Scalable Shared Window System for Large Groups Based on Multicast

Lassaâd Gannoun and Jacques Labetoulle
Institut Eurécom
2229, route des crêtes, BP. 193
06904 Sophia Antipolis, France
Voice: +33 4 93 00 26 71
Fax:+33 4 93 00 26 27
E-mail:{gannoun,labetoul}@eurecom.fr

Abstract

IP Multicast, Lightweight sessions and Application Level Framing are principles that guide the design of multimedia conferencing applications, but they do not provide specific solutions. In this paper, we use these design principles to guide the design of a shared window system over IP Multicast. In contrast to previously implemented systems, our shared window system addresses issues related to a very large group size of participants and dynamic leaving and joining of a high number of participants. Our shared window system offers a join session protocol that scales to a large number of latecomers and also provides a control floor policy adapted to a large group size. A first version of the system implementation justifies the viability of the design decisions followed.

1. Introduction

Today, high speed networks enable people throughout the world to interact with each other without the need of travelling. They may attend virtual meetings without even leaving their offices, through the provision of facilities enabling them to cooperate via joint construction of a common information space [24]. One fundamental facility is a shared visual space, which allows the members of a cooperating ensemble, each on his/her workstation, to simultaneously share and revise information.

Shared window systems are one solution to providing a shared visual space. They exploit properties of the base window system to allow joint usage of unmodified applications [13]. Such an approach has several advantages. In fact, users are not required to use new applications, they can share applications already in place. Furthermore, the system does not need to be modified to support new applications. Finally, the task of developing a cooperative ensemble is greatly reduced.

However, existing shared window systems [13, 2, 18, 19, 3], use unicast channels (TCP/IP) to distribute X protocol data streams to all participants of the conference session. This is not optimal, since the source that multiplexes X protocol data streams will be heavily loaded when the group size of participants becomes very large (e.g. up to 10^3). In fact, the sender will spend all of time in switching from a connection to another one, which results in a bottleneck at the sender, consequently the shared window session will crash.

Furthermore, the Internet architecture provides a more efficient approach of multicasting the information to all participants [5]. Multicast transmissions save the network bandwidth compared to unicast transmissions. Moreover, with multicast transmissions the network bandwidth is reduced compared to unicast transmissions. That's why most multiparty conferencing systems over the Internet are being implemented over the IP Multicast [5] service model which is provided by the MBone [15].

Due to the limitations of traditional application sharing systems, we have designed a new approach that is based on multicast delivery of X protocol data streams (i.e. requests). Implementing a system that allows X applications to be shared, however, is very complex due to some decisions taken when the X protocol was designed [22, 26]. This degree of complexity becomes more critical when we design a shared window system in a Multicast environment. Taking in account this new environment, a maximum of the system functionality should be handled by the end-hosts. Indeed, we can minimize the feedback between receivers and the sender, and hence we can avoid the problem of "*feedback implosion*" in the case of multicast transmissions for large groups.

Thus, a new shared window system is designed for high requirements such as scalability to very large number of participants, TCP-like congestion control and high efficiency. This system is intended to be used in a distance learning environment for large groups.

This paper presents a fully distributed system for sharing applications running under the X window system. In contrast to previously implemented systems, our shared window system addresses issues that are related to a very large group size of participants and dynamic leaving and joining of a high number of participants (e.g. step join). Using the IP multicast [5] service model, we have adopted some design principles such as **Lightweight Sessions** (LWS) and **Application Level Framing** (ALF) [27] to guide the design of our shared window system based on the IP Multicast service model.

These principles are the background for distributed applications, that are developed on the MBone [15], in order to guide their design, so that these applications can co-exist with others developed on the MBone.

The remainder of this paper is organized as follows: Section 2 exposes the design principles that are the guidelines for the design of the system. The related work and previous implemented shared window systems and their limited performance are presented. In section 4 we explore the design process of the shared window system. The system architecture is exposed in section 5. After, we describe a floor control policy adapted with a large scale environment. Section 6 and 7 explore the implementation process and the performance and the limitations of a shared window system on a multicast environment. Finally we provide a conclusion and outline our future work.

2. Design Principles

IP Multicast service model provides a simple "send a packet" interface by which a group of senders and receivers can exchange data without the senders needing to know who the receivers are or the receivers needing to know who the senders are. In addition this service model can lead to scalable and robust applications because the members of the group do not need to come with any agreement about who is actually in the group [14].

Because IP Multicast service model provides only a simple "send a packet" primitive by which a group of senders and receivers can exchange data, richer application semantics must be built from this simple primitive. This has led to a more structured multimedia conferencing architecture which is the **"Lightweight Sessions" (LWS)** model that underlies the MBone tools. In this framework, a session is a collection of end-hosts that communicate using a particular set of IP Multicast group addresses.

In a LWS, group management is not explicit. Unlike many conference management schemes, there is no protocol for controlling access to the group or for maintaining a consistent group view of each receiver. There is no explicit group membership mechanism other than joining the multicast groups and sending periodic session messages to indicate membership.

The concept of **Application Level Framing** [27] emerged due to the fact that applications did not want a single transport layer to perform everything for the application. The ALF design principle suggests that data should be transmitted on the network in units which are idempotent and hence, can be utilized by the application independently of other application data units. This is exactly what is required by some multicast based shared applications, so that data can be presented to the user as soon as it is available. Permitting this heterogeneity is essential for applications to be scalable [14].

As a result, the application can handle reliability and consistency issues as it sees appropriate depending on the application context. Under ALF, multimedia applications are simplified and both application and network performance are enhanced by reflecting applications semantics in the network protocol. And ALF-based, media specific tools are a natural way to implement a simple solution to multimedia's complex problem---high rate, high volume, continuous media data streams [16] as well as discrete media streams.

Since the tools are directly involved in processing the media data flows, ALF can be used to tune all the performance-critical media data paths within the application and across the network.

3. Related Work

Much research is carried out over the past years to design and develop shared window systems for the X Window System [13, 2, 18, 19, 1]. The X window system has a client/server model in which the client communicates with the server using the X protocol.

From the user's point of view, the main differences between these systems lies in the users interface and the applicable policies. These differences, however, are more deep and are related to different design choices and architectural topology and features. We examine in this section the different classes of these systems and their main technical approaches. We distinguish three main classes.

The first class consists of shared window system implemented as a set of modifications to the standard X library, Xlib, yielding to a new library shXlib. The SharedX [1] is an example of the first class. In a shared window system of this class (figure 1) we have an X client which simply talks to several X servers. This architecture is referred as to be the V topology architecture [3].

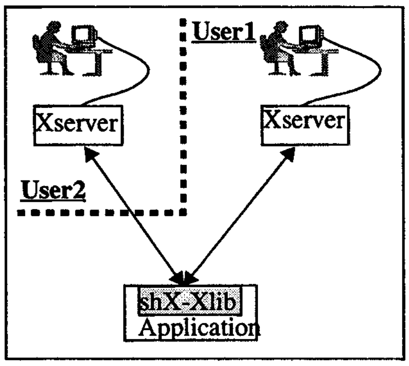

FIGURE 1. The shXlib based architecture

This architecture is done to avoid having to modify the client itself. Therefore it is linked to a new X library shXlib which is responsible for talking to remote X servers. The fact that it requires either object code for the application or at least dynamically linked library a such shared window system has a centralized architecture. This system can not scale to a large number of X servers, since the "augmented" client manages all connections and processes all the data to and from remote X servers.

Having to link a new library (Xlib) into the applications is a difficulty that the second class is designed to overcome. Most current window sharing systems let the client talk to a proxy server (pseudo server) that pretends to be an X server and hence, the latter multiplexes the protocol to the real X servers (figure 2). This architecture is referred to be the Y topology architecture Borm94.

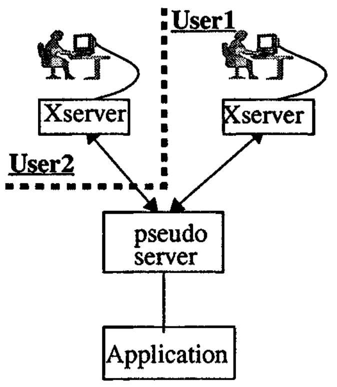

FIGURE 2. The pseudo server based architecture

One of the most prominent example for this pseudo server based architecture is the XTV system [2]. In contrast to the shXlib approach the pseudo server can be run on a different system than the client being shared, separating the communication work load from the application load. Despite these advantages, the pseudo-server architecture as well as the shXlib based architecture can not handle a large number of users because the pseudo server will be heavily loaded when managing a very high number of connections to remote X servers.

It's necessary to note, that both the shXlib and the pseudo-server based architecture have some problems with accommodating latecomers in a conference long after the X client has set up all its necessary resources in the X servers [11, 4]. In fact, only the X servers have complete knowledge about all the resources created by the application, and then it's advantageous to give one of the participant's X servers the responsibility for multiplexing X protocol data streams to other servers (figure 3).

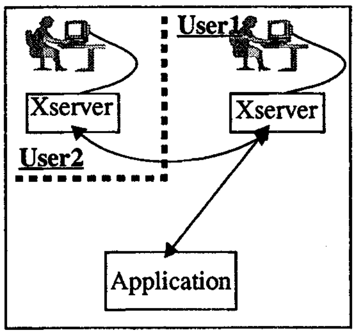

FIGURE 3. SharedX based architecture

This approach is related to the third class of shared window systems. The prominent example here is HP's sharedX product. Unfortunately, as X servers are hardware dependent products, this approach is unlikely to ever be universally available [3]. This class of architecture can not scale to a large number of users, because a special X server is responsible for multiplexing all the data to other X servers. When the number of managed servers becomes very high, the X server processing data sent to and coming from remote servers will be heavily loaded, consequently the performance of the shared window session will be highly degraded.

However, we note that the Xwedge framework [13] can be distinguished from all these classes by its distributed topology in which the sharing functionality are distributed among all sites that are involved in a shared window session (figure 4).

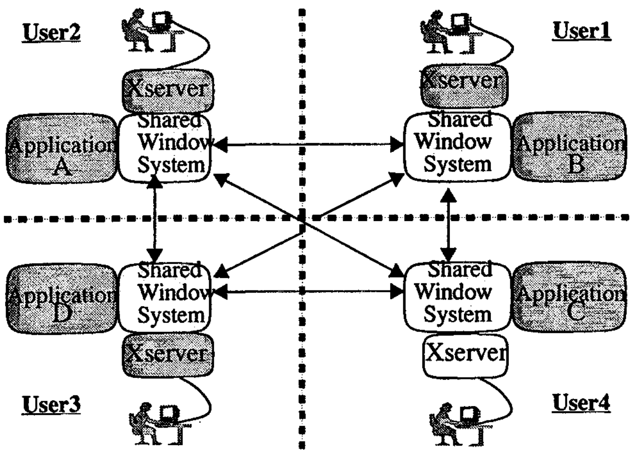

FIGURE 4. Distributed Architecture for the Shared Window System

In this architecture we have no central server like the pseudo-server based architecture, but we have a shared window system agent in the side of the shared application and another one on the side of the remote X servers.

This concept of independent cooperating entities is more promising for further development of open systems for cooperative work. That's why we have adopted a distributed architecture to design our shared window system over IP Multicast.

However, all of the studied architecture classes for the shared window systems rely on establishing a number of point to point connections from one host (the application sharing host) to the remote hosts (i.e. participants hosts). Large amount of data in an X-connection flow from the client to the server (i.e. requests) need to be replicated on these connections, making this form of distribution highly expensive. In particular for large conferences (e.g. with hundreds of thousands of participants), this would be quite limited in bandwidth.

On one hand, the availability of the MBone network and the advent of multicast transport protocols distribution via point-to-point connections appears particularly limited and degrades the performance of the shared window system when it incorporates a large number of users. On the other hand, the X protocol is designed for running between a client and a server on a point-to-point connection.

That's why the task of designing a shared window system over IP Multicast relies on how we can fit the X protocol requirements into the network model features in order to achieve a high scalability and efficiency of the designed system. In the next section we expose the design approach and the mechanisms used to develop a shared window system that scales to a large number of users.

4. Design

To achieve a maximum of scalability we adopt a distributed architecture with a centralized shared application instance. This allows to distribute the sharing functionality among all cooperating hosts, and hence alleviate the load from the shared application site which will not be insensitive to the growth of the number of participants.

To achieve a correct and a consistent shared window session, loss of X protocol data is not acceptable. Since IP Multicast provides many-to-many unreliable communications, some form of multicast reliability is required in the context of a shared window system. The reliable multicast protocol adopted here and its features will be discussed later.

4.1 Application Data Units

The shared window system's (SWS) data model is determined by both user requirements and network requirements.

- The user requirements specify that several users must be able to work on the same shared application. Moreover, users should be able to view simultaneously on their displays any change that occur on the GUI of the shared application.
- Network requirement are to divide the data setup into parts that are independent as much as possible to reduce potential consistency[1] conflicts caused by failures and partitioning.

Moreover, the purpose of the SWS is to share the GUI of an X application (e.g. a set of X clients) running on one host by distributing its GUI to a number of participants at different hosts. Further, it is

1. Consistency as defined in the field of replicated databases: a consistent transaction takes the system from one internally self-consistent state to another. Consistency is one property of the so-called ACID properties.

not only the GUI that is shared but also the control of the shared application that is provided by several variants of floor control policies.

From both this requirements and the purpose of the SWS, we choose an X protocol request [22] as an application data unit that can be processed independently[1] at the SWS application layer. Multiple requests can be grouped and transmitted together. This group of requests represent a logical operation requested by a client from an X server (e.g. draw a line in the display). It's necessary to group these requests that belong to a same context of a logical operation.

In our context of the SWS, an Application Data Unit (ADU) is not a unit that can be viewed and manipulated directly by a user. Rather, it represents a unit that can be manipulated and processed by the application layer of the SWS. This is due to the context of SWS which has not only one type of application to share over multiple hosts as for example the case of the Network Editor Hand97, but a class of single-user X applications, each one having its special semantics.

Interactivity with the shared application is achieved by allowing a user to enter input to the shared application. This is allowed by transmitting generated events [22] at a remote X servers to the shared application site. These events will not be distributed to all users (i.e. receivers), because these events will be processed by only the shared application (or the X client). That's why events can not form a part of an ADU that is processed by all participants' hosts. This will be discussed in the next section on the distribution of the data model.

4.2 Distributing the data model

When a user interacts with a shared application, a group of requests are generated from this application and should be distributed to all the participant hosts. Since, our design is based on ALF [27], these multiple ADUs are transmitted over IP Multicast using the Real Time Transport protocol [25] (RTP).

A key goal in RTP is to provide a very thin transport layer without overly restricting the application designer. In accordance with ALF the definitions of several fields in the RTP fields are referred to an "RTP profile" document, which specifies their semantics according the given application. For our shared window system we have defined an RTP profile for carrying X protocol data requests over IP Multicast. This profile is being to be submitted to the IETF as an INTERNET-DRAFT [10]. Multiple requests that result, from a logical operation sent from the client to the server, can be carried over an RTP packet. All different packets of the RTP payload format are well defined in [10].

In order to reduce the effect of loss packets on the quality of the shared window system (i.e. speed of displaying changes occurred on the GUI), the application should decode each Network Data Unit (NDU) that it receives. This implies that the ADU should not be carried over several NDUs.

Furthermore, to allow a maximum of flexibility and efficiency, it's necessary that the source should transmit packets that are not too large compared to the maximum packet size allowed by the underlying network protocol (i.e. NDU). For unicast transmissions, a source can use "Path MTU Discovery" [20] to learn about the maximum transmission unit allowed along a certain unicast path through the network. But, for multicast transmission, we have no such path discovery mechanism and rely a rule of thumb for choosing packet sizes [17]: The size chosen is small enough to be carried by all of the prevalent physical-layer network technologies and is simultaneously large enough to amortize the cost of packet headers. In the MBone, we currently use packet sizes of roughly 1000 bytes.

1. Independently of other requests that arrives in order.

Thus, each RTP packet have a size of 1000 bytes (Path MTU for multicast). Then, multiple requests can be carried over a single RTP packet. Otherwise, if a request have a size greater than this MTU, it is then fragmented and transmitted over several RTP packets.

This distribution model deals only with the distribution of generated X protocol data requests from the shared application site. In fact, for the requirements of interactivity with the shared application, it is necessary to redirect the input of a remote user (i.e. generated X events) to the shared application site. Then, these events are not subject to distribution, and then are transmitted over a unicast channel (e.g. TCP/IP) to the shared application site. The architecture of the over all system will be detailed in a later section dealing with the system architecture.

4.3 Reliability Mechanisms

Since, using IP Multicast service model [5] which provides unreliable many-to-many communications, reliability mechanisms are required to allow a consistent shared window session over IP Multicast. The literature is rich with architectures and protocols for reliable multicast [6, 23]. Because we need a reliable multicast protocol we should examine the requirements of the shared window system, and hence the convenient reliable multicast protocol should fit these requirements. The following are the most important requirements of the shared window system:

- Reliability: X protocol requests are loss-intolerant data and should be reliably delivered to all participant's hosts. Loss of requests at an end-host may cause inconsistencies in the set of resources managed by the X server. The latter may generate errors to indicate for example that these requests reference resources that are not already created.

- Ordering: in the case of a shared window system there is *a priori* a requirement for ordered delivery of X protocol data requests, because some of these requests may have attributes that reference for example resources created in precedent requests.

- Latejoin step: in a shared window session we should allow a latecomer to join the shared session and to interact with the shared application if it's allowed to do so.

- Scalability: the shared window system should scale to a large number of participants. This is a requirement of a teleteaching scenario in which we have several distributed classes and a professor that remotely teachs a special course to all these classes. The scalability is also needed when a large number of users join an on-going shared window session.

For reliability, ordering and scalability requirements we have opted for the Scalable Multicast Protocol (SMP) [12] built on top of the Scalable Reliable Multicast (SRM) protocol [8]. Since, SRM does not provide guaranteed ordered delivery of data, SMP uses SRM mechanisms for loss recovery and enhance reliability with ordered delivery of data packets.

SRM reliability mechanisms are classified as receiver-oriented loss recovery, each receiver is individually responsible for detecting loss and requesting retransmission. Loss is normally detected by finding a gap in the sequence space of received packets. When members of the group detect a loss, they wait a random time and then multicast their "repair request", to suppress requests from other members sharing that loss.

In SRM the interval over which the request timer is set is a function of the member's estimated distance to the source (S) of the packet. If a host A detects a loss it schedules its request timer from the uniform distribution on

$$[C_1 d_{S,A},\ (C_1+C_2)\ d_{S,A}] \text{ (in seconds)}$$

seconds, where $d_{S,A}$ is host A's estimate of the one-way delay to the source S. The numbers C_1 and C_2 are parameters of the request algorithm. If host A receives a request for the missing data before its own request expires then host A does an exponential back-off and resets its request timer[1].

When some other member B which is capable of answering, receives a request from A, it sets a repair timer to a value from the uniform distribution on

$$[D_1 d_{A,B}, (D_1+D_2)\, d_{A,B}] \text{ (in seconds)}$$

where $d_{A,B}$ is host B's estimate of the one-way delay to host A and numbers D_1 and D_2 are parameters of the repair algorithm. If host B receives repair for the missing data before its repair timer expires, then host B cancels its repair timer.

In order to work most effectively, SRM requires that all participants calculate a delay (round trip time) matrix to all other participants's hosts, and this is done using timestamps in the session message [8]. These session messages for reliable multicast [7] are been proposed to enable receivers to detect the loss of the last packet in a burst, and to enable the sender to monitor the status of the receivers.

The SMP protocol based on SRM provides reliability and ordered delivery of data packets. However, the requirement of allowing a latecomer to join a shared window session can not be provided directly by SMP. Because a latecomer needs a special shared window context to be able to display the current shared window and to apply with consistency the received requests to the latecomer's X server. This needs a join session protocol which is described in detail in the next section.

4.4 Join Session Protocol

In contrast to an audio/video conference where a latecomer can join the conference without a need of a particular context of the current conference session, in a shared window session the latecomer needs a *shared window context* of the current session. This is due to the nature of the X protocol data requests that are always dependent of a set of allocated resources on the X server. It is necessary to note that a shared window context is not static but is dynamically changing each time a participant interacts with the shared application.

A shared window context C_i^t can be defined at an instant t (of the space time of the shared window system) and relative to a shared X client i, as to be composed of two components:

- The context of allocated X resources at the instant t
- The context of the Graphical User Interface (GUI) of the shared X client i at the instant t

The instant t can also represent the sequence number of the last request group distributed to all the participants. In other words a shared window context is consistent for a particular group of request's sequence number (i.e. RTP packet sequence number). After imposing this context to a latecomer's X server, the first valid group request that we should apply must have the next sequence number (i.e. last sequence number +1).

This context and the methods that allow to maintain and retrieve locally these two components are well studied in [9] when dealing with the selective recording and replaying of X protocol data streams. It's also important to note that the shared window context is not only provided by the shared application site but also can be retrieved from any host that participates in the shared window session.

1. if the current timer is chosen from the uniform distribution on $2^i\, [C_1 d_{S,A}, (C_1+C_2)\, d_{S,A}]$, then the back-off timer is chosen from distribution on $2^{i+1}\, [C_1 d_{S,A}, (C_1+C_2)\, d_{S,A}]$.

While joining a shared window session, a latecomer should request the current shared window context. We present in the following two variants of a scalable request and delivery of the shared window context in the case of individual joins and the case of a step join (a high number of users that join simultaneously the shared window session).

4.4.1 Individual join

Distributed point-to-point delivery

In the case of individual join or isolated join we present the following steps that describe the join session protocol. This protocol is performed by four steps:

1. Upon joining the multicast group the latecomer multicasts a *join-request* packet to request the current shared window context. The format of the request is described in [10].

2. Each receiver (i.e. potential provider) that is able to provide the shared window context, sends a *join-response* packet on the multicast channel. To avoid the ***implosion problem*** [21], before sending the *join-response* each potential provider schedules a ***timer*** in the interval [0, T]. This allows to suppress scheduled responses from others hosts when receiving a response from a special host. The choice of the distribution timer will be discussed later.

3. The latecomer elects a provider based on the distance to the provider host (i.e. the nearest will be elected), by sending on a unicast channel (e.g. TCP/IP) a *join-confirm* to an elected provider.

4. The elected provider sends on unicast the current shared window context to the latecomer.

The timer distribution is exponential and based on the work done in [21] to allow an optimal feedback to the latecomer requesting a shared window context. The parameters T and λ of the exponential timer are discussed in [21] to allow a required number of feedback responses and an optimal feedback latency. The exponentially distributed timer is given by the density :

(EQ 1)

$$f_{Zi}(zi) = \begin{cases} \frac{1}{e^{\lambda}-1} \cdot \frac{\lambda}{T} e^{\frac{\lambda}{T} \cdot z_i}, 0 \le z_i \le T \\ 0, otherwise \end{cases}$$

When receiving the shared window context, the latecomer should apply the next group request which is consistent with the current window context.

This point-to-point distributed delivery of the shared window context is convenient to allow a low number of simultaneous latecomers which are geographically dispersed to retrieve the shared window context from a nearest participant. However, when a high number of latecomers join the multicast group this protocol does not scale (e.g. implosion of join-request packets) with this high number of latecomers and hence, the point to point delivery of the shared window context consumes a large bandwidth and thus is too costly.

4.4.2 Step join

When we have a high number of users joining simultaneously the multicast group, it is preferabely to provide the shared window context to all these users. This can be done either in a centralized multicast or by a distributed multicast delivery of the shared window context.

• *Centralized multicast delivery*

In this case the shared context is transmitted over a particular multicast join channel. This allows to separate the join traffic from the shared window system traffic and hence the join traffic will not affect participants that are already involved in the shared window session. This follows the concept of layered multicast transmission. The protocol is performed by the following steps:

1. The source or the shared application site, multicasts on the join-channel periodically (i.e. each period of time T_j) a poll request for requesting the current shared window context.

2. When a latecomer joins the multicast join-channel, it listens to this channel.

3. Upon receiving a poll for request from the source, each receiver listening to the join-channel schedules an exponential timer in the interval $[0, T_c]$.

4. If the timer expires and no join-request from an other host is received, the latecomer multicasts a join-request. The latecomer host suppresses its timer, if it receives a join-request from an other host.

5. If the source receives a join-request it multicasts the shared window context on the join-channel.

The reliability mechanisms used to recover lost packets on the join-channel are those of the SMP (i.e. SRM) protocol. The format of the packets for the shared window context are detailed in the Draft [10].

• *Distributed multicast delivery*

This is adapted when we want to serve locally a high number of latecomers by dedicated servers. This corresponds to a teleteaching scenario in which we have several classes that are located in the same campus and join simultaneously the current shared window session. Thus, this distributed protocol is adapted to the following topology:

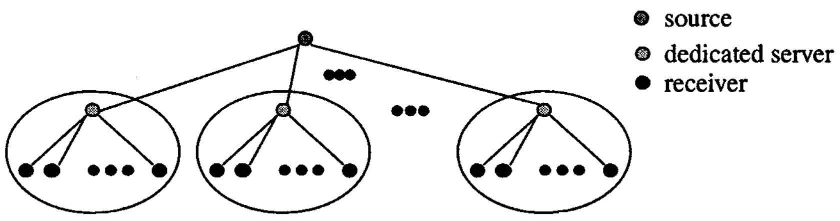

FIGURE 5. Topology for Distributed Delivery of Shared Window Context

Each dedicated server plays the role of the source in the case of centralized multicast delivery of the shared context. All the steps are similar to those of the centralized case with the following exceptions:

1. The dedicated server (can be a particular receiver in the shared window session) multicasts the poll for request with a special scope (TTL) to the group.

2. The interval of the exponential timer (i.e. interval for the choice of the timer is $[0, T_d]$) for the join-request suppression is adapted to the size of local groups.

The choice of a dedicated server for multicasting the shared window context is then statique. This can constraint the flexibility of the shared window system, since we need to reserve a number of hosts to perform this protocol. However, an other variant of this protocol can be designed to elect a server that is responsible for providing the shared window context to a group of latecomers. This will be studied in future work.

In our shared window system we have opted for the centralized multicast delivery protocol, because it can be simply implemented and we should not reserve special sites that plays the role of dedicated servers for providing the shared window context. This protocol also scales to a high number of latecomers.

5. System Architecture

The proposed architecture for the shared window system based on IP Multicast aims to satisfy the following requirements:

- Distribute the sharing functionality among all involved hosts
- Allowing a large number of participants to join and leave the shared window session
- Allow and control interactivity with the shared application

The most important feature of the shared window system is its distributed architecture which allows to process the received requests and adapt them to the local X server (figure 6). Thus, this alleviates the load of the shared application site which becomes insensitive to the number of users that can join the multicast channel to participate to the shared window session. The sharing functionality are described in a later section of the system implementation.

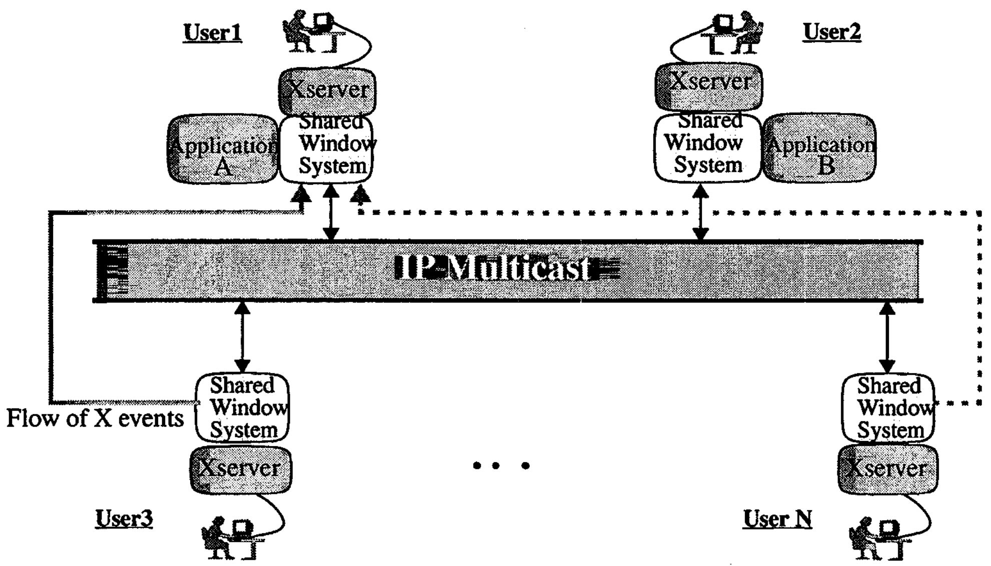

FIGURE 6. Architecture of the Shared Window System

Since using IP Multicast, a large number of participants can join as well as leave the multicast group without degrading the performance of the shared application or those of the other participants. A latejoin procedure that allows a spontaneous latejoin (i.e. without restriction) should be allowed at the end-host shared window system agent (figure 6). A major design choice is to separate the latejoin traffic from the shared window system traffic by reserving a *join multicast channel* for providing the shared window context to a large number of latecomers. This also prevents congestion of the network and receivers that are already involved in the shared window session.

While more than one user can interact with the shared application, there is at most one participant (i.e. current floor holder) that can effectively enter input to the shared application at a given point in time. To allow interactivity it is necessary to redirect the input (i.e. generated X events) of a remote user to the shared application site. Thus, these events are carried over a unicast channel (e.g. TCP/IP) to the shared application host (figure 6). Consequently in our architecture, each host that is allowed to interact with the shared application opens a unicast channel to the shared application site to transmit the local generated X events.

6. Floor Control Policy

This section describes a scalable **floor control** mechanism that is provided by the shared window system to allow different users among a high number of participants, to interact with the shared window application. Various policies of floor control have been proposed [13] for shared window systems.

In the Xwedge framework [13], the application owner's control agent grants the floor to a user and also revokes it using *EnableInput* and *DisableInput* request primitives of the control protocol. However these requests can be applied once the candidates for the *floor holder* are well known by the control agent. By contrast, in a large scale environment there may be a large number of candidates for the floor holder. Thus, we present a scalable policy to solicit candidates for a floor holder and to grant the floor to a selected candidate.

In this policy we consider two group classes among all participants: a group of experts (or professors) and an other one of ordinary participants, such as students. There is a **priority** between these two groups: a member of the group of experts have a priority to hold the floor than an other one of the other participants (e.g. students).

The steps of the floor control policy are as follows:

- When the application owner (e.g. chairman of the conference) decides to pass the floor to an other participants, he/she sends a request poll packet on the multicast channel for soliciting candidates for the holding the floor.
- Each participant of both two groups that want to hold the floor can request that from its local control agent. The latter will use this request when it listens the poll request packet sent from the source (i.e. shared application site).
- When a participant control agent listens the request poll from the source, it sends a *token-request* to the source (if the participant have already requested the floor). In order to avoid implosion of feedback to the shared application host :
    - For the expert's group the token-request is delayed for a time chosen with an exponential distribution with parameters λ_e and T_e. If the timer expires and no token-requests are received then this expert participant send its token-request packet.
    - Each other participant's control agent delays the token-request for a random time chosen with an exponential distribution with parameters λ_s and T_s.
    - For the two groups, if a token-request is received from any other participant, the current timer is cancelled and no token-request is sent.

The goal of delaying transmission of token-requests is to suppress other token-requests from other hosts and hence minimize the feedback to the source. An other important goal of this floor control policy is to allow members of the high priority group (i.e. experts group) to send their request-tokens before members of other groups.

This is related to the **feedback latency** (E(M)) which has the following formula [21].

$$E(M) = T \times \int_0^1 \left(\frac{e^{\lambda} - e^{\lambda m}}{e^{\lambda} - 1}\right)^R dm \qquad \text{(EQ 2)}$$

Where M is the exponential random variable with parameters λ and T. R denotes the number of group participants. For the expert group the feedback latency $E(M_e)$is given by:

$$E(M_e) = T_e \times \int_0^1 \left(\frac{e^{\lambda_e} - e^{\lambda_e m}}{e^{\lambda_e} - 1}\right)^{R_e} dm \qquad \text{(EQ 3)}$$

and for the student group $E(M_S)$ is given by:

$$E(M_s) = T_s \times \int_0^1 \left(\frac{e^{\lambda_s} - e^{\lambda_s m}}{e^{\lambda_s} - 1}\right)^{R_s} dm \qquad \text{(EQ 4)}$$

These latencies depend strongly on the choice of the width of the interval [0, T] for the exponential distribution. The figure 7 shows the feedback latency for the two groups and their differences ($E(M_S)$-$E(M_e)$. The constant c is the maximum *One Trip Time* (OTT) delay between any hosts in the network.

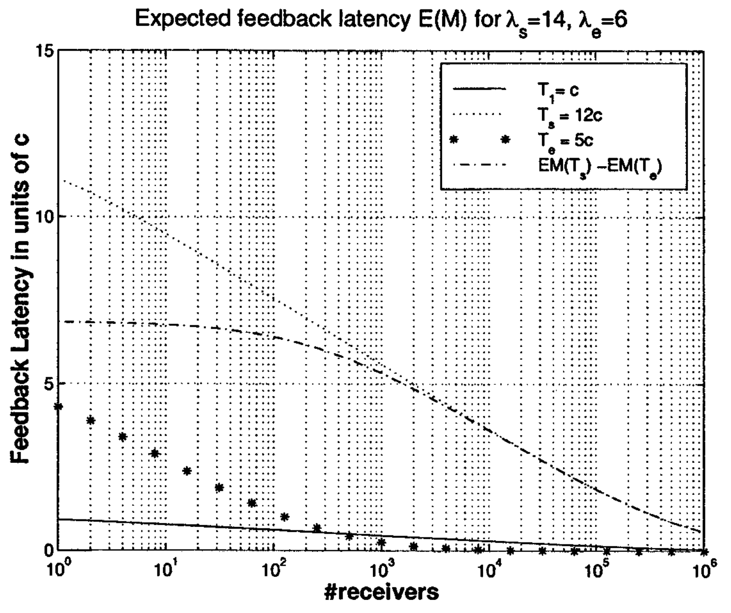

FIGURE 7. Feedback Latency for the two Groups

The parameters λ_e and T_e are chosen corresponding to the size group of the experts with a maximum 100 receivers (R_e). While, the parameters λ_s and T_s of the student group are chosen regarding the size of the student group which can be up to 10^4 receivesr (R_s). In order to work properly with the fixed priority between the two groups, the control floor policy needs to satisfy the following relation:

$$E(M_s)\text{-}E(M_e) > c$$

This relation imposes that the token-request latency of the student group should be greater than the one of the expert group plus a constant c (OTT). This relation is satisfied when the group size of the students is not greater than 10^4, and was not satisfied when the group size is up to 10^5 (in figure 7, the difference of the two latency is lower than c if the group size is up to 10^5)

In order to guarantee this relation, and consequently the priority of the experts group in sending their ***token-requests,*** a possible solution is to add a constant delay of c for each participant in the student group. Then, when receiving the *poll* for holding the token from the source, each participant from the students' group chooses its exponential timer from the interval [0, T_s] after a constant back-off delay of c seconds.

Once the source receives a small number of token-requests that represents the candidate floor holders, the shared application owner uses the same primitives as in [13] (*EnableInput* request sent in unicast) to grant the floor to a particular candidate floor holder.

The floor token is also revoked by the shared application owner by using the *DisableInput* request, that can breaks down the connection to the current floor holder's host.

Finally, the floor policy proposed in this section allows the shared application owner to solicit floor holder candidates from a high number of participants with two level of priorities. After having a set of floor holder candidates, the floor is granted to a participant with the high level of priority (e.g. from the experts' group).

7. Implementation

The environment of our shared window system consists of several interconnected workstations. In order to use the shared window system, a user needs to run both the **shared window system agent** and the **scalable multicast server** (SMP). The shared window system agent intercepts and processes the **X protocol** data streams, while the scalable multicast server guarantees a reliable and an ordering transmission of X protocol requests on IP Multicast. Hence, we separate low level communications from the shared window system core.

The shared window system agent is an extended version of the Xwedge framework [13], and consists of two components which are the ***pseudo server*** and the ***pseudo client.*** The global system architecture is detailed in figure 8. This figure shows a sharing scenario of two X clients simultaneously between two users, user1 and user2. All other users share the first X client (Client 1) running on the user1 's host. Figure 8 shows the communication paths for sharing these two clients according to this scenario. We expose in the next the main tasks of the pseudo server and the pseudo client.

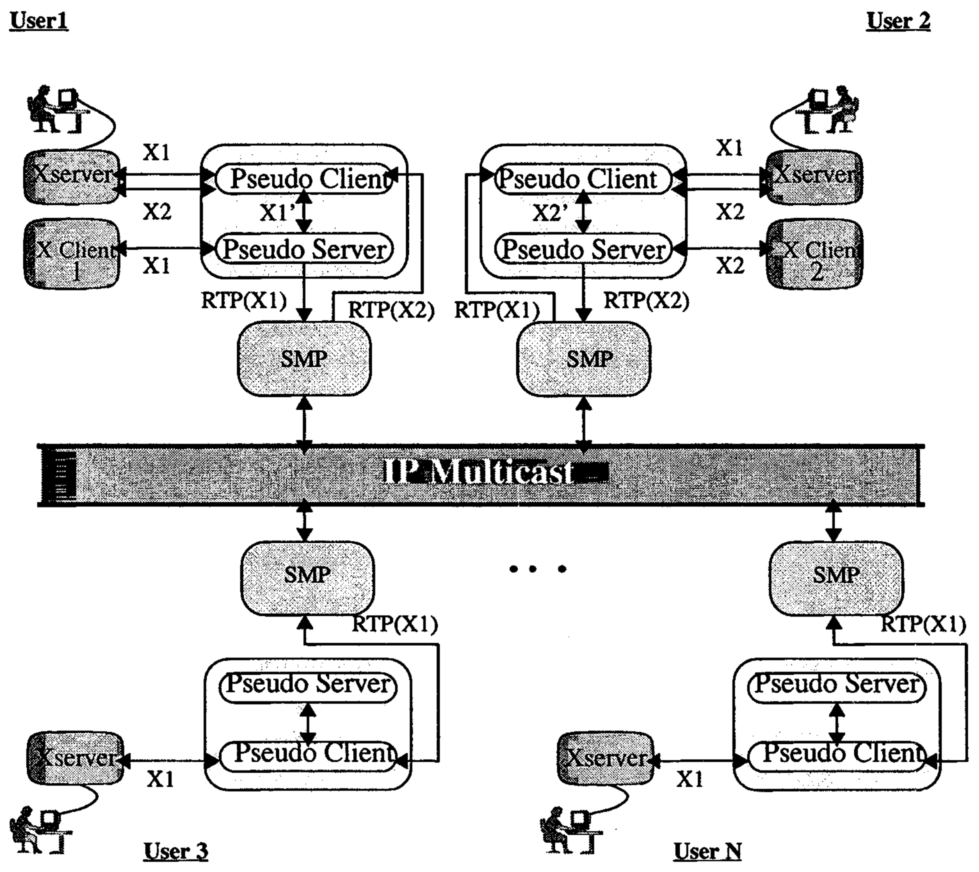

FIGURE 8. Global System Architecture

Pseudo server

Besides the traditional role of the pseudo server which consists of intercepting the X protocol data streams, our main extensions to the framework described in [13] are:

• Allowing the pseudo server to maintain the created X resources by the shared X client [Goop94] and to generate the shared window context which consists of both the context of allocated X resources and that of the shared X client's GUI described in section 4.4. The functions of how generating these contexts are described in [9].

• Encapsulates a group of X protocol requests in an *RTP* packet and send them to the SMP server in order to reliably transmit them over IP Multicast. The format and the semantics of the RTP packet fields are described in [10]. We note here that requests from multiple shared X clients can be transmitted over the same IP Multicast group.

• Converts the generated X events that receives from a remote host. This is necessary to allow a correct interpretation of a remote user inputs to the shared application running locally.

Pseudo client

The pseudo client plays the role of an ordinary client when communicating with an X server. Its main functionality are mapping the received X protocol data requests to the respective local characteristics of an X server [2] (e.g. conversion of resource IDs, screen coordinates, atom numbers, pixel values, etc...). The pseudo client is extended with the following functionality:

- Decodes received RTP packets and extract the encapsulated requests.
- Most requests that need replies (round trip requests) are satisfied locally.
- Allowing the shared application GUI to be **persistent**. Persistence here means that the local X server is responsible for refreshing the shared application GUI when a window is hidden by an other window application. This is performed by adding the *backingstore* attribute to each received *XCreateWindow* Request.

We note that the latejoin protocol is incorporated inside an extended version of the SMP framework [12]. This is done first by implementing a periodically poll for requesting the shared window context on the *join multicast-channel.* Secondly, by extending the protocol primitives (protocol command) between SMP and SMP's client (i.e. pseudo server or pseudo client). For example, in the side of the shared application host, new protocol primitive allows SMP to send a *join-request* indication to the pseudo server. The latter generates the current shared window context (a sequence of requests) and sends it on the join multicast-channel via SMP.

8. Performance & Discussion

The goal of designing and developing a shared window system over IP Multicast is to bring together a high number of participants in the same shared window session. This is a requirement for various application scenarios such as an *open teleteaching session* in which we have a number of professors and experts and a high number of students. Moreover, *software training or promotion* for big and distributed companies requires a shared window session that provides a joint viewing for thousands of distributed people. Our shared window system answers these requirements and provides a designed shared window system and a first implementation version in order to carry "large scale" experiments on the MBone network.

However, the performance of the shared window system can be constrained when various mechanisms of minimizing the feedback to the shared application host can not be guaranteed. Such a mechanism is related to the **persistence** of the shared application's GUI at a participant display host. When we can not guarantee the GUI persistence (because an X server does not guarantee the *backingstore* feature in the XCreateWindow request) there is no efficient mechanism that allows to refresh the GUI locally. Moreover, there may be some Round Trip Requests (RTR) which requires replies from the server and can not be satisfied locally.

Finally the nature of the X protocol which is not designed to be run in a point-to-multipoint environment, introduces a new level of complexity when designing a consistent shared window system over IP Multicast. Since, shared window systems depends strongly on the X Window System [22], and efficient sharing of X window applications requires the design of a new generation of X window systems (i.e. a new design of X servers) which define a new level of abstractions for X protocol requests and minimizing round trip requests. These abstract requests should be applied to a large number and heterogeneous X servers, in order to have the same result on the application GUI.

9. Conclusion

In this paper we have described the functionality and features of a shared window system for the X window system over IP Multicast. In contrast to previously implemented systems, which are limited in bandwidth and efficiency, our shared window system enables a high number of users to share a single X Window application and to join and leave the multicast shared window session in a scalable manner.

We used general design principles- those of IP multicast, light-weight sessions, and application level framing as starting points. The application data model is intended for generic X window applications and hence based on X protocol requests. The data distribution model uses a group of requests as an ADUs which reflects a logical operation required by an X client from an X server. In addition, The distribution model uses the Real Time Transport protocol (RTP) to distribute the ADUs to all host participants. The use of RTP provides a very thin transport layer that allows a semantic background for integration and cooperation of various types of multicast applications.

The reliability mechanisms used in our shared window system are based on SRM mechanisms. Because X protocol data requests are loss intolerant media, these mechanisms provides a full reliability and ordering of transmitted requests. SRM is also a more appropriate choice for reliability, since it is scalable and allows local recovery of lost packets. This scalability feature is also required in the design process of the shared window system.

In order to accommodate a large number of latecomers to a shared window session, a join protocol is designed on top of SRM to allow a scalable provision of shared window context to these recent participants. For high requirements of scalability and efficiency, the shared window context is transmitted over a separate join-multicast channel to which connects each latecomer. Reliability mechanisms used to recover lost packets in this channel are based on SRM.

A floor control policy is also designed to allow participants to interact with the shared application. This floor control policy provides mechanisms to solicit candidates among a large number of users to be floor holders. This policy introduces two levels of priorities among participants, and privilege the high level priority group to respond to this solicitations and thus to be assigned the floor token.

The global shared window system architecture is detailed and their components are exposed. The system implementation is based on previous work of shared window systems, such as Xwedge, XTV and a reliable multicast server SMP. The implementation status is in its first version which consists of allowing the distribution of X requests and providing the service of late joining a shared window session.

Finally, implementation of the proposed control floor policy and hence allowing interaction with the shared application will be done in the near future. For future work, a shared window control protocol that uses a multicast control channel must be designed to allow various levels of floor control policies to be implemented. This control channel can also distribute to participants general information about the current floor holder and current shared applications as well as multicast messages from a conference chair or an expert participant in the shared window session. Work can be investigated in allowing some reliability mechanisms that use Forward Error Correction based on some relevant requests transmitted by system. An algorithm for a partial order acceptance of received requests can be developed to process requests that arrive out of order and that are independent (i.e. uses resources that already created in the X server) of precedent transmitted requests. Finally, we note that some proposed mechanisms may be generalized to other multicast applications.

10. References

1. M. Altenhofen. Erweiterung eines fenstersystems fur tutoring-funktionen. Master's thesis, Diploma Thesis at university of karlsruhe, Karsruhe, 1990.

2. H. M. Abdel-Wahab and M. A. Feit. XTV: A framework for sharing X window clients in remote synchronous collaboration. In *Proceedings of the IEEE Conference on Communications*

Software:Communications for Distributed Applications and Systems, pages 159 – 167, North Carolina at Chapel Hill, 1991.

3. C. Bormann and G. Hofmann. Xmc and Xy, scalable window sharing and mobility or from X protocol multiplexing to X protocol multicasting. In *Proceedings of the 8th Annual X Technical Conference*, January 1994.

4. G. Chung. Accommodating latecomers in a system for synchronous collaboration. Ms thesis, University of North Calorina at Chapel Hill, August 1991.

5. S. Deering and D. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, pages 85–111, May 1990.

6. C. Diot, W. Dabbous, and J. Crowcroft. Multipoint communications: a survey of protocols, functions and mechanisms. *IEEE Journal on Selected Areas in Communications*, 1997.

7. A. Erramilli and R.P. Singh. A reliable and effecient multicast protocol broadband broadcast networks. In *Proceedings of the ACM SIGCOMM*, pages 343–352, 1987.

8. S. Floyd, V. Jacobson, S. McCanne, C. Liu, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *ACM Computer Communications Review*, August 1995.

9. L. Gannoun and J. Labetoulle. Scalable asynchronous interaction based on selective recording and replaying of X protocol data streams. In IEEE Computer Society Press, editor, *Proceedings of the IEEE PROMS-MmNet*, Santiago, Chile, November 1997.

10. L. Gannoun. RTP payload format for X protocol media streams. INTERNET DRAFT,, February 1998. available at: http://www.eurecom.fr/gannoun/rtp_forX11.txt.

11. C. Goopeel, K. Jeffay, and H. Abdel-Wahab. Accommodating latecommers in a shared window systems. *Computer Communications*, 17(1):7–16, january 1994.

12. Grumann, M. Master's Thesis. SMP - a scalable reliable multicast protocol. Master's thesis, Master's Thesis,University of Mannheim, Germany, Feb 1997. available at http://www.informatik.uni-mannheim.de/informatik/pi4/projects/teleTeaching/publikationen.eng.html.

13. T. Gutekunst, D. Bauer, G. Caronni, Hasan, and B. Plattner. A distributed and policy-free general-purpose shared window system. *IEEE/ACM Transactions on Networking*, February 1995.

14. M. Handly and J. Crowcroft. Network text editor(nte) a scalable shared text editor for the mbone. In *Proceedings of the ACM Sigcomm*, Cannes, FRANCE, September 1997.

15. M. R. Macedonia and D. P. Brutzmannn. Mbone provides audio and video across the internet. *IEEE Computer*, April 1994.

16. S. McCanne. *Scalable Compression and Transmission of Internet Multicast Video*. PhD thesis, University of California, Berkley, California, December 1996.

17. S. McCanne and M. F. Speer. Rtp usage with layered multimedia streams. Internet Draft, December 1996.

18. G. McFarlane. Xmux-a system for computer supported collaborative work. In *Proceedings of the 1st Australian Multi-Media Communications, Applications and Technology Workshop*, pages 12–28, Sydney, 1991.

19. W. Minenko. The application sharing technology. *The X advisor*, 1(1), June 1995.

20. J. Mogul and S. Deering. Path mtu discovery. ARPANET Working Group Requests for Comment, DDN

Network Information Center, SRI International, Menlo Park, CA, November 1990.

21. J. Nonnenmacher and E. Biersack. Optimal multicast feedback. *To appear in the Proceedings of the INFOCOM'98*, 1998.

22. A. Nye. *X Protocol Reference Manual.* Sebastopol: O'Reilly and Associates, 1992.

23. Multicast Transport Protocols WWW Page. Url http://hill.lut.ac.uk/ds-archive/mtp.html.

24. K. Schmidt and L. Bannon. Taking cscw seriously. *Computer Supported Coopeative Work: CSCW,* 1(1-2):7–40, 1992.

25. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp: a transport protocol for real-time applications. Internet enginnering task force, audio-video transport working group, January 1996. RFC-1889.

26. R. W. Sheifler and J. Gettys. The X window system. *ACM Trans. Comput. Graphics*, pages 79–109, 1986.

27. D.D. Clark nad D.L.Tennnhouse. Architectural considerations for a new generation of protocols. *In Proceedings of the ACM SIGCOMM,* Philadelphia, Pa, 1990.