Implementação de um Serviço de Sincronização Intra e Intermídia Considerando Relações Temporais Implícitas[1]

Marcelo Gonçalves Rubinstein
rubi@gta.ufrj.br

Otto Carlos Muniz Bandeira Duarte
otto@gta.ufrj.br

Grupo de Teleinformática e Automação - GTA
Universidade Federal do Rio de Janeiro
COPPE/EE - Programa de Engenharia Elétrica/Departamento de Eletrônica
Caixa Postal 68504 - CEP 21945-970 - Rio de Janeiro - RJ - Brasil
FAX: +55 21 2906626

Resumo

Este trabalho descreve a implementação de um serviço de continuidade e sincronização intermídia e do seu respectivo protocolo utilizando uma arquitetura de alto desempenho. Este serviço visa aplicações multimídias que envolvam fontes de dados armazenados ou ao vivo, relações temporais implícitas, múltiplas fontes e destinos e não utilizem relógio global. O serviço é definido através das suas primitivas de serviço. O tipo e o formato das unidades de dados de protocolo e o ambiente de desenvolvimento são apresentados. Foi utilizada uma arquitetura de implementação de alto desempenho, contendo módulos de escalonamento de tarefas com prioridades, gerenciamento de memória otimizado e gerenciamento de temporizações. A implementação das camadas Aplicação, Sincronização e Transporte é descrita em linguagem natural.

Abstract

This paper defines a continuity and intermedia synchronization service, and specifies its protocol using a high performance architecture. This service aims multimedia applications that involve stored or live data sources, implicit temporal relationships, multiple sources and destinations, and do not utilize global clock. The service is defined by its service primitives. Protocol data units' type and format and development environment are presented. A high performance implementation architecture has been used, defining specialized modules of priority-based task scheduling, optimized memory management, and timer management. Application, Synchronization, and Transport layer implementations are described in natural language.

1 Introdução

O novo conjunto de aplicações distribuídas em redes, a ser suportado pelas redes de alta velocidade, inclui aplicações multimídias interativas com vários participantes (videoconferência e telesseminário), visualização científica (reações nucleares e imagens de satélite), visualização médica e telediagnóstico, distribuição domiciliar de TV de alta definição (*High Definition TeleVision* - HDTV), videotelefonia, teleducação (videobiblioteca e educação

[1]Este trabalho foi realizado com recursos da UFRJ, FUJB, PROTEM-CC, CNPq, FAPERJ, CAPES e COFECUB.

domiciliar distribuída), realidade virtual, trabalho cooperativo (*Computer Supported Cooperative Work* - CSCW), transferência de dados a alta velocidade e outras [Albu95]. Para este conjunto de aplicações, novos requisitos de Qualidade de Serviço (QoS) foram introduzidos, tais como o sincronismo entre as amostras de áudio e entre os quadros de vídeo.

A integração de mídias requer técnicas de sincronização que devem reproduzir no(s) destino(s), as relações temporais existentes na(s) fonte(s) quando da captura da(s) mídia(s). **Fluxo** de mídia, ou simplemente fluxo, é a seqüência de unidades de dados referentes a uma mídia transferida pela rede e **fluxo isócrono** é o fluxo que requer a geração e a apresentação contínua de suas unidades de dados a intervalos fixos de tempo. A utilização da sincronização em aplicações multimídias é necessária quando a cadência de um fluxo isócrono e/ou a relação temporal entre fluxos, isócronos ou não, precisam ser mantidas. A apresentação de fluxos sincronizados é essencial para obter-se uma impressão de algo natural, pois fluxos fora de sincronismo são percebidos como sendo algo artificial, estranho ou até mesmo enjoado [Stei96].

Nos últimos anos, várias técnicas de sincronização foram propostas. O servidor de Anderson e Homsy [AnHo91] utiliza algoritmos para sincronizar aparelhos de entrada e saída de mídias contínuas em uma única estação utilizando um sistema operacional de tempo real. O modelo de Horn e Stefani [HoSt93] utiliza uma linguagem de programação chamada ESTEREL para descrever os métodos e o comportamento de objetos multimídias. Shepherd et al. [SHGC92] e Lamont e Georganas [LaGe94] descrevem, respectivamente, mecanismos de transporte e arquitetura de *software* para sistemas distribuídos. Uma arquitetura para sistemas multimídias em tempo real, na qual pontos de sincronização são utilizados para sincronizar fluxos relacionados, é descrita por Nicolaou [Nico90]. No esquema de Ferrari [Ferr92], é descrito um mecanismo distribuído para controlar a variação do atraso (*jitter*) em redes que têm os comutadores e as operações de comutação muito complexos. O modelo de Little e Ghaffor [LiGh91] utiliza Redes de Petri para a Composição de Objetos (OCPNs) a fim de especificar as relações temporais entre os objetos multimídias; enquanto que Woo et al. [WQG94] propõem uma extensão da OCPN, a XOCPN (*eXtended Object Composition Petri-Net*), onde objetos são divididos em subobjetos a fim de melhor coordenar a continuidade de fluxos isócronos. Um extrator de dependências condicionais é proposto por Carmo e Courtiat, de modo a reforçar o poder da sincronização temporal [CaCo96]. A técnica de Escobar et al. [EPD94] utiliza um protocolo de sincronização adaptativo a mudanças de atraso de fluxos, baseado na utilização de relógios sincronizados e capaz de sincronizar aplicações ao vivo. Ramanathan e Rangan [RaRa93] propõem técnicas de realimentação e protocolos para a sincronização intra e intermídia de dados pré-armazenados, na presença de variação de atraso da rede e de diferenças não determinísticas nas taxas de captura e/ou exibição de mídias de diferentes estações, e o modelo de Son e Agarwal [SoAg93] baseia-se neste esquema, embora não trate de continuidade e aceite a utilização de relações temporais explícitas entre os dados da aplicação. Nenhuma das técnicas garante todos os requisitos das várias aplicações existentes [EFI94] e estas técnicas podem ser analisadas através de suas principais características [Rubi96, BlSt96].

Este trabalho descreve a implementação de um serviço de sincronização intra e intermídia, especificado através de suas primitivas de serviço, apropriado para aplicações multimídias que envolvam fontes de dados armazenados ou ao vivo, relações temporais implícitas, múltiplas fontes e destinos e não utilizem relógio global. O serviço provê a sincronização intramídia para as mídias mestras de todos os feixes de sincronização (conjunto de fluxos que possuem relações temporais entre si) e a sincronização intermídia, para todos os feixes de sincronização. O serviço é definido através das primitivas de serviço e o seu respectivo protocolo é especificado através de diagramas em SDL (*Specification and Description Language*) [ITU-T94]. Uma arquitetura de implementação de alto desempenho e

o ambiente de desenvolvimento relativo ao áudio e ao vídeo são apresentados. O sistema implementado, envolvendo as camadas Aplicação, Sincronização e Transporte, é descrito através de suas principais características.

O trabalho está organizado da seguinte maneira: a seção 2 apresenta o mecanismo de sincronização, a seção 3 descreve o serviço de sincronização proposto, a seção 4 descreve a implementação das camadas Aplicação, Sincronização e Transporte e na seção 5 são apresentadas as conclusões.

2 Mecanismo de sincronização

O mecanismo de sincronização destina-se a aplicações que possuem as seguintes características:

- envolvem fontes de dados armazenados ou ao vivo, de modo a cobrir todos os tipos de fontes de dados;
- envolvem relações temporais implícitas, pois o objetivo das apresentações é exibir os dados de maneira igual os quais foram capturados;
- envolvem múltiplas fontes e destinos, de modo a cobrir qualquer configuração de estações participantes das aplicações;
- não utilizam relógio global.

Para ter-se uma única base de tempo (essencial em sincronização), uma arquitetura centralizada foi utilizada. Com isto, as diferenças temporais entre os relógios envolvidos na sincronização são facilmente levadas em consideração, através de um mapeamento do relógio de cada fonte/destino no relógio do servidor (centralizador).

O mecanismo baseia-se em uma arquitetura centralizada constituída de um Servidor de Sincronização (SS) e diversas estações multimídias interconectadas por uma rede de computadores (figura 1). A idéia chave da centralização é concentrar a maior carga de processamento no Servidor de Sincronização, permitindo a utilização de estações multimídias de menor capacidade de processamento e, portanto, de menor custo. Além disto, o Servidor de Sincronização permite o atendimento de diversas aplicações ao mesmo tempo, garantindo que dados gerados por fontes distribuídas sejam apresentados de uma forma sincronizada nos seus destinos.

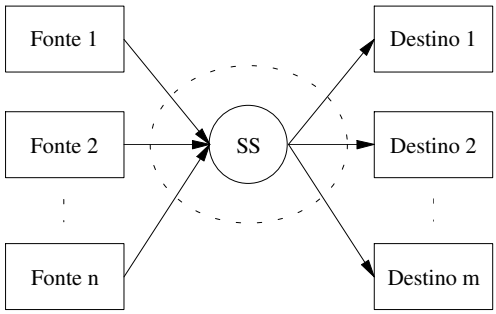

Figura 1: Arquitetura do sistema.

As relações temporais implícitas entre os dados são representadas através de **Estampas de Tempo Relativo** (ETRs) que indicam o tempo relativo à geração/exibição da primeira unidade elementar de uma mídia, doravante denominada **unidade de mídia** (UM), de

um feixe de sincronização, a chegar ao SS. O Servidor de Sincronização é responsável por colocar ETRs nas unidades de mídia envolvidas na aplicação e controlar o envio destas unidades de mídia para os seus respectivos destinos, de modo a evitar que *buffers* nos destinos fiquem vazios (*starvation*) ou cheios (*overrun*).

A técnica de sincronização utilizada, descrita com maiores detalhes em [RuDu96, Rubi96], baseia-se na proposta de Rangan et al. [RaRa93, RRK95]. Portanto, assume-se que os relógios não são sincronizados e que o tempo entre a transmissão pela fonte da primeira unidade de dado desta unidade de mídia e a recepção pelo SS da última unidade de dado desta unidade de mídia está entre Δ_{min} e Δ_{max}. Define-se uma **unidade de realimentação** (UR) como sendo uma unidade elementar utilizada para controlar o envio de unidades de mídia para os seus respectivos destinos, de modo a evitar que *buffers* nos destinos fiquem vazios (*starvation*) ou cheios (*overrun*). Assim, o tempo entre a transmissão pelo destino da primeira unidade de dado de uma unidade de realimentação e a recepção pelo SS da última unidade de dado desta unidade de realimentação está entre Δ^r_{min} e Δ^r_{max}. Apesar da técnica não utilizar relógios sincronizados, as variações nas taxas de crescimento dos tempos dos relógios (*clock drift*) de diferentes estações são levadas em conta através das estimativas do menor e maior tempos de duração de uma unidade de mídia da aplicação. Sendo θ a duração nominal de uma unidade de mídia e ρ a variação máxima fracional em relação a θ, a duração real de uma unidade de mídia *n*, $\theta(n)$, estará entre $\theta \times (1-\rho)$ e $\theta \times (1+\rho)$, pois $\rho = |\theta(n) - \theta| / \theta$.

Em cada feixe de sincronização é definida uma **mídia mestra**, para a qual é garantida a continuidade. Todas as outras mídias de um feixe de sincronização são escravas. As **mídias escravas** são sincronizadas em relação à mestra, estando sujeitas a saltos (para adiantá-las) ou repetições (para atrasá-las), quando a assincronia ultrapassa um valor predeterminado. Os **saltos** correspondem à não exibição de *n* unidades de mídia, enquanto que as **repetições** correspondem à reexibição de uma mesma unidade de mídia *n* vezes. A escolha da mídia mestra depende da aplicação. No caso de uma videoconferência, prioriza-se a mídia áudio em detrimento da mídia vídeo, pois um grande intervalo entre amostras de áudio e/ou uma repetição ou perda de amostras de áudio podem prejudicar a inteligibilidade.

Dados gerados por diversas fontes em um determinado instante terão a garantia de uma apresentação simultânea em diferentes destinos. De uma forma simplificada, tem-se as fontes gerando unidades de mídia que deve ser encaminhadas aos seus respectivos destinos. O SS coloca estampas de tempo relativo nas unidades de mídia baseado no tempo de transferência estimado entre a fonte e o SS, organiza os dados que constituem um feixe de sincronização com relações temporais e garante a continuidade para a mídia mestra de um feixe de sincronização. O destino determina o tempo relativo e exibe as informações de uma forma sincronizada.

3 Camada Sincronização

3.1 Serviço de sincronização

O presente trabalho concentra as funções de sincronização em uma camada denominada Sincronização. O serviço de sincronização está especificado em duas subcamadas: continuidade (sincronização intramídia) e sincronização intermídia e é descrito com maiores detalhes nas seções 3.3 a 3.7.

Devido às características centralizadas atribuídas ao Servidor de Sincronização, as funções da camada Sincronização são assimétricas. Portanto, as funções executadas pelo SS e pelas outras estações (e conseqüentemente suas Máquinas de Estados) são distintas. A arquitetura de protocolos utilizada é apresentada na figura 2.

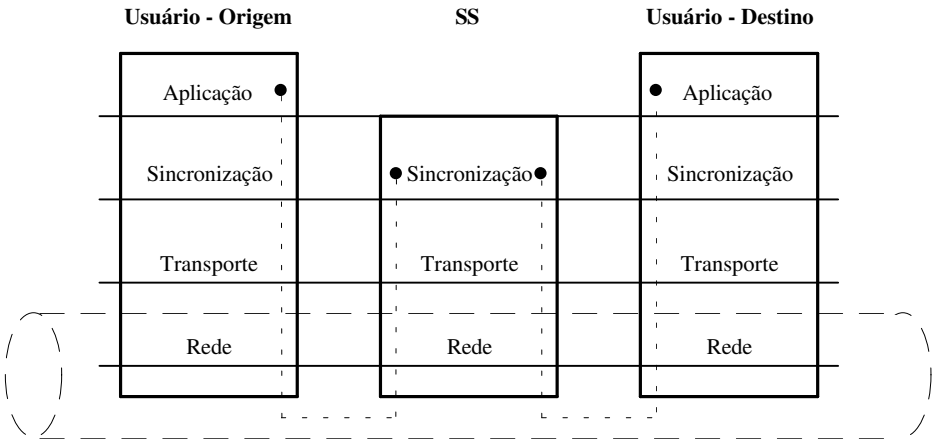

Figura 2: Arquitetura de protocolos utilizada.

Conforme descrito na seção 2, em cada feixe de sincronização é definida uma mídia mestra, para a qual é garantida a continuidade, e todas as outras mídias de um feixe de sincronização são definidas como escravas.

Cabe à subcamada continuidade oferecer um serviço de continuidade da mídia mestra, para todos os feixes de sincronização. Esta subcamada utiliza o serviço da camada Transporte que oferece um serviço de transferência de dados fim-a-fim não confiável e não orientado à conexão.

A subcamada sincronização intermídia utiliza o serviço da subcamada continuidade para prover à aplicação pares de mídias (*mestra*, *escrava*), cuja assincronia não ultrapassa um valor predeterminado $A_{\max}$, de cada feixe de sincronização. Como foi explicado na seção precedente, a idéia básica do serviço de sincronização é bastante simples pois consiste em garantir a continuidade da mídia mestra e a sincronização das mídias escravas em relação à mídia mestra, para cada feixe de sincronização.

A tabela 1 mostra as primitivas de serviço com uma breve descrição e os parâmetros de cada uma. A primitiva S-JOIN.Request contém os parâmetros de sincronização da seção 2 que são delta_min, delta_max, delta_r_min, delta_r_max, theta e ro e o parâmetro assinc que é a assincronia máxima permissível ($A_{\max}$) entre uma escrava e a mestra de um feixe de sincronização. Estes parâmetros são fornecidos pelo usuário, quando na fase de adesão a um grupo (seção 3.5). Além disto, o parâmetro opções (seção 3.2.1) também é utilizado. Todos estes parametros estão indicados na tabela 1 como parâmetros de qualidade de serviço.

Tabela 1: Primitivas de serviço de continuidade e sincronização intermídia.

Primitivas	Descrição	Parâmetros
S-CONNECT.*Request*/ S-CONNECT.*Indication*	*Pedido*/*Indicação de pedido* de estabelecimento de conexão	Endereço destino Endereço fonte Tipo de mídia
S-CONNECT.*Response*/ S-CONNECT.*Confirm*	*Resposta*/*Confirmação a*/*de pedido* de estabelecimento de conexão	Endereço respondedor Resposta
S-DISCONNECT.*Request*/ S-DISCONNECT.*Indication*	*Pedido*/*Indicação de pedido* de liberação de conexão	*Tipo*/*Razão* de desconexão
S-DATA.*Request*/ S-DATA.*Indication*	*Pedido*/*Indicação de pedido* de envio de unidade de mídia	Identificador de feixe Tipo de mídia Estampa de Tempo Unidade de realimentação Dados do usuário
S-FEEDBACKED-DATA.*Request*	*Pedido* de envio de unidade de realimentação	Identificador de feixe Estampa de Tempo
S-JOIN.*Request*	*Pedido* de adesão a um feixe de sincronização	Identificador de fluxo Identificador de feixe Qualidade de serviço
S-JOIN.*Confirm*	*Confirmação de pedido* de adesão a um feixe de sincronização	Resposta Identificador de fluxo Identificador de feixe
S-LEAVE.*Request*	*Pedido* de desligamento de um fluxo de um feixe de sincronização	Identificador de fluxo Identificador de feixe
S-ABORT.*Indication*	*Indicação de falta de recursos* para processar um feixe	Código de erro

3.2 Protocolo de sincronização

O protocolo proposto neste trabalho segue a tendência dos protocolos de alta velocidade, onde uma parte do cabeçalho das PDUs (*Protocol Data Units*) e seus campos possuem tamanhos fixos. Todos os campos, do cabeçalho e de dados, são alinhados em quatro *bytes*, de modo a aproveitar o potencial das máquinas de 32 *bits*, bem como para acelerar a decodificação dos campos das PDUs [Nune95]. Assim, todas as PDUs deste protocolo utilizam uma mesma parte fixa do cabeçalho, com três campos de 32 *bits*. Estas características visam minimizar o processamento das PDUs que se constitui na principal limitação de desempenho de um protocolo.

3.2.1 PDUs do protocolo

As PDUs são divididas nos seguintes grupos funcionais: conexão, desconexão, transferência de dados, pré-armazenamento, feixe de sincronização e erro. A tabela 2 mostra as PDUs do protocolo de sincronização separadas nos grupos funcionais com uma breve descrição de suas utilizações.

Tabela 2: Unidades de dados de protocolo de sincronização.

Grupos Funcionais	PDUs	Descrição
Conexão	**CR** (Connect Request)	Pedido de conexão ponto-a-ponto
	NCR (Numbered Connect Request)	Pedido de conexão numerada ponto-a-ponto
	NCC (Numbered Connect Confirm)	Confirmação de pedido de conexão
Desconexão	**DR** (Disconnect Request)	Pedido de desconexão ponto-a-ponto
Transferência de dados	**DT** (Data)	Envio de dados do usuário
	SD (Stamped Data)	Envio de dados com estampa de tempo relativo
	FD (Feedbacked Data)	Envio de unidade de realimentação
Pré-armazenamento	**PF** (Prefetch)	Pedido de pré-armazenamento de dados
Feixe de sincronização	**JR** (Join Request)	Pedido de adesão de um fluxo a um feixe de sincronização
	JC (Join Confirm)	Confirmação de adesão de um fluxo a um feixe de sincronização
	LR (Leave Request)	Pedido de desligamento de um fluxo de um feixe de sincronização
Erro	**ERROR**	Fornecimento de diagnósticos para diversas situações de erro

As PDUs contém uma parte fixa do cabeçalho e uma parte variável. As PDUs DT e SD são as únicas a possuir segmento de dados.

A parte fixa do cabeçalho contém os campos *command*, *key* e seq, todos de 32 *bits* (figura 3).

O campo *cmd* (*command*), conforme a figura 3, contém informações sobre o tipo da PDU (tipo), o número de *bytes* necessários para completar o campo de dados do usuário, de modo a tornar seu comprimento múltiplo de 32 *bits* (*padding*) e as opções de configuração do protocolo (opções). Há duas políticas de sincronização: conservativa e agressiva. A política conservativa atua quando a assincronia excede A_{max}, enquanto que a política agressiva atua quando há uma chance da assincronia exceder A_{max}. As opções, CON e AGR, indicam a utilização da política conservativa ou agressiva, respectivamente. Elas são usadas na PDU de pedido de adesão JR (Join Request), apresentada na seção 3.2.1.5. Os *bits* de 18 a 31 estão reservados para futuras extensões.

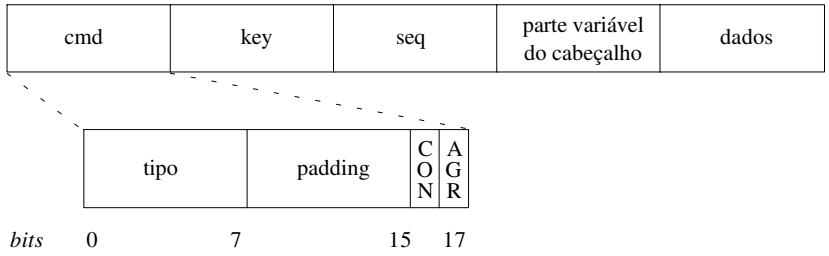

Figura 3: Formato das PDUs e do campo *command*.

O campo *key* serve para associar uma PDU a uma determinada conexão. Este campo é preenchido pelo SS, que atua como um gerente de conexões de sincronização.

O campo *seq* indica o número de seqüência de PDUs DT, SD e FD. Este campo deve conter o valor zero em quaisquer outras PDUs.

3.2.1.1 PDUs de conexão

As PDUs ***CR*** (Connect Request) e ***NCR*** (Numbered Connect Request) são utilizadas para pedido de conexão ponto-a-ponto. A PDU ***NCR*** indica no campo *key* do cabeçalho qual o número de conexão dado pelo SS. Na PDU ***CR*** o campo key contém o valor zero. A parte variável do cabeçalho de ambas é igual. Os campos *end_dest* e *end_fonte* contém, respectivamente, os identificadores lógicos das entidades respondedora e iniciadora da conexão. O campos *end_dest_SINC* e *end_fonte_SINC* contém, respectivamente, os identificadores lógicos das entidades destino final da conexão e fonte inicial da conexão. Pelo campo *end_dest_SINC*, utilizado na PDU CR, é que o SS sabe para quem enviar a PDU NCR. Já o campo *end_fonte_SINC*, preenchido na PDU NCR, é usado pelo destino final para enviar uma PDU NCC para a fonte inicial (antes porém, passando pelo SS). O campo *tipo_mídia* contém o identificador do tipo da mídia relacionado à conexão em questão.

A PDU ***NCC*** (Numbered Connect Confirm) é utilizada para confirmação de pedido de conexão ponto-a-ponto. O campo *end_resp* contém o identificador lógico da entidade respondedora da conexão. O campo *resposta* contém a resposta (aceitação ou recusa).

3.2.1.2 PDU de desconexão

A PDU ***DR*** (Disconnect Request) é utilizada para pedido de desconexão ponto-a-ponto. O campo *razão* contém a razão da desconexão.

3.2.1.3 PDUs de transferência de dados

A PDU ***DT*** (Data) é utilizada para a transmissão de uma UM para o SS. A parte variável do seu cabeçalho contém dois campos não utilizados, reservados para futuras extensões, e um campo *feixe* que indica a qual feixe o fluxo em questão pertence. O fato de não utilizar dois campos surge devido à necessidade de evitar-se uma cópia de memória no SS, quando do recebimento de uma PDU DT e o envio da correspondente PDU SD, de modo a evitar uma degradação de desempenho devido a cópia [Rubi96]. Seu segmento de dados inclui os *bytes* de enchimento que venham a ser necessários.

A PDU ***SD*** (Stamped Data) é utilizada para a transmissão de uma UM, após sua passagem pelo SS. A parte variável do seu cabeçalho contém os campos ETR e UR. O campo *ETR* contém a Estampa de Tempo Relativo da unidade de mídia. O campo *UR* habilitado indica que quando esta unidade de mídia estiver sendo exibida deve ser transmitida uma unidade de realimentação para o SS. Ambos são preenchidos pelo SS. O campo *feixe* indica a qual feixe pertence o fluxo em questão. O segmento de dados é igual ao apresentado na PDU DT.

A PDU ***FD*** (Feedbacked Data) é utilizada para a transmissão de unidade de realimentação. A parte variável do seu cabeçalho contém os campos *ETR* e *feixe*, já apresentados na PDU SD.

3.2.1.4 PDU de pré-armazenamento

A PDU ***PF*** (Prefetch) é utilizada para pedido de pré-armazenamento de UMs de um mestre no destino. A parte variável do seu cabeçalho contém os campos ident, num_UM, feixe, buffer e theta. O campo *ident* contém o identificador do fluxo mestre e o campo *num_UM* contém o número de UMs que devem ser pré-armazenadas. O campo f*eixe* indica a qual feixe pertence o fluxo em questão, o campo *buffer* indica qual o tamanho do *buffer* que deve ser alocado no destino para a recepção das UMs do fluxo em questão. O campo *theta* foi descrito na seção 2.

3.2.1.5 PDUs de feixe de sincronização

A PDU ***JR*** (Join Request) é utilizada para pedido de adesão a um feixe de sincronização. A parte variável do seu cabeçalho contém os campos ident, feixe, assinc, delta_min, delta_max, delta_r_min, delta_r_max, theta e ro. O campo *ident* contém o identificador do fluxo e o campo *feixe* contém o identificador do feixe de sincronização. O campo *assinc* contém o valor máximo da assincronia permissível entre o escravo e o mestre, dentro de um feixe de sincronização. Este campo contém o valor zero, no caso do fluxo que está sendo adicionado ao feixe de sincronização ser mestre. Os campos *delta_min*, *delta_max*, *delta_r_min*, *delta_r_max*, *theta* e *ro* foram descritos na seção 2.

A PDU ***JC*** (Join Confirm) é utilizada para confirmação de pedido de adesão de um fluxo a um feixe de sincronização. A parte variável do seu cabeçalho contém os campos *resposta* (aceitação ou recusa), *ident* e *feixe*.

A PDU ***LR*** (Leave Request) é utilizada para pedido de desligamento de um fluxo de um feixe de sincronização. A parte variável do seu cabeçalho contém os campos *ident* que é o identificador do fluxo e *feixe* que contém o identificador do feixe de sincronização ao qual pertence o fluxo em questão.

3.2.1.6 PDUs de erro

A PDU ***ERROR*** é utilizada para fornecer diagnósticos para diversas situações de erro do protocolo. A parte variável do seu cabeçalho contém o campo *cod_erro* que indica qual o código do erro ocorrido, segundo a tabela 3.

Tabela 3: Códigos de erro da PDU ERROR.

cod_erro	Condições de erro
01	Pedido de conexão recusado pelo SS
02	Pedido de conexão recusado pela entidade respondedora
03	Pedido de adesão recusado
04	Pedido de mudança de mestre recusado

Nas seções seguintes são apresentadas diversas fases do serviço e do protocolo de sincronização. As tabelas de estados do protocolo de sincronização e os pseudocódigos relativos a estas fases podem ser encontrados em [Rubi96].

3.3 Abertura de conexão

Quando uma estação deseja conectar-se a outra, o usuário do serviço deve fornecer o nome da estação remota e enviar uma primitiva S-CONNECT.Request, desencadeando o envio de uma PDU CR para o SS. Quando o SS recebe a PDU CR, este pode obter ou não um número de conexão livre. Caso não obtenha um número de conexão livre, o SS envia uma PDU ERROR com cod_erro = 1 para a entidade iniciadora da conexão que, ao recebê-la, gera uma primitiva S-DISCONNECT.Indication. Caso contrário, o SS envia uma PDU NCR para a entidade destinatária. Na entidade destinatária, a recepção da PDU NCR gera uma primitiva S-CONNECT.Indication. A entidade respondedora pode aceitar ou não a conexão. A entidade respondedora, aceitando a conexão, gera uma primitiva S-CONNECT.Response e envia uma PDU NCC para o SS. O SS, ao receber a PDU NCC, envia-a à entidade iniciadora que, ao recebê-la, envia uma primitiva S-CONNECT.Confirm ao usuário, abrindo definitivamente a conexão. Caso a entidade respondedora não aceite a conexão, uma primitiva S-CONNECT.Response indicará a recusa da conexão, causando o envio de uma PDU ERROR com cod_erro = 2 ao SS. O SS ao receber a PDU ERROR, envia-a à entidade iniciadora da conexão, que ao recebê-la, envia uma primitiva S-DISCONNECT.Indication para o usuário, fechando a conexão.

3.4 Fechamento de conexão

Qualquer uma das entidades envolvidas em uma conexão ponto-a-ponto pode decidir terminá-la, exceto o SS. Em uma desconexão normal, uma primitiva S-DISCONNECT.Request é gerada, causando o envio de uma PDU DR ao SS, fechando a conexão. Quando o SS recebe a PDU DR, este repassa a PDU à entidade par. Na entidade par, a recepção da PDU DR faz com que seja enviada uma primitiva S-DISCONNECT.Indication ao usuário, fechando definitivamente a conexão.

3.5 Adesão a feixe

Quando a estação iniciadora da conexão quer adicionar o fluxo a um feixe, a estação pede um determinado número de parâmetros ao usuário. Os parâmetros são diferentes para o caso da mídia correspondente à conexão ser mestra ou escrava. No caso da mídia ser mestra, os parâmetros são o feixe, os atrasos mínimo e máximo de unidades de realimentação, o theta, o ro e os atrasos mínimo e máximo de unidades de mídia (seção 3.2.1.5). No caso da mídia ser escrava, os parâmetros são o feixe, a assincronia máxima permissível entre esta mídia e a mídia mestra e os atrasos mínimo e máximo de unidades de mídia (seção 3.2.1.5). Após a configuração destes parâmetros, uma primitiva S-JOIN.Request é gerada, causando o envio de uma PDU JR ao SS. Quando o SS recebe a PDU JR, caso aceite a adesão ao feixe, este envia uma PDU JC à entidade que pediu a adesão ao feixe e, se a mídia for mestra, o SS calcula o número de unidades de mídia a serem pré-armazenadas e a taxa de realimentação [RuDu96, Rubi96] e envia uma PDU PF à entidade destinatária. Na entidade que pediu a adesão, a recepção da PDU JC faz com que seja enviada uma primitiva S-JOIN.Confirm ao usuário. Na entidade par, a recepção da PDU PF faz com que a variável *Espera_Pre_Armazenamento*[*feixe*] receba VERDADEIRO e que a fase de transmissão de dados esteja aberta. Caso o SS não aceite a adesão ao feixe, é enviada uma PDU ERROR com cod_erro = 3. A entidade que pediu a adesão ao feixe, ao receber a PDU ERROR, envia uma primitiva S-ABORT.Indication ao usuário.

3.6 Desligamento de feixe

Quando a estação iniciadora da conexão quer desligar o fluxo de um feixe, a estação gera uma primitiva S-LEAVE.Request, causando o envio de uma PDU LR ao SS. Quando o SS recebe a PDU LR, o fluxo é retirado do feixe.

3.7 Transferência de dados

A primitiva S-DATA.Request é utilizada pelo usuário para a transferência de dados. Os dados são transmitidos ao SS através de PDUs DT seqüencialmente numeradas (começando em um). O SS, ao receber uma PDU DT, verifica se esta é relativa a uma mídia mestra ou escrava. No caso da PDU ser relativa a uma mídia mestra, o SS atribui ao campo ETR o número de seqüência da PDU DT - gerando uma PDU SD - e verifica se para esta PDU será necessário o envio de uma UR, quando da exibição da mesma no destino, habilitando ou não o campo *UR* (seção 3.2.1.3). Após isto, no caso de ter recebido uma UR, o SS calcula o menor e o maior tempos de transmissão da UM em questão, enviando a PDU SD neste intervalo de tempo. Caso ainda não tenha recebido uma UR, o SS envia a PDU SD para o destino imediatamente. No caso da PDU ser relativa a uma mídia escrava, o SS calcula a ETR da UM em questão e envia a PDU SD para o destino. Na figura 4, a recepção da PDU DT pelo SS é descrita em SDL (*Specification and Description Language*). A estação remota, ao receber uma PDU SD, verifica o valor da variável booleana *Espera_Pre_Armazenamento*[*feixe*]. Se a variável for falsa o tempo de exibição da UM é calculado e ao estourar o temporizador correspondente, uma primitiva S-DATA.Indication é entregue ao usuário. Se a variável for verdadeira, a criação dos temporizadores só começará após o recebimento da P-ésima UM [Rubi96]. Cada temporizador que estourar gerará uma primitiva S-DATA.Indication. Na figura 5, a recepção da PDU SD pelo destino é descrita em SDL. O usuário, ao receber a primitiva S-DATA.Indication, verifica se o parâmetro UR da primitiva é verdadeiro, neste caso gerando uma primitiva S-FEEDBACKED-DATA.Request e transmitindo uma PDU FD ao SS. O SS, ao receber a PDU FD, atualiza o tempo de chegada e o número de seqüência da última UR, valores utilizados no cálculo do intervalo de transmissão de UMs [RuDu96, Rubi96].

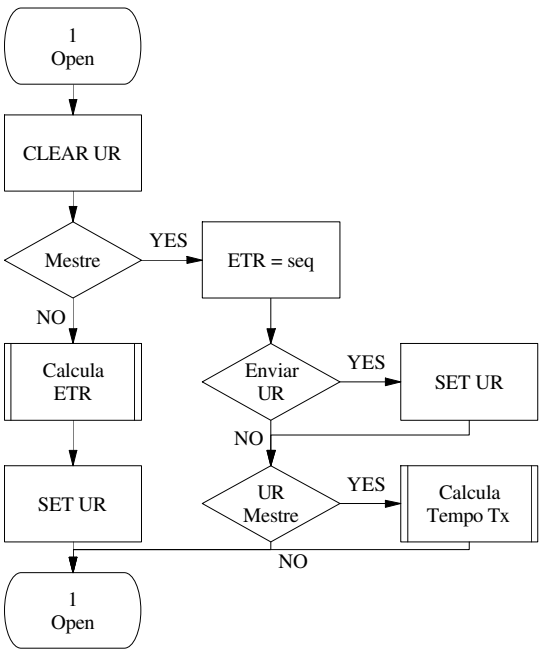

Figura 4: Diagrama em SDL da recepção da PDU DT pelo SS.

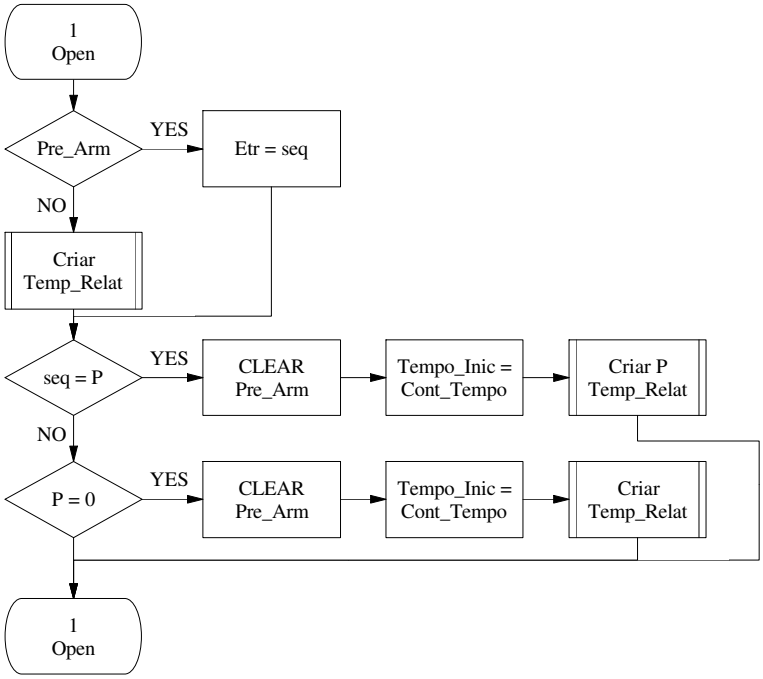

Figura 5: Diagrama em SDL da recepção da PDU SD pelo destino.

4 Implementação

O protocolo apresentado foi implementado utilizando-se um ambiente de desenvolvimento de sistemas de comunicação de alto desempenho com garantias de qualidade de serviço denominado GTA/MM (GTA/MultiMídia) [Albu95]. A Arquitetura GTA/MM é uma evolução da Arquitetura GTA/OSI [Bagi92], elaborada para o desenvolvimento de sistemas de comunicação OSI, e introduz mecanismos apropriados aos sistemas de comunicação em tempo real com garantias de qualidade de serviço. Além disto, são oferecidos alguns módulos específicos, opcionais, que definem as características do sistema de comunicação de alta velocidade a ser desenvolvido, tais como os módulos que implementam mecanismos de controle de tráfego e policiamento, os módulos de garantia de qualidade de serviço e outros. Por ser uma estrutura modular e flexível, é possível escolher e configurar os módulos de acordo com as características do sistema desejado, bem como criar novos módulos. A arquitetura definida para este trabalho utiliza a camada de protocolo Sincronização e os módulos de escalonamento de tarefas com prioridade, de gerenciamento de memória e de gerenciamento de temporizações.

Cada módulo possui diversas funções, com o objetivo de tornar o sistema bastante flexível. No Módulo de Gerenciamento de Memória (GM) são oferecidas funções de alocação e liberação de estruturas de dados inerentes à arquitetura, tais como estruturas de primitivas e de parâmetros, blocos de dados de tamanho fixo e dados de tamanho variável. O Módulo de Escalonamento de Tarefas (ET) possui funções para inserir, executar e retirar tarefas de baixa, média e alta prioridade, para atender aos dados que possuem restrições temporais. O Módulo de Gerenciamento de Temporizações (GT), de grande utilidade em sistemas de comunicação em tempo real, oferece funções para iniciar, criar, interromper, estourar, reiniciar e finalizar temporizadores.

A comunicação entre as camadas é feita de forma assíncrona e as interfaces entre camadas consistem de um par de filas (uma para cada sentido de transmissão) por conexão, exceto a interface SINC_T (Sincronização-Transporte) que não possui a fila SINC_T. Portanto, a comunicação Sincronização-Transporte é feita diretamente, através de chamada de procedimentos. O esquema da arquitetura proposta é apresentado na figura 6.

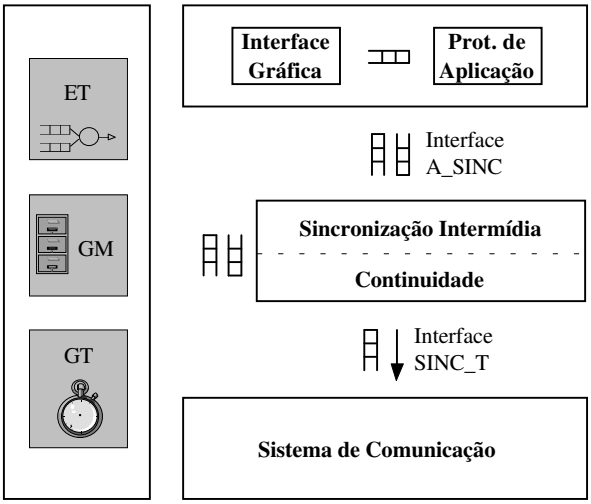

Figura 6: Arquitetura de implementação.

Foram feitas algumas modificações nos módulos de Escalonamento de Tarefas e Gerenciamento de Temporizações (denominados versões 3.0 [Rubi96]) da Arquitetura GTA/MM de modo a permitir a implementação das funcionalidades propostas pelo serviço de sincronização e torná-los mais flexíveis.

4.1 Ambiente de desenvolvimento

O ambiente de desenvolvimento compreende estações de trabalho SUN-SPARC, interconectadas através de uma rede Ethernet, com placa e câmera para aquisição de vídeo e um sistema para captura e reprodução de áudio. A linguagem de programação utilizada é o C e o sistema operacional Solaris 2.4.

4.1.1 Aquisição e exibição de vídeo

A placa utilizada para a aquisição de vídeo é a SunVideo [Sun94a]. Esta placa permite a captura de até 30 quadros/s e compressão por *hardware* nos formatos de vídeo MPEG-1, JPEG, CellB e UYVY. A descompressão de dados de vídeo é feita por *software*. Um pacote de desenvolvimento de aplicações (*Software Developer's Kit* - SDK) utilizando a placa SunVideo e uma biblioteca chamada XIL *Imaging Library* [Sun94b], contendo funções para a captura, a compressão e o escalamento de quadros, dão suporte ao desenvolvimento das aplicações.

As seguintes funções foram criadas para manipular vídeo ao vivo:

- **Inicializar_Video_Live_[Tx][Rx] ( )**: inicializam o dispositivo de vídeo e realizam todos os procedimentos de inicialização relacionados com a captura/apresentação e a compressão/descompressão de quadros de vídeo para a utilização da biblioteca XIL;

- **Finalizar_Video_Live_[Tx][Rx] ( )**: finalizam o dispositivo de vídeo e realizam todos os procedimentos de finalização relacionados com a captura/apresentação e a compressão/descompressão de quadros de vídeo;
- **Obter_Video_Live (Buffer)**: obtém os dados da placa de vídeo, relativos a um quadro de vídeo, e coloca-os no **Buffer**;
- **Apresentar_Video_Live (Buffer, Quadro, Tamanho)**: exibe um quadro de vídeo no dispositivo de saída de vídeo, tendo como parâmetros o ponteiro do **Buffer** dos dados relativos ao quadro, o número do **Quadro** e o **Tamanho** em *bytes* do mesmo.

4.1.2 Aquisição e exibição de áudio

O áudio é capturado ao vivo a uma taxa de 8000 amostras/s. A lei μ, padrão para dados de voz utilizado nos EUA, Canadá e Japão, é utilizada como lei de compressão, reduzindo o tamanho de uma amostra de 12 *bits* para 8 *bits*. Com isto, a taxa de captura de áudio é de 64 kbps.

As seguintes funções foram criadas para manipular áudio ao vivo:

- **Inicializar_Audio_Live_[Tx][Rx] ( )**: inicializam o dispositivo de áudio para leitura/escrita somente, especificando a taxa de captura, número de *bytes* por amostra de áudio, número de canais, tipo da codificação e outros;
- **Finalizar_Audio_Live_[Tx][Rx] ( )**: fecham o dispositivo de áudio. Estas rotinas são executadas no final do programa;
- **Obter_Audio_Live (Tamanho, Buffer)**: retorna **Tamanho** *bytes* de áudio a partir de **Buffer**;
- **Apresentar_Audio_Live (Tamanho, Buffer)**: apresenta **Tamanho** *bytes* de áudio a partir de **Buffer**.

4.2 Sistema implementado

O sistema implementado divide-se em: programa principal, módulos de áudio e vídeo e camadas Aplicação, Sincronização e Transporte. A implementação dos módulos de áudio e vídeo foi descrita na seção anterior.

4.2.1 Programa principal

O programa principal está dividido em três partes principais: inicialização, execução e finalização.

Na inicialização, o usuário deve informar se a sua estação é o SS ou não. São feitas as inicializações dos módulos globais GM, ET e GT e das interfaces entre as camadas A_SINC e SINC_T. Caso não seja o Servidor, é obtido o nome do SS. Após isto, caso o UDP (seção 4.2.4) não tenha sido inicializado para a porta do SS, isto é feito. Além disto, se não for o servidor, é obtido o nome da estação fonte.

Na execução, é feita uma repetição visando verificar se há alguma tarefa a ser executada pelo Escalonador de Tarefas e se chegou algo em alguma das portas relacionadas a cada uma das mídias e ao controle [Rubi96]. Esta repetição acaba quando o usuário aperta o botão **Fim** (seção 4.2.2.1).

Na finalização, as principais ações realizadas são a devolução ao sistema de toda a memória alocada e o fechamento dos dispositivos de áudio e vídeo.

4.2.2 Aplicação

A Aplicação está dividida em interface gráfica e protocolo de Aplicação, que se comunicam através de uma fila. Somente o protocolo de Aplicação se comunica com a camada Sincronização, conforme a figura 6.

4.2.2.1 Interface gráfica

A interface gráfica foi criada utilizando-se a ferramenta de auxílio à confecção de janelas Xview. Os botões da interface são: **Destino**, **Conectar**, **Transmitir** e **Fim**. Estes botões chamam funções correspondentes do protocolo de Aplicação.

4.2.2.2 Protocolo de Aplicação

A Máquina de Estados SINC_APLIC foi implementada de acordo com as tabelas de estados apresentadas em [Rubi96]. Outras funções de destaque são: Executar_Usuário, Executa_Audio e Obter_Destino.

A função *Executar_Usuario* é chamada quando o botão **Transmitir** é apertado (seção 4.2.2.1). Neste caso, é feita a inicialização do áudio para a transmissão e é criado um temporizador auto-disparável que chama a função Executa_Audio a cada theta[feixe] / PRECISAO unidades de tempo.

Na função *Executa_Audio* é feita a alocação de memória e uma amostra de áudio é obtida através da função Obter_Audio_Live (seção 4.1.2). O tamanho desta amostra é especificado pela constante TAM_MAX_QUADRO_AUDIO, encontrada em um arquivo de constantes de sincronização.

A função *Obter_Destino* é chamada quando o botão **Destino** é apertado (seção 4.2.2.1). Neste caso, o nome da estação destino é obtido através de interação com o usuário.

4.2.3 Sincronização

Os pseudocodigos das funções Máquina de Estados APLIC_SINC e Máquina de Estados TRAN_SINC, são apresentados em [Rubi96].

4.2.4 Sistema de comunicação

O sistema de comunicação utiliza os protocolos UDP/IP, de modo a permitir uma comunicação entre estações. A escolha do UDP deve-se à grande utilização de áudio em aplicações multimídias e ao fato de não ser usual a utilização de retransmissões e reconhecimentos para fluxos isócronos.

O acesso ao UDP foi feito através de *sockets* [Naum93]. A camada Transporte corresponde à implementação do sistema de comunicação. Nesta camada estão definidas uma série de rotinas de acesso ao UDP, dentre as quais, destacam-se:

- **Inicializar_UDP (Midia, Tamanho)**: cria um soquete, nomeia-o usando *wildcards*, especifica metade da associação do soquete, altera o tamanho do buffer de recepção se **Tamanho** for maior que zero e faz a variável UDP_Inicializado[**Midia**] receber VERDADEIRO;
- **Finalizar_UDP (Midia)**: faz UDP_Inicializado[**Midia**] receber FALSO e fecha o soquete correspondente à **Midia**;
- **Setar_Destino (Midia)**: finaliza a inicialização do soquete correspondente à **Midia**;
- **Trata_Porta ( )**: utiliza a função *Obter_Porta_Tipo_Midia* para futuramente receber ou transmitir a mídia em questão e inicializa o soquete correspondente a esta mídia;
- **Obter_Porta_Tipo_Midia ( )**: obtém um número de porta de um arquivo que contém os números de portas livres;
- **Maquina_Estados_SINC_TRAN (Primitiva, Midia)**: envia o tamanho em *bytes* e os próprios dados através do soquete correspondente à **Midia** e libera a memória alocada na camada Aplicação;
- **UDP_Rx (Midia)**: aloca memória para receber os dados (TAM_MAX_QUADRO), recebe o tamanho em *bytes* e os próprios dados. Após isto, envia uma primitiva T-DATA.Indication para a camada Sincronização.

Os pseudocodigos das funções Trata_Porta, Maquina_Estados_SINC_TRAN e UDP_Rx são apresentados em [Rubi96].

O primeiro protótipo deste sistema, constituído de aproximadamente 9000 linhas de código, encontra-se em funcionamento no Laboratório do Grupo de Teleinformática e Automação da COPPE/UFRJ.

5 Conclusões

Este trabalho apresentou um serviço de continuidade e de sincronização intermídia apropriado para aplicações multimídias que envolvam fontes de dados armazenados ou ao vivo, relações temporais implícitas, múltiplas fontes e destinos e não utilizem relógio global. O serviço provê a sincronização intramídia para as mídias mestras de todos os feixes de sincronização e a sincronização intermídia, para todos os feixes de sincronização.

O serviço é definido através das primitivas de serviço e o seu respectivo protocolo é especificado através dos tipos e formatos das unidades de dados de protocolo e por suas máquinas de estados responsáveis pela execução das funções de sincronização. Com a finalidade de diminuir tempo de processamento do protocolo, uma parte do cabeçalho das PDUs (*Protocol Data Units*) possui tamanho fixo e todos os campos (do cabeçalho e de dados) são alinhados em 32 *bits*.

A implementação do protocolo utiliza um ambiente de desenvolvimento de sistemas de comunicação de alto desempenho com garantias de qualidade de serviço denominado Arquitetura GTA/MM. As principais características desta arquitetura são:

- comunicação assíncrona entre as camadas, modeladas por filas. O resultado é um alto grau de modularidade e um fraco acoplamento entre as camadas;

- um gerenciamento de memória que oferece funções de alocação e liberação de estruturas de dados inerentes à arquitetura, tais como estruturas de primitivas e de parâmetros, blocos de dados de tamanho fixo e dados de tamanho variável;
- um escalonamento de tarefas que possui funções para inserir, executar e retirar tarefas de baixa, média e alta prioridade, para atender aos dados que possuem restrições temporais;
- um gerenciamento de temporizações que oferece funções para iniciar, criar, interromper, estourar, reiniciar e finalizar temporizadores e que inclui funcionalidades requeridas pelo mecanismo de sincronização utilizado.

O sistema de comunicação foi implementado na linguagem C e roda em SUN-SPARC, com sistema operacional Solaris 2.4, ligadas por uma rede Ethernet. As estações envolvidas na aplicação possuem placa e câmera para aquisição de vídeo e um sistema para captura e reprodução de áudio. O sistema de comunicação implementado utilizou os protocolos UDP/IP, de modo a permitir uma comunicação entre estações.

A continuação deste trabalho visa o uso do protocolo IPv6 e do protocolo de reserva de recursos RSVP (*resource ReSerVation Protocol*), e também de um protocolo de transporte multidestinatário. Está prevista também a migração de todo o sistema para um ambiente de rede ATM.

Referências

[Albu95] C. V. N. Albuquerque, "Arquitetura de Implementação de Alto Desempenho para Sistemas de Comunicação Multimídia", *Tese de Mestrado*, PEE-COPPE/UFRJ, abril de 1995.

[AnHo91] D. P. Anderson e G. Homsy, "A Continuous Media I/O Server and Its Synchronization Mechanism", *IEEE Computer*, pp. 51-57, outubro de 1991.

[Bagi92] L. F. Baginski, "Ambiente de Implementação para Sistemas de Alto Desempenho", *Tese de Mestrado*, PEE-COPPE/UFRJ, janeiro de 1992.

[BlSt96] G. Blakowski e R. Steinmetz, "A Media Synchronization Survey: Reference Model, Specification, and Case Studies", *IEEE Journal on Selected Areas in Communications*, vol. 14, nº 1, pp. 5-35, janeiro de 1996.

[CaCo96] L. F. R. C. Carmo e J.-P. Courtiat, "Extrator de Dependências Condicionais para Sistemas de Comunicação Multimídias", *XIV Simpósio Brasileiro de Redes de Computadores*, Fortaleza, pp. 56-73, 20 a 23 de maio de 1996.

[EFI94] L. Ehley, B. Furht e M. Ilyas, "Evaluation of Multimedia Synchronization Techniques", *Proceedings of the International Conference on Multimedia Computing and Systems*, pp. 514-519, 14 a 19 de maio de 1994.

[EPD94] J. Escobar, C. Partridge e D. Deutsch, "Flow Synchronization Protocol", *IEEE/ACM Transactions on Networking*, vol. 2, nº 2, pp. 111-121, abril de 1994.

[Ferr92] D. Ferrari, "Delay Jitter Control Scheme for Packet-switching Internetworks", *Computer Communications*, vol. 15, nº 6, pp. 367-373, julho/agosto de 1992.

[HoSt93] F. Horn e J. B. Stefani, "On Programming and Supporting Multimedia Object Synchronization", *The Computer Journal*, vol. 36, nº 1, pp. 4-18, fevereiro de 1993.

[ITU-T94] International Telecommunications Union - Technical section, "Functional Specification and Description Language SDL", *ITU-T Recommendation Z.100*, junho de 1994.

[LaGe94] L. Lamont e N. D. Georganas, "Synchonization Architecture and Protocols for a Multimedia News Service Application", *Proceedings of the International*

Conference on Multimedia Computing and Systems, pp. 3-8, 14 a 19 de maio de 1994.

[LiGh91] T. D. C. Little e A. Ghafoor, "Multimedia Synchronization Protocols for Broadband Integrated Services", *IEEE Journal on Selected Areas in Communications*, vol. 9, nº 9, pp. 1368-1382, dezembro de 1991.

[Naum93] C. M. Naumann, "Bibliotecas de Socket (Berkeley)", *Apostila*, NCE/UFRJ, 1ª edição, agosto de 1993.

[Nico90] C. Nicolaou, "An Architecture for Real-Time Multimedia Communication Systems", *IEEE Journal on Selected Areas in Communications*, vol. 8, nº 3, pp. 391-400, abril de 1990.

[Nune95] M. D. Nunes, "Protocolo Multidestinatário com Configuração de Facilidades e Gerenciamento de Acesso", *Tese de Mestrado*, PEE-COPPE/UFRJ, abril de 1995.

[RaRa93] S. Ramanathan e P. V. Rangan, "Feedback Techniques for Intra-Media Continuity and Inter-Media Synchronization in Distributed Multimedia Systems", *The Computer Journal*, vol. 36, nº 1, pp. 19-31, fevereiro de 1993.

[RRK95] P. V. Rangan, S. Ramanathan e T. Kaeppner, "Performance of Inter-Media Synchronization in Distributed and Heterogeneous Multimedia Systems", *Computer Networks and IDSN Systems*, vol. 27, pp. 549-565, 1995.

[Rubi96] M. G. Rubinstein, "Um Serviço de Sincronização Contínua para Aplicações Multimídias", *Tese de Mestrado*, PEE-COPPE/UFRJ, novembro de 1996.

[RuDu96] M. G. Rubinstein e O. C. M. B. Duarte, "Definição de um Serviço de Sincronização Contínua para Aplicações Multimídias", *XIV Simpósio Brasileiro de Redes de Computadores*, Fortaleza, pp. 38-55, 20 a 23 de maio de 1996.

[RVR92] P. V. Rangan, H. Vin e S. Ramanathan, "Designing and On-Demand Multimedia Service", *IEEE Communications Magazine*, vol. 30, nº 7, pp. 56-65, julho de 1992.

[SHGC92] D. Shepherd, D. Hutchinson, F. Garcia e G. Coulson, "Protocol Support for Distributed Multimedia Applications", *Computer Communications*, vol. 15, nº 6, pp. 359-366, julho/agosto de 1992.

[SoAg93] S. H. Son e N. Agarwal, "Synchronization of Temporal Constructs in Distributed Multimedia Systems with Controlled Accuracy", *Relatório Técnico do Departamento de Ciência da Computação da Universidade de Virgínia*, EUA, 1993.

[Stei96] R. Steinmetz, "Human Perception of Jitter and Media Synchronization", *IEEE Journal on Selected Areas in Communications*, vol. 14, nº 1, pp. 61-72, janeiro de 1996.

[Sun94a] Sun Microsystems Inc., "SunVideo User's Guide", 1994.

[Sun94b] Sun Microsystems Inc., "XIL Programmer's Guide", 1994.

[WQG94] M. Woo, N. U. Qazi e A. Ghafoor, "A Synchronization Framework for Communication of Pre-orchestrated Multimedia Information", *IEEE Network*, pp. 52-61, janeiro/fevereiro de 1994.