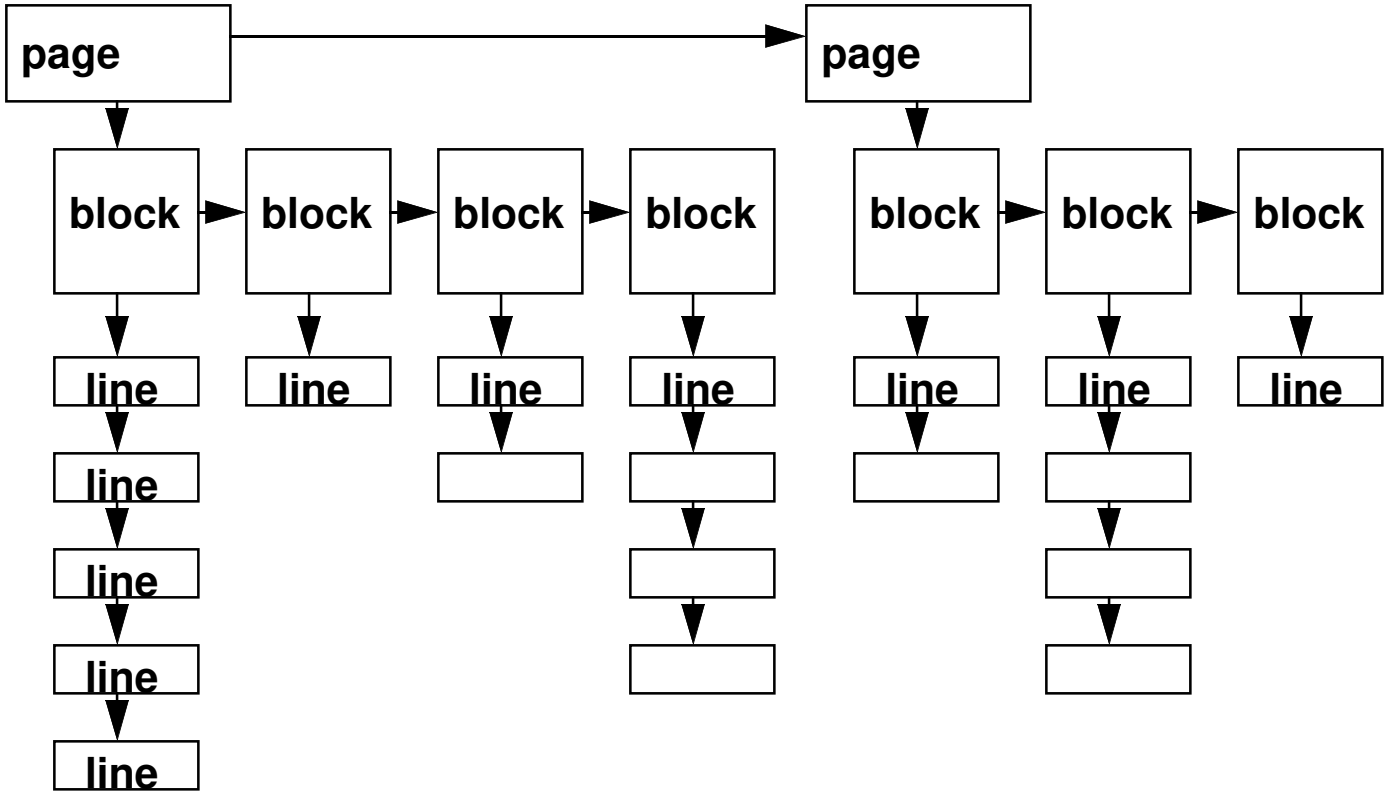
- **Many people should be able to manipulate the same data object over time.**
  - wb doesn't allow this, but we must do for a shared editor
- **Eventual consistency**
  - probably can't compromise on this
- **Fully interactive**
  - most systems compromise this in the presence of loss
- **Deterministic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

# *NTE 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.**

# *NTE 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?***

cannot now tell from missing changes

# *Redundancy and reliability*

**When 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 is Simple*

**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.**

***This needs 4 components to implement it:***

- clock synchronisation
- an announcement mechanism
- a nack-implosion avoidance mechanism
- rules to ensure that inconsistencies are always detectable and correctable.

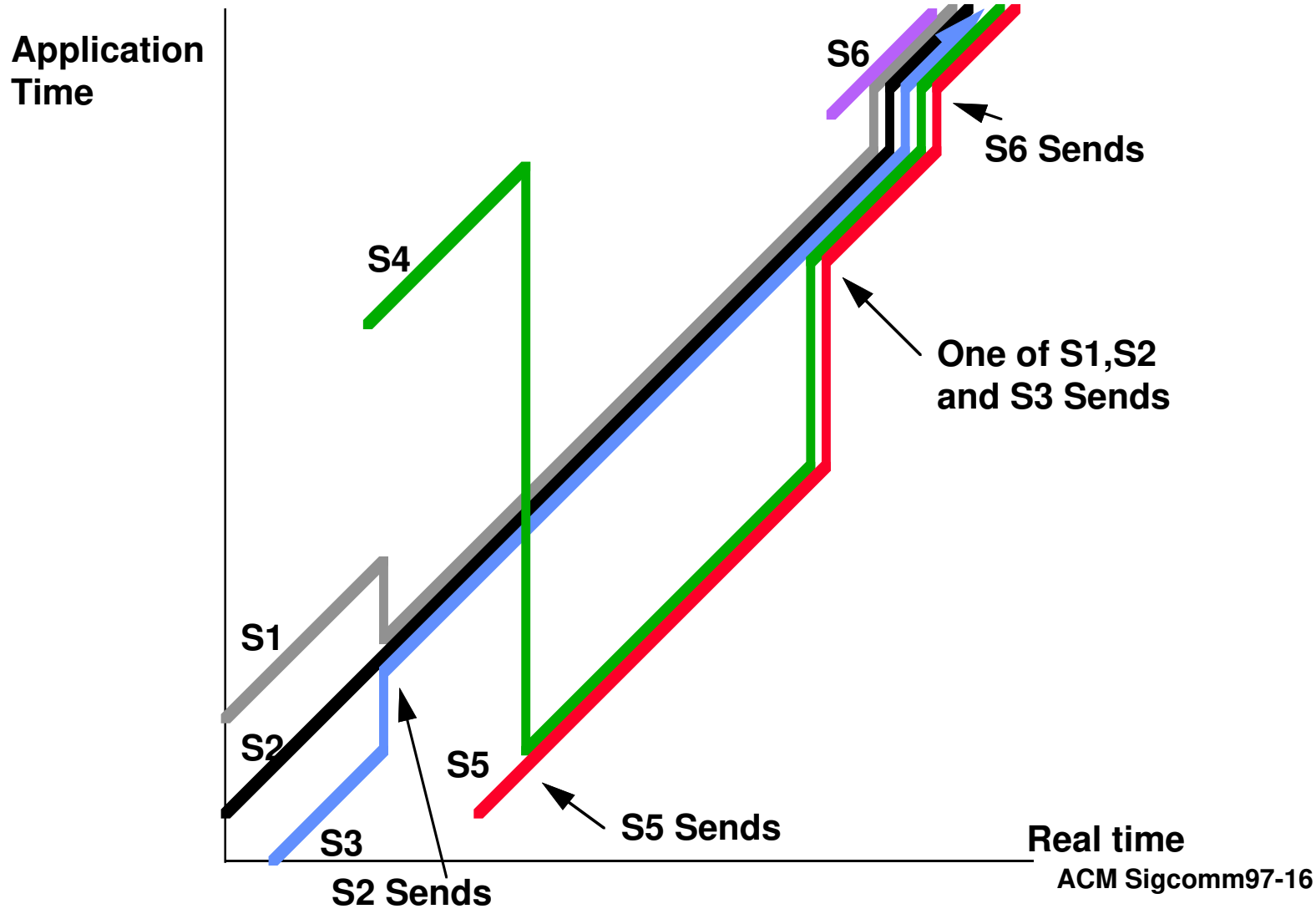
# *Clock Synchronisation*

- **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 Synchronisation*

- **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 synchronisation*



# *Announcement Mechanism*

**This is the mechanism by which sites discover that they have an out-of-date or missing object.**

- though missing objects are usually noticeable by references from other objects.

**There are two main situations where we need this:**

- packet loss on 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

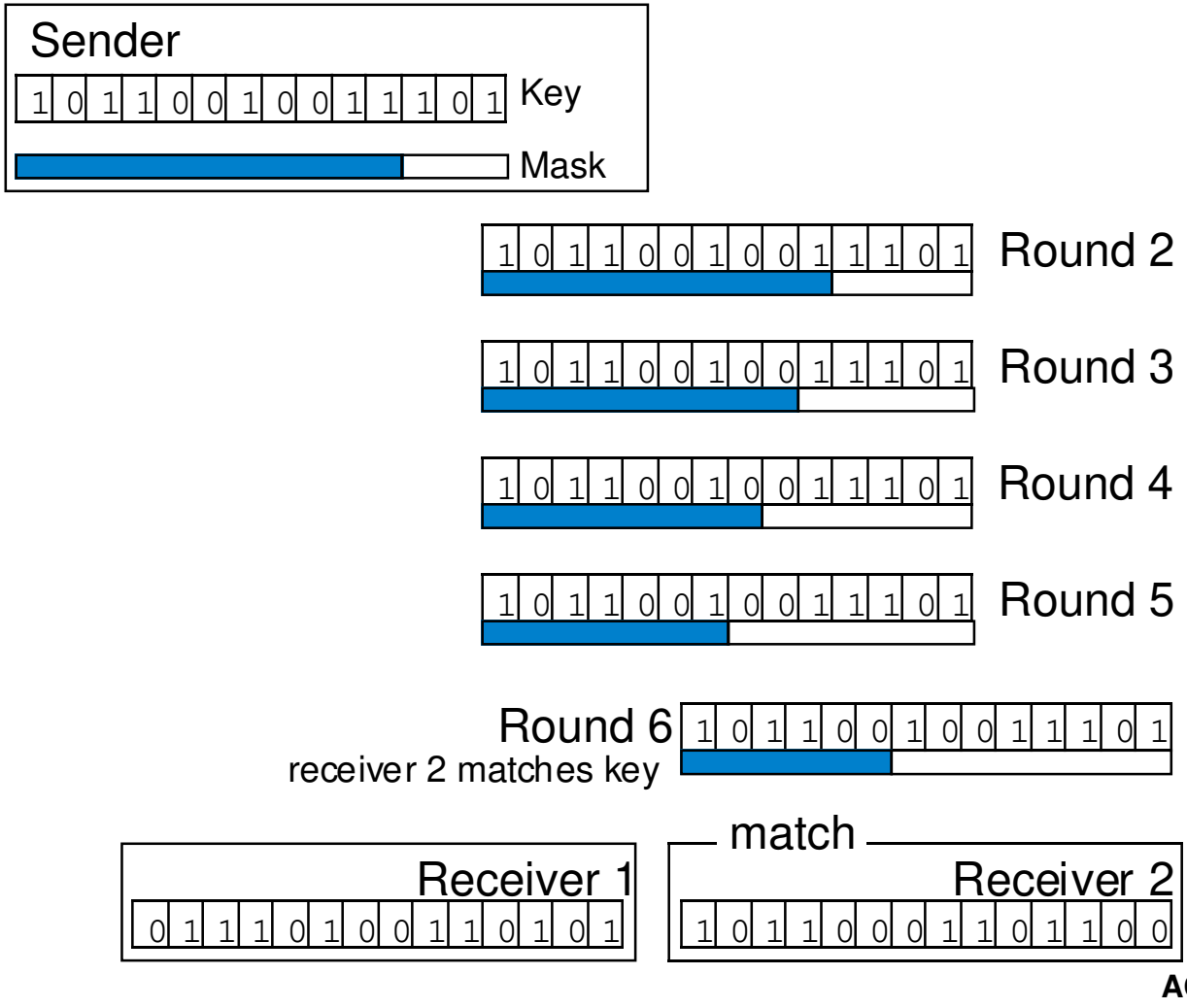
**We could "centralise" it**

- elect a site to list all the objects that recently changed
- it fitted well with the RTX scheme, so we centralised it.

# *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...

# *Constraints and Rules for Dataset Manipulation*

- **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*

- **NTE 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 complicates reliability!**

# *Conclusions 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.**