A Distributed Whiteboard for Network Conferencing

Steven McCanne
CS 268 Term Project

Computer Science Division
Electrical Engineering and Computer Science
University of California
Berkeley, CA 94720
mccanne@cs.berkeley.edu

May 25, 1992

Abstract

Advances in network technology and research in real-time packet-switched networks have prompted the emergence of internet conferencing. While audio and video conferencing tools are readily available, shared workspaces and drawing tools have been slower to emerge. We present a design for a network conferencing *whiteboard* that differs in notable respects from other shared workspace prototypes. We describe its transport layer, based on IP multicasting and application level framing, and its object oriented imaging model, based on persistent PostScript graphics operations. We then propose a user interface, and finally, describe a C++ implementation based on the InterViews structured graphics library.

1 Introduction

In the early days of the Internet, experiments with packet voice proved that computer networks could be used for integrated services [5]. However, the technological limitations of the time made such use impractical. This is now changing.

Dramatic advances in network bandwidth and computational resources have paved the way for a new generation of distributed applications. Indeed, computer workstation manufacturers are anticipating this trend by equipping machines with high speed network interfaces and audio and video hardware. These developments have facilitated the emergence of a new form of human productivity — computer supported collaborative work (CSCW). By integrating collaborative applications with real-time internetworking, virtual workspaces can be made to span the globe. Researchers worldwide will be able to collaborate without ever leaving their respective workplaces.

Because of its interactive nature, networked CSCW demands real time delay and bandwidth guarantees of the underlying networks. Accordingly, research is underway to develop protocols that can manage the network resources and provide performance guarantees [9, 3, 2]. At the same time, collaborative applications are being developed to exploit the new resources.

While audio and video applications are becoming widely available, another essential component of collaborative work, the network equivalent of a conference room "whiteboard", has been slower to evolve. Ironically, some network conferences are currently administered by faxing a speaker's slides to the conferences before commencing the conference over the electronic medium. In this paper, we present a design for a whiteboard tool that can overcome this poor man's approach to slide distribution. We envision an environment where documents, images, and arbitrary figures can be interactively displayed on the distributed whiteboard, allowing conference participants to interact by annotating the displayed material.

2 The Communication Model

In a distributed environment, we must make a fundamental design choice in partitioning the application. The two obvious approaches are a *centralized approach*, where a single copy of the application runs on a central site, and a *replicated approach*, where the application is replicated across all participating sites. Each approach has its advantages and disadvantages, and is influenced by the underlying network communication primitives. We consider both approaches with respect to unicast and multicast communication channels.

2.1 A Centralized Approach

In the centralized model, the application runs on only one host, the central site. The remote sites, or clients, redirect their I/O channels to the central site, which orchestrates the inbound and outbound event streams to effect the shared semantics.

The major advantage of this approach is its simplicity. An implementation is relatively straightforward since the problems of consistency and synchronization are defined away. Because there is only one copy of the application, consistency is not an issue, and because a total ordering on all events can be imposed by the central site, any synchronization primitives are easily implemented. Furthermore, the communications can all be carried out using traditional unicast channels.

An obvious disadvantage of the centralized approach is its demands on the network. The central site is a bottleneck. Since traffic from all participants must be processed by the central site, the bandwidth requirements degrade linearly with the number of participants. Additionally, communication latency can be dramatically increased, which is critically detrimental to interactive CSCW applications. For instance, even though the network delay between two nearby hosts may be quite small, they may have to communicate via a distant server over high latency link. Even worse, depending on the application, the overall communication rate may be limited by the slowest link in the network. Finally, if the central site fails, the application fails outright.

2.2 A Replicated Approach

If we are limited to unicast communication channels, then the centralized model may very well be a viable approach. However, a more efficient scheme can be devised using multicast channels. Multicast addressing allows a single datagram to be efficiently sent to a group of hosts. The network automatically forwards the packet along the appropriate links, making sure that only one copy is sent on any given link. In this fashion, a group of communicating hosts (usually) require only sublinear bandwidth requirements.

Most of the problems with the centralized model are not present in the replicated model. Here, the application is replicated on each remote site and there is no central authority. Each host communicates to all other hosts simultaneously via a multicast channel. Since there is no central site, there is no bottleneck. Also, communication latencies are minimized since they depend only on the intrinsic network delays. Furthermore, network failures and intermittent connectivity may be tolerated, since each site maintains complete state of the application.

But many of the advantages of the centralized approach become the disadvantages of the replicated approach. Because the application is replicated, inconsistencies can arise between sites. If the application cannot tolerate inconsistencies, either a priori or heuristically, a higher level protocol must be devised to guarantee consistency. Yavatkar [19] describes such a scheme using token passing.

A related problem is synchronization. Because multicast channels (usually) make no guarantees on packet ordering, messages can be received out of order. Even worse, causal relationships can be violated. Figure 1 illustrates this phenomenon. Host B sends a message, M1, that causes host A to reply with message M2. Because of network idiosyncracies, Host C may receive the response before the request.

These problems turn out to be present in the whiteboard and we will address solutions in subsequent sections.

2.3 IP Multicasting

The whiteboard's communication model is built upon the Internet Protocol (IP) Multicast network layer [7]. The Unix operating system provides access to this layer via the user datagram protocol (UDP). The standard *socket* interface [13] provides mechanisms for sending and receiving UDP packets. Thus, an application accesses a multicast channel by creating a datagram socket and binding it to a multicast address.

IP Multicasting has several attractive characteristics. First of all, IP is ubiquitous. The TCP/IP protocol suite dominates the current Internet and many believe this domination will continue even as networks scale to gigabit rates [12].

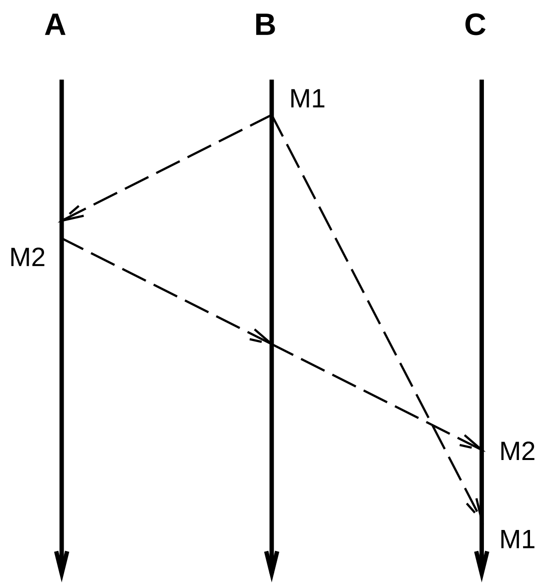
Second, the multicast extensions outlined in RFC 1112 [7] are readily available on a variety of systems. Even in heterogeneous environments, where multicasts are not supported at every gateway, IP multicasts can still be delivered using a technique called *tunneling*. A *tunnel* allows multicast routers to send packets across unicast networks using IP source routes. In essence, tunnels bridge islands of multicast networks over a sea of unicast networks.

Third, the application interface is simple, as simple as its unicast counterpart. Because the multicast channel semantics are encoded exclusively via the IP address[1], the operating system interface is identical in both the unicast and multicast cases[2].

Finally, because group membership is encoded in the multicast address, session management is simplified. An IP multicast address and UDP port number is the only information necessary to define a conference session. Groups are maintained transparently by the network, via the Internet Group Management Protocol (Appendix I of [7]). In contrast, a connection oriented approach handles group membership explicitly in the application, usually requiring each site to establish connections with all other sites before the session begins.

[1] Class D Internet addresses, whose high-order four bits are 1110, are reserved for multicast groups.

[2] Actually, the current kernel implementation requires special socket operations for group membership, specification of the time-to-live, and enabling of packet loopback. Group membership should be implicit in the address while the time-to-live and loopback flag should be stored in the route. This will be fixed in 4.4BSD.



This figure illustrates causality violations that may be introduced by the network layer. Time proceeds down the figure while the horizontal axis represents the spatial location of the three hosts, A, B, and C. B sends a message that causes A to reply, but C might receive the reply before the initial message. A transport layer may need to account for this behavior.

Figure 1: Causality violation in a multicast channel.

2.4 The Transport Layer

Since the UDP/IP multicast layer provides only *best effort* delivery semantics, the transport layer must implement reliable delivery. Furthermore, the network can introduce packet reordering and causality vioalations that may be problematic to the application. But not all applications need similar qualities of service. For instance, audio and video applications can tolerate certain amounts of data loss but require timely delivery. On the other hand, the whiteboard cannot tolerate packet loss, while communication delays, to a certain extent, can be tolerated. Thus, transport level policy decisions, at least for real-time applications, should be deferred to the application.

2.4.1 Application Level Framing

Indeed, Clark [4] argues that applications themselves should manage lost and misordered data. This approach is critical to the success of real-time, interactive applications, which must make progress in the presence of missing data. In the case of the whiteboard, most packets are idempotent and can be processed independently. Lost data can be repaired “in the background”.

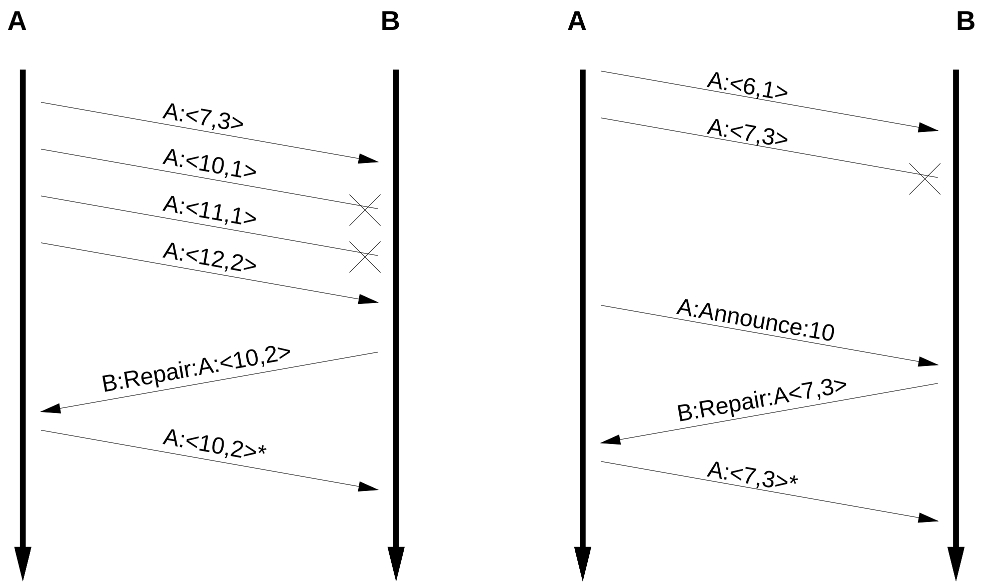
2.4.2 Reliable Delivery

Our design achieves reliable delivery with sequence numbering and periodic state announcement. Each packet is assigned a monotonically increasing sequence number. If a receiver notices a hole in the sequence space, it issues a repair request for the packets that are missing. Holes are noticed when a source sends data subsequent to a loss, or upon state announcement. Figure 2 illustrates these two modes of repair.

On lost data, a receiver could either unicast the repair request back to the sender or multicast it to the group. Since a loss for one receiver is likely a loss for many, a scheme that could minimize the number of acks generated by the group as a whole would be desirable. If the requests are multicast, receivers that need to issue an identical request can notice that it has already been issued. Additionally, since each site maintains complete state of the application, any site, not just the sender, is capable of answering a repair request (provided that site has the requested information). Ideally, the site closest to the requester should answer. For these reasons, we have adopted a multicast repair scheme.

2.4.3 Ack Implosion

The repair scheme mentioned here has a problem. Repair requests can pile up on top of each other, causing large network transients which errari90Danzig [6, Sec. 6.1.1] calls *implo-*



This figure illustrates the two repair modes for reliable delivery of packets. The diagrams represent two sessions with two hosts, A and B. Time proceeds down the page. $< s, n >$ represents n drawing operations beginning at sequence number s. In the lefthand session, A sends four packets, two of which are dropped. On receipt of the fourth packet, B detects missing data and requests a repair. A repairs the two missing operations with a single packet.
On the righthand side, the last in a sequence of packets from A is dropped. B does not detect the data loss until A sends its periodic announcement, at which time B issues the repair request.

Figure 2: Whiteboard Repair Mechanism

sions. To illustrate this phenomenon, first consider a PAR[3] scheme, where each data packet triggers acks from all the receivers simultaneously. All the acks are directed back at the sender, hence the term *implosion*. If the multicast group is large, the implosion can deterministically cause acks to be dropped (i.e., because of gateway queue overflows). The lost acks result in a retransmission, resulting in more lost acks, ad infinitum[4].

Now consider the negative acknowledgment repair scheme discussed in the previous section. While this approach eliminates ack implosions for error free communication, implosions can still result when all receivers discover a loss simultaneously. Furthermore, since any host may answer requests, the retransmissions themselves can result in implosions. To alleviate these problems, we propose a variation of Danzig's scheme, which uses randomization to smooth the transients. Our algorithm has two halves, one set of rules to generate repair requests, and another to honor replies. A request is generated as follows.

- On missing data D from host S, choose a random value τ, from a probability distribution given by some function $F_{\tau_S,N}$, parameterized by the size of the group, N, and the network delay from the local site to S, τ_S.
- Schedule a repair request packet, REQ_D, for transmission in τ seconds.
- If the data, D, or the repair reply, REP_D, is received before τ seconds, cancel REQ_D.

Since the request is multicast to the entire group, any site can answer it. The reply is generated as follows.

- On receipt of REQ_D, and if D is present, choose a random value τ, from distribution $F_{\tau_S,N}$.
- Schedule REP_D for transmission in τ seconds.
- If REP_D is received before τ seconds, cancel REP_D.

The choice of the distribution function $F_{\tau_S,N}$ is still under investigation. We conjecture that a simple uniform distribution on $[0, K\tau_S]$, where K is a small constant greater than one, will be adequate.

The net effect of the randomization and network delay term is to bias the retransmissions toward those sites that are the closest to the sender. In an Internet environment, where loss is often concentrated at the backbone gateways, this scheme is quite effective. Here, a packet drop will generally affect a large fraction of the group. Thus, the multicast repair mechanism will efficiently update everyone with a single packet.

[3]Postive Acknowledgment with Retransmission

[4]Actually, Danzig's scheme is smarter. It limits ack retransmissions by explicitly specifying the unacknowledged hosts in the data retransmission. Thus, progress is guaranteed.

However, other less well behaved scenarios are possible. Consider a network that is mostly reliable except for one link, with a leaf site S, at the far end of the congested link. In this case, S will issue many repair requests which the entire multicast group must process, wasting host processing time and network bandwidth. It would be better to unicast the repairs back to S, but that would compromise the more common scenario mentioned previously. An alternative approach is to limit the range of multicast replies using the time-to-live field in the IP header. Since the closest host usually honors the repair, the network traffic will be minimized. More study is needed to determine the real behavior of our repair scheme, and whether the suggested improvement is feasible.

2.4.4 Synchronization

Note that the repair scheme uses network delay estimates in its algorithm. A source usually estimates delay to a receiver by timing the interval between sending a packet and receiving the corresponding acknowledgment. Because our scheme uses mostly unidirectional communication (i.e., there are no acks), this method of delay estimation cannot be used. However, if the hosts can synchronize their clocks, then delay times can be computed very simply by timestamping each packet. Clocks can be synchronized via some protocol external to the application, NTP for example [15].

Since host clocks will be synchronized, temporal ordering can be recreated at any receiver. Furthermore, since network dynamics can introduce substantial variations in delay jitter [8], timestamps allow the receiver to reconstruct the sender timing relationships. Since (most) current networks do not have delay jitter-control policies[5], timestamps are critical to reproduce the sender's actions at the receiver.

2.4.5 Joining an Active Conference

When the whiteboard starts up, a site announces its presence with a special identification packet, allowing all participants to register the new member's presence. For instance, each host might update its display with the new member's session name. However, the joining party is not immediately notified of all active sites. Instead, it will quickly learn them, since each site periodically[6] transmits an announcement packet. The transport repair mechanisms will establish consistency between the global state and the new member's local state.

Work is currently underway to develop a general session management system[11]. The assignment of Internet addresses to sessions should be dynamic, requiring some network agent to distribute addresses for temporary sessions. Regularly scheduled sessions could be assigned permanent

[5][17] discusses the implications of delay jitter-controlled channels and argues that isochronous applications require only strict delay bounds.

[6]every few seconds

addresses, maintained in the domain name system [16]. In addition, methods for announcing conferences and notifying the target audience need to be developed.

3 The Imaging Model

We foresee two dominant usage modes of the whiteboard. In both cases, we assume the existence of some other form of interaction, for instance, audio and video. The whiteboard is not meant as a standalone tool.

In the first scenario, we envision a small collaborative setting, where people use the whiteboard as a scratchpad, annotating each others' diagrams. This is analogous to a small office meeting. The other scenario is an electronic talk or lecture. The speaker will have previously prepared a set of slides which are distributed, page by page, as the talk proceeds. The audience is free to annotate the diagram and interact with the speaker.

In either case, we adopt a *page model* of interaction. Namely, some site lays down a figure, causing the previous images to be covered. Participants then annotate the new figure, possibly deleting their annotations and creating more. Then, this cycle may repeat with a new page. This next page corresponds to the next slide in the electronic talk, or to the clearing of the screen by a member of the office meeting.

3.1 Persistent Graphics Streams

Each site in the conference produces a stream of graphics operations, which we call DrawingOps. DrawingOps may be grouped into aggregate objects, which can be referred to later via a RefOp, a type of DrawingOp. Control information is produced externally to the graphics stream.

Each site's graphics stream is completely independent of all other sites. For instance, one site cannot delete an object displayed by another, but it could cover it up with another opaque object. This approach greatly simplifies the problem of consistency. Since sites cannot affect each others' graphics streams, the transport repair mechanism can adequately establish consistency between the local application and the global state. However, there still remain consistency problems in correlating multiple graphics streams into sa composite image. This problem is addressed in Section 3.5.

A graphics stream is persistent and arbitrarily long. Thus, we have the ability to *scroll* backward through a session while it is in progress. We could go back to an earlier page, add additional annotation, and bring it forward. This process could be made quite efficient, both computationally and in terms of network bandwidth, by using RefOps to refer to the earlier graphics.

This history mechanism is flexible. The user can specify no history (i.e., just remember the most recent page) or an infinite history. In the latter case, the conference history could persist even across sessions. For example, we might want to scroll back to some slides that were presented in last week's, or even last year's, meeting.

3.2 Local Store vs. Network Bandwidth

The history policy represents a tradeoff in local storage versus network bandwidth. Since RefOps can refer arbitrarily far into the past, network repairs may be necessary to update sites that have discarded the corresponding DrawingOps. If a site maintains an infinite history, it will never need to issue history repairs. On the other hand, if a site maintains only its own history, it must use the transport repair mechanisms to retrieve references to discarded DrawingOps. At one extreme, no network bandwidth is used for history repairs but local storage is required for the histories. At the other extreme, local storage is minimized but global persistence then requires network bandwidth for repairs.

The history mechanism must be flexible. We conjecture that infinite (or very long) histories, spooled to the local file system, will be quite useful. In conjunction with saving other aspects of the conference, such a history allows one to build a library of talks and sessions which could be replayed at any future time.

3.3 No Telepointing

A popular notion in a shared workspace is the mouse based *pointer*, allowing sites to point at items on the screen using an animated icon. We have left this out of our design, in favor of an annotative model. Items are "pointed to" by annotating with appropriate marks, for example, arrows or circles. The annotations are easily erased.

We believe there is a current prejudice toward the pointer approach, instilled by the prevalence of present mouse based window systems. But mouse based drawing is clumsy and inefficient for whiteboard interaction. Instead, we look to pen based computing as a more viable means of graphical expression. Here, the natural mode of pointing will not be an animated pointing device, but will instead be annotations and graphical gestures [18].

3.4 DrawingOps

All DrawingOps are timestamped and assigned sequence numbers, relative to the sender. Timestamps allow the receiver to play out the graphics stream with the same time structure of its sender. Additionally, timestamps aid in the graphics rendering process, since more recent DrawingOps should appear on top of older ones. Sequence numbers are needed by the transport layer.

Except for RefOps and grouping commands, all DrawingOps are idempotent. This means that the graphics update can usually be rendered immediately upon receipt of the packet, a critical consideration for an interactive application.

For ease of portability, the graphics operations must be device independent. Since computer environments are inherently heterogeneous, the whiteboard will only be successful if it can function across a range of hardware configurations. The de facto standard in device independent page description is the PostScript language [1], which we adopted as the representation model for DrawingOps. Indeed, almost all modern typesetting or graphics systems are capable of generating PostScript output. Thus, the whiteboard will immediately leverage off a vast array of existing tools.

To simplify and make efficient the most common graphics operations, a set of predefined PostScript functions are implicitly defined by the whiteboard. In other words, the PostScript rendering is carried out with respect to an agreed upon code preamble. The predefined commands can be short-circuited directly to the display, while arbitrary PostScript is processed via a fully general interpreter. Finally, a standard binary encoding format [1] may be used to compress the commands and eliminate lexical analysis.

3.5 The Rendering Method

While we have defined a method for rendering the individual graphics streams from each site, we have not described how they can be combined into a single composite display. Initially, we chose what we thought was a very simple approach. Since each host is autonomous, we can create a composite by merely stacking the layers in some agreed upon order. This *layering model* approach is similar to the tack taken by the Teamworkstation project designers [10].

An initial problem was devising a protocol for reaching a consistent global stacking arrangement. While we finally devised a scheme that seemed to work well, a new problem became apparent. The layering model is not a close match of reality. When a site made annotations to a page, that layer would be raised to the top so that the new annotations would, as expected, appear on top. As a side effect, old annotations from the site would as well be raised, which is obviously not natural. Furthermore, the site that initiated the drawing would appear underneath all the other layers. Thus, any annotations made by the original site would appear underneath all others.

The problem is simply that the layering model does not capture the temporal aspect of the annotations. Newer marks should always appear on top of older ones. To account for this, we devised a new rendering model. All DrawingOps are rendered into a single layer in the order dictated by their timestamps. Again, we rely on loosely synchronized clocks to achieve meaningful correlation between independent graphics streams.

Of course, if two people draw at the same time slight inconsistencies may arise. For example, consider two sites A and B which annotate the same drawing. If A draws before B, but B starts drawing before receiving updates from A, B will see A's marks drawn underneath its own, even though they appear after its own. This is, at worst, a minor annoyance that is easily tolerated.

3.6 Floor Control

A more problematic consistency problem entails page boundaries. If two or more sites simultaneously initiate a new page, the conference must choose only one, in a distributed, consistent manner. More generally, we must establish a means of *floor control*.

Yavatkar describes several token-based concurrency approaches in [19]. Two of his approaches, called *brain storming* and *chalkboard interaction*, are applicable to the whiteboard. Brain storming is analogous to the office meeting scenario. In this case, all participants may interact at any time. Similarly, *chalkboard interaction* is analogous to the electronic talk. Here, the speaker holds the floor, preventing others from interacting. Then, a question and answer period ensues where participants may interact as dictated by the speaker. The process then repeats.

However, we believe that, in many cases, these floor control primitives should not be built into the application. Rather, human behavioral protocols will solve the problem on their own. A (seemingly) undisputed example is audio conferencing. If two people begin to speak simultaneously, then human instinct will result in quick agreement as to who should speak. Similarly, with the whiteboard, if two people initiate new pages simultaneously, they will both notice this collision. Resolution can be reached by human negotiation over other media channels.

Even in the *chalkboard interaction* example, floor control should be managed at the human level. An environment in which participants may speak out of turn, though it permits rude behavior, is more realistic than one in which members are mute until artificially allowed to interact.

3.7 Distributed Consistency

Because we have not adopted a stringent floor control mechanism, one subtle consistency problem has chance to arise. Annotations can sometimes be applied to the wrong page. If a site is annotating while another site begins a new page, some of the annotations that were meant for the old page will appear on the new one. Instead of implementing an explicit consistency protocol, we take advantage of the specifics of this application, and propose a simple fix — each DrawingOp will explicitly contain the page identifier to which it applies. A unique page identifier can be constructed by combining the

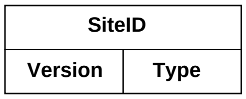


Figure 3: Transport Level Header

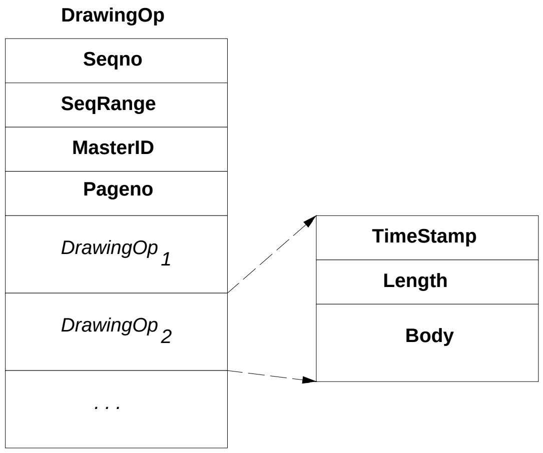


Figure 4: Format of DrawingOp Packets.

site identifier that initiated the page with its current sequence number at the time of the initiation. Thus, a receiver can prevent annotations from being displayed on the wrong figure.

Distributed consistency entails a heavy tradeoff between precision and efficiency. The more precisely consistency is maintained among sites, the costlier it is. By avoiding data dependencies in the graphics streams and loosening the constraints on temporal synchronization, we have avoided the necessity of a token based approach, which has clean semantics but poor performance implications.

3.8 Packet Layout

A transport level header is prepended to each packet, as shown in Figure 3. The SiteID identifies the sending participant, while the Type field encodes the packet type, and is one of DrawingOp, Announcement, RepairRequest, or RepairReply. The Version field is a simple version number to prevent mismatches between protocol generations as they evolve.

The DrawingOp format is shown in Figure 4. A packet may contain multiple DrawingOps but they must be contiguous in the sender's sequence space. The Seqno and SeqRange fields indicate the range. MasterID and Pageno reference the abstract page to which the DrawingOps apply. As explained in Section 3.7, these fields convey information to maintain consistency between annotations and page boundaries.

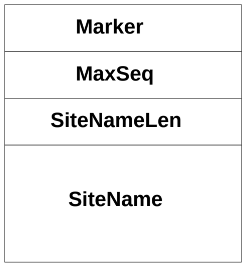


Figure 5: Layout of announcement packets.

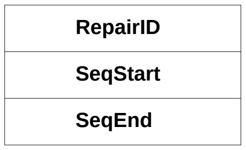


Figure 6: Layout of repair packets.

An individual DrawingOp is a timestamped PostScript code fragment. The PostScript is encoded in a binary format in Body and is of the length indicated by Length.

The Announcement packet, shown in Figure 5, contains the sequence number of the current page, Marker, the maximum sequence number generated thus far, MaxSeq, and a counted string that identifies the participant, SiteNameLen and SiteName.

Figure 6 shows the format of repair requests. RepairID indicates the site whose graphics stream should be repaired, while SeqStart and SeqEnd indicates the range of DrawingOps that are missing.

The format for RepairReply's is not shown. It is identical to the DrawingOp format, with the exception of an extra field to indicate the original source of the DrawingOps.

4 User Interface

The user interface consists of four main pieces:

- the whiteboard canvas,
- the palette,
- the history interface, and
- the session manager.

The canvas is simply a large area of the screen dedicated to the display of the whiteboard's contents. Marks are made on this canvas via stylus, mouse, or keyboard input. The palette contains various tools for creating specific instances of objects, for example, boxes, arrows, lines, freehand drawing, etc. Patterns, brush widths, fonts, and all the standard features of a drawing editor are included. The palette also allows arbitrary PostScript files, for instance, a set of slides, to be introduced a page at a time.

The graphical interface is object oriented. Objects can be created, manipulated, and deleted. Groups of objects may be selected and scaled, moved, or deleted as a unit.

The history interface lets the user manipulate time. A time scroller lets the user scroll back through previous drawings, either a page at a time or an operation at a time. Additionally, the user can position the scroller at an earlier time, and let the session "replay" itself. Any frame from the past can be cloned and incorporated into the current session point, or can be worked on off-line and later integrated back into the session.

The session manager is the visual interface to the rest of the conference. The names or Internet addresses of the conference participants are displayed in a box. A site's name is highlighted whenever it performs a drawing operation, allowing easy identification of the responsible party. A check box next to the site's name can be selected to temporarily disable the displaying of graphics from that site. The whiteboard still records all operations, so resetting the check box will cause the hidden annotations to be displayed.

Figure 7 shows the graphical interface for the prototype described in the next section. This contrived example illustrates the annotation model, where a diagram is presented, in this case a network topology, and then annotations are made by the participants.

5 A Prototype

A prototype of the whiteboard was implemented. Much of the design has been completed, but more work is needed.

The current implementation consists of about 6000 lines of C++ code. It is based on the outdated "Graphic" library from InterViews-2.6 [14], Mark Linton's object oriented graphics system. We had hoped to just extend the existing InterViews graphical editor, *idraw*, with our networking ideas. However, this proved impossible, as the organization of *idraw* would not easily map to our network model.

5.1 Event Dispatching

A weakness of InterViews-2.6 applications is their approach to event dispatching. Rather than dispatch events from a single rooted point, objects (like alert messages or menu items) will often drop into their own event reading loops. The effect is to starve other event handlers until the executing one finishes. This is sometimes tolerable. For instance, in *idraw*, while a user manipulates a menu item, nothing else interesting can happen. Hence, the menu item can be implemented with its own event loop. On the other hand, this behavior in the whiteboard is unacceptable. While the user manipulates a menu, drawing actions may arrive from the network at any time and should appear immediately on the display.

```
Object::handle(event e)
{
        if (e.type == Down) {
                do {
                        e = read();
                        if (e.type == Motion)
                                track(e);
                } while (e.type != Up)
        }
}
```

Figure 8: Starvation Prone Event Dispatch

```
Object::handle(event e)
{
        if (down) {
                if (e.type == Up)
                        down = false;
                else if (e.type == Motion)
                        track(e);
        } else if (e.type == Down)
                down = true;
}
```

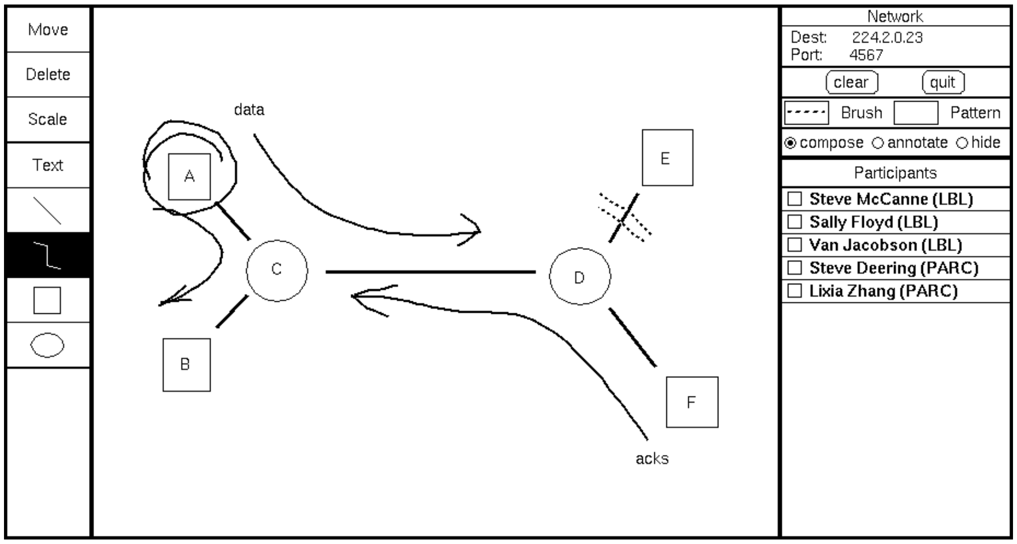
Figure 9: Non-Starving Event Dispatch

A solution might be to use asynchronous notification, which would allow event loops to be preempted and, for instance, the whiteboard display to be updated. However, InterViews is not reentrant and requires synchronous operation. Thus, event handlers must never enter into their own event loops. Instead, they must keep enough internal state to be able to reconstruct the desired event sequence.

Figures 8 and 9 illustrate with code the two approaches to event handling. The function *handle()* calls another function *track()* with mouse motion events, but only while the mouse button is held down. The approach of Figure 8 uses flow control, namely the event-reading while-loop, to "remember" that the button is down. But this starves other event handlers. The starvation is circumvented by the code in Figure 9, which uses a state variable, *down*, to remember the button position.

5.2 A Code Overview

InterViews allows an application to "listen" on arbitrary I/O channels, thereby providing a convenient hook for the packet



Dest: 224.2.0.23
Port: 4567

[✓ compose] annotate hide

Steve McCanne (LBL)
Sally Floyd (LBL)
Van Jacobson (LBL)
Steve Deering (PARC)
Lixia Zhang (PARC)

Figure 7: The Prototype

handling mechanism. The arrival of a packet triggers an event which is delivered to the correponding listener. In the object-oriented spirit, we created a "network object", whose "Handle" member function is called by InterViews when packets arrive on its socket descriptor. The network object reads the packets from the device, then passes them to the *Demuxer* object.

The *Demuxer* dispatches the packet to the appropriate *Receiver* object, based on its site identifier. The *Receiver* object implements all of the transport layer mechanisms discussed earlier. Additionally, it maps packets to *Graphic* objects, which are inserted into the master *View* object.

The *View* object is responsible for updating the whiteboard display. Double buffering is employed to minimize screen flicker. Because this is a costly operation, we restrict the updates to the localized areas where "damage" occurs. This approach proved quite effective.

As the user manipulates the application locally, a *Sender* object maps actions into *Graphic* objects. The *Graphic* objects are then looped back to the local *Receiver*. At the same time, a *Graphic* object maps itself to a packet, which is sent to the *Network* object and ultimately to the network itself.

Because the *Sender* loops back *Graphics* to the local *Receiver* much of the implementation is simplified. There are no special cases for the local host. The graphics handling and transport repair mechanisms are uniform for the local and remote sites.

Figure 10 gives a simplified view of the object decomposition described here. The nodes represent instances of objects, while the arrows represent data flow and dependencies of packets and graphics.

6 Future Work

Several aspects of the design presented in this paper have not yet been implemented in the prototype. The ack implosion avoidance algorithm is not yet in place. Repair requests are always issued as soon as possible, and the original sender (and only the sender) honors them. The retransmission algorithms require further study, on both experimental and theoretical levels.

The PostScript imaging machinery still needs to be incorporated. Currently, objects are encoded in a temporary format that was thrown together to make things work. We plan to use the GNU PostScript interpreter, Ghostscript.

The new rendering model has not been implemented. The prototype currently renders the composite image by stacking all site layers on top of each other. The transition to the temporal model should not be difficult.

The prototype should use the most recent version of InterViews, as major improvements have been made since version 2.6. The whiteboard will be ported to this new environment before it enters production use.

Finally, drawing with a mouse is clumsy. The utility of the whiteboard should be greatly enhanced by replacing the mouse with a pen.

7 Summary

We have presented a design for a collaborative whiteboard in the form of a communication protocol, an imaging abstraction, and a user interface. The communication protocol employs IP multicast and application level framing to efficiently

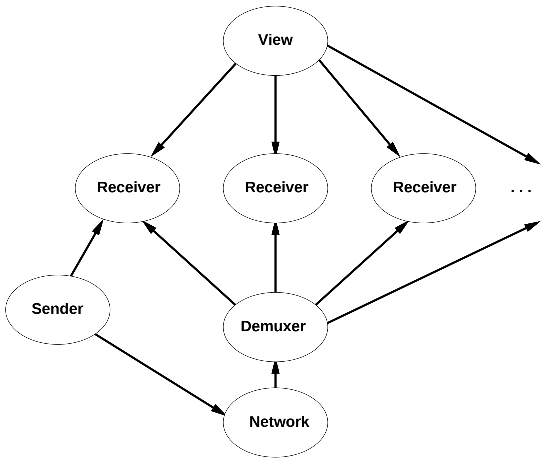


Figure 10: Whiteboard Object Hierarchy

utilize the underlying network. Initial feedback from the prototype indicates that the transport mechanism is robust. The imaging abstraction evolved into the temporal rendering approach that uses loosely synchronized clocks to effect global consistency. The temporal model is simple, intuitive, and practical. Finally, the user interface ties everything together in an easy to use package.

As network conferencing becomes commonplace, and personal pen and pad based computers become available, we believe that the electronic whiteboard will prove to be an indispensable collaborative tool.

8 Acknowledgments

Van Jacobson suggested this project and contributed many ideas to its design. He provided valuable advice and helped steer the design through its development. Sally Floyd helped me understand the probabilistic foundation of the ack implosion avoidance algorithm. Finally, Vern Paxson provided valuable comments and corrections on a draft of this paper.

References

[1] ADOBE SYSTEMS INCORPORATED. *PostScript Language Reference Manual*, second ed. Addison-Wesley, Dec. 1990.

[2] CASNER, S., LYNN, J., PARK, P., SCHRODER, K., AND TOPOLCIC, C. *Experimental Internet Stream Protocol, version 2 (ST-II)*. ARPANET Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, Oct. 1990. RFC-1190.

[3] CLARK, D. D., AND JACOBSON, V. Resource control in datagram networks. DARPA PI Meeting Viewgraphs, Mar. 1991.

[4] CLARK, D. D., AND TENNENHOUSE, D. L. Architectural considerations for a new generation of protocols. In *Proceedings of SIGCOMM '90* (Philadelphia, PA, Sept. 1990), ACM.

[5] COHEN, D. On packet speech communication. In *Proceedings of the Fifth International Conference on Computer Communications* (Atlanta, Georgia, Oct. 1980), IEEE, pp. 271–274.

[6] DANZIG, P. B. *Optimally Selecting the Parameters of Adaptive Backoff Algorithms for Computer Networks and Multiprocessors*. PhD thesis, University of California, Berkeley, Dec. 1989.

[7] DEERING, S. *Host Extensions for IP Multicasting.* ARPANET Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, Aug. 1989. RFC-1112.

[8] FERRARI, D. Distributed delay jitter control in packet-switching networks. Tech. Rep. TR-90-056, International Computer Science Institute, Berkeley, CA, Oct. 1990.

[9] FERRARI, D., AND VERMA, D. A scheme for real-time communication services in wide-area networks. *IEEE Journal on Selected Areas in Communications 8*, 3 (Apr. 1990), 368–379.

[10] ISHII, H., AND MIYAKE, N. Toward an open shared workspace: Computer and video fusion approach of Teamworkstation. *Communications of the ACM 34*, 12 (Dec. 1991), 37–50.

[11] JACOBSON, V. Private communication, May 1992.

[12] JACOBSON, V., BRADEN, R., AND ZHANG, L. *TCP Extension for High-Speed Paths*. ARPANET Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, Oct. 1990. RFC-1185.

[13] LEFFLER, S. J., MCKUSICK, M. K., KARELS, M. J., AND QUARTERMAN, J. S. *The Deisgn and Implementation of the 4.3BSD UNIX Operating System.* Addison-Wesley, 1989.

[14] LINTON, M., CALDER, P. R., AND VLISSIDES, J. M. InterViews: A C++ graphical interface toolkit. Tech. Rep. CSL-TR-88-358, Stanford University, Palo Alto, CA, July 1988.

[15] MILLS, D. L. *Network Time Protocol (version 2) specification and implementation*. ARPANET Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, Sept. 1989. RFC-1119.

[16] MOCKAPETRIS, P. *Domain names - implementation and specification*. ARPANET Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, Nov. 1987. RFC-1035.

[17] PARTRIDGE, C. *Isochronous Applications Do Not Require Jitter-Controlled Networks*. ARPANET Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, Sept. 1991. RFC-1257.

[18] RUBINE, D. Integrating gesture recognition and direct manipulation. In *Proceedings of the 1991 Summer USENIX Technical Conference* (Nashville, TN, June 1991), pp. 281–298.

[19] YAVATKAR, R. Issues of coordination and temporal synchronization in multimedia communication. In *Proceedings of the Fourth IEEE ComSoc International Workshop on Multimedia Communications* (Monterey, CA, Apr. 1992).