UNIVERSIDADE DE BRASILIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROPOSTA DE ARQUITETURA ADAPTATIVA PARA TRANSMISSÃO MULTIDESTINATÁRIA E AO VIVO DE VÍDEO ESCALÁVEL EM REDE PAR-A-PAR

BERNARDO VERGNE DIAS

ORIENTADOR: PAULO ROBERTO DE LIRA GONDIM

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO: PPGEE – 507/2012
BRASÍLIA/DF, OUTUBRO – 2012

UNIVERSIDADE DE BRASILIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROPOSTA DE ARQUITETURA ADAPTATIVA PARA TRANSMISSÃO MULTIDESTINATÁRIA E AO VIVO DE VÍDEO ESCALÁVEL EM REDE PAR-A-PAR

BERNARDO VERGNE DIAS

DISSERTAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISÍTOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM TELECOMUNICAÇÕES.

APROVADA POR:

_____

Prof. Paulo Roberto de Lira Gondim, PhD (ENE-UnB)
(Orientador)

_____

Prof. Marcelo Menezes de Carvalho, PhD (ENE-UnB)
(Examinador Interno)

_____

Prof. Cláudio de Castro Monteiro, D.C. (IFTO)
(Examinador Externo)

BRASÍLIA/DF, 26 DE OUTUBRO DE 2012

FICHA CATALOGRÁFICA

DIAS, BERNARDO VERGNE
Proposta de Arquitetura Adaptativa para Transmissão Multidestinatária e Ao Vivo de Vídeo Escalável em Rede Par-a-Par [Distrito Federal] 2012.
xvi, 128p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2012).
Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia. Departamento de Engenharia Elétrica.
1. Redes Par-a-Par 2. Vídeo Escalável
3. Adaptação de Conteúdo 4. Controle de Congestionamento
5. Qualidade de Experiência
I. ENE/FT/UnB II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

DIAS, B. V. (2012). Proposta de Arquitetura Adaptativa para Transmissão Multidestinatária e Ao Vivo de Vídeo Escalável em Rede Par-a-Par. Dissertação de Mestrado em Telecomunicações, Publicação PPGEE – 507/2012, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 128p.

CESSÃO DE DIREITOS

AUTOR: Bernardo Vergne Dias.
TÍTULO: Proposta de Arquitetura Adaptativa para Transmissão Multidestinatária e Ao Vivo de Vídeo Escalável em Rede Par-a-Par.
GRAU: Mestre ANO: 2012

_____

Bernardo Vergne Dias
Rua 4 Note, Lote 6, Apto nº 705, Ed. Lumini. Águas Claras.
71.907-540 Brasília – DF – Brasil.

Dedicatória

Aos meus pais, Bernardino e Maria Auxiliadora, por terem sido meu alicerce emocional e de inspiração durante toda a jornada de minha carreira acadêmica. Os esforços imensuráveis e toda a dedicação deles me permitiram chegar onde cheguei e poder estar agora concluindo mais uma importante etapa da minha vida.

Bernardo Vergne Dias

Agradecimentos

Agradeço...

Em primeira instância ao professor Paulo Roberto de Lira Gondim pela oportunidade, atenção, paciência e orientação que me foram depositadas durante os vários anos de convivência.

Aos funcionários do Departamento de Engenharia Elétrica (ENE) e, principalmente, aos membros do Laboratório de Engenharia de Redes, pela tão calorosa receptividade, fiel amizade e pelos momentos de alegria.

Ao meu amigo Magno Batista Correa por sua disposição e colaboração na ideialização do esquema de compressão vídeo em blocos.

E agradeço, especialmente, aos meus pais que me foram inspiração, do começo ao fim, possibilitando a realização dessa conquista.

Bernardo Vergne Dias

RESUMO

PROPOSTA DE ARQUITETURA ADAPTATIVA PARA TRANSMISSÃO MULTIDESTINATÁRIA E AO VIVO DE VÍDEO ESCALÁVEL EM REDE PAR-A-PAR

A distribuição de fluxo contínuo de vídeo (*video streaming*) é, atualmente, uma das aplicações de maior sucesso na Internet, sendo as redes par-a-par (P2P, *Peer-to-Peer*), baseadas em princípios como a escalabilidade e o compartilhamento de recursos, uma das opções importantes para contribuir para a solução de diversos gargalos decorrentes do emprego da tradicional arquitetura cliente-servidor, comumente presente na Internet.

Por outro lado, esquemas de codificação de vídeo têm sido recentemente propostos, que permitem o ajuste entre a disponibilidade de recursos (banda) de rede e a quantidade de informação (camadas) sendo enviada para transmissão. Dentre tais esquemas, destaca-se o H.264 SVC (*Scalable Video Coding*), normatizado pela ITU-T (*International Telecommunication Union*).

Dessa forma, o trabalho inicia-se com a apresentação de conceitos relativos à codificação escalável de vídeo bem como sobre alternativas relativas à distribuição de vídeo sobre redes P2P. Adicionalmente, o trabalho discute alternativas de protocolos de controle de congestionamento multidestinatário, considerando-se a importância desse controle para a distribuição de sequencias de vídeo e o tratamento de tráfego de tempo real na Internet.

Uma arquitetura adaptativa para transmissão multidestinatária e ao vivo de vídeo escalável em rede P2P é então proposta, que se baseia, inicialmente, na definição de um paradigma de compressão de vídeo em blocos independentes, de curto intervalo de tempo, contendo uma ou algumas unidades completas de GOP (*Group of Pictures*). Para esse fim, os blocos são codificados no padrão H.264 SVC, e dotados das três dimensões de escalabilidade – temporal, espacial e de qualidade, sendo tal padrão adotado como forma de prover adaptabilidade.

É apresentado, em seguida, um formato para encapsulamento dos blocos de vídeo em pacotes que serão transmitidos com emprego do protocolo RTP (*Real-Time Protocol*). A transmissão dar-se-á via rede lógica de sobreposição (*overlay*) organizada em árvore (*Tree-based*) valendo-se da estratégia de comunicação *Push*. O *Scribe* – protocolo descentralizado e escalável construído sobre uma rede em malha *Pastry* – é utilizado para criação de grupos multidestinatários (*Multicast*) na topologia árvore.

O RTP é estendido para embarcar informações do protocolo de controle de congestionamento TFMCC (*TCP-Friendly Multicast Congestion Control*), o qual auxiliará o processo de ajuste dinâmico da qualidade de vídeo dos blocos SVC, compondo então, esse esquema, uma estratégia de CAT (*Content Adaptation Technique*). O TFMCC alimenta-se de parâmetros de QoS (*Quality of Service*), que serão mensurados pelos nós participantes da rede de transmissão do vídeo; além disso, este protocolo requer sinalização de retorno (*feedback*), que ocorrerá por canal de comunicação inverso, adicionado na rede *Scribe*. A literatura mostra que o TFMCC, especialmente em função de seu comportamento justo e moderadamente agressivo, se adequa bem no contexto de transmissão contínua de vídeo considerado nesta obra.

A proposta inclui também o emprego de técnica de retransmissão da camada base de forma a poder compensar perdas de bloco buscando assegurar uma quantidade mínima de informação para reprodução de vídeo.

A arquitetura proposta visa oferecer ao usuário melhor qualidade de experiência em transmissões ao vivo e de grande escala, sendo avaliada, a título preliminar, pela realização de testes em diferentes cenários e com base em diferentes sequências de vídeo.

ABSTRACT

AN ADAPTIVE ARCHITECTURE FOR LIVE STREAMING OF SCALABLE VIDEO OVER P2P NETWORK

Video streaming is currently one of the most successful applications on the Internet and P2P (Peer-to-Peer) network, based on scalability and resource sharing principles, is one of the important options to contribute to solve many bottlenecks that result from the use of traditional client-server architecture, commonly present on the Internet.

Furthermore, video encoding schemes have been recently proposed to allow adjustment between the availability of resources (bandwidth) of the network and the amount of information (layers) being sent. Among these schemes, we highlight the H.264 SVC (Scalable Video Coding), standardized by ITU-T (International Telecommunication Union).

In this way, the work begins with the presentation of concepts of scalable video coding as well as alternatives for distribution of video over P2P networks. Additionally, this work discusses alternatives of multicast congestion control protocols, considering the importance of this control for the distribution of video sequences and processing of real-time traffic on the Internet.

An adaptive architecture for live streaming of scalable video over P2P network is then proposed, which is based initially on the definition of a paradigm of video compression based on independent compressed blocks of short time interval, containing one or more GOP (Group of Pictures). To this end, the blocks are encoded in H.264 SVC (Scalable Video Coding) endowed with the three dimensions of scalability - temporal, spatial and quality. Such standard was adopted in order to provide adaptability.

It is presented a format for encapsulation of video blocks for transmission over RTP (Real-Time Protocol). A tree-based overlay network is used for content distribution according to the Push strategy. The Scribe - a scalable application-level multicast infrastructure – which is built on top of Pastry – a decentralized and large-scale mesh peer-to-peer system - is used to create the multicast groups in a tree topology.

The RTP is extended to support piggybacking of TFMCC (TCP-Friendly Multicast Congestion Control) protocol, which will assist the process of dynamic adjustment of video quality of the SVC blocks. This overall scheme composes a CAT (Content Adaptation Technique). The TFMCC takes as input a set of QoS (Quality of Service) parameters, which will be measured by the nodes of the tree overlay network. In addition, this protocol requires feedback of reports. To meet this need, a return channel is added to the Scribe. Related works show that the TFMCC, especially in light of his fair and moderately aggressive behavior, fits well in the context of video streaming addressed in this work.

The proposal also includes a technique of retransmission of the base layer in order to compensate block losses, aiming to ensure a minimal amount of information for video playback.

The architecture outlined aims to provide the highest quality user experience in live broadcasts and large scale, being evaluated, at first glance, by testing in different scenarios and based on different video sequences.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

4CIF	4 vezes CIF (resolução 704 x 576)
ACP	Auxiliary Coded Picture
AIAD/H	Additive Increase / Decrease with History
AIMD	Additive Increase / Multiplicative Decrease
ALM	Application Level Multicast
API	Application Program Interface
ASM	Any-Source Multicast
AU	Access Unit
B-Frame	Bi-directionally predicted frame
BSD	Berkeley Software Distribution
CABAC	Context-Adaptive Binary Arithmetic Coding
CAT	Content Adaptation Technique
CBR	Constant Bitrate
CDN	Content Distribution Network
CGS	Coarse-Grain Scalable coding
CIF	Common Intermediate Format (resolução 352 x 288)
CLR	Current Limiting Receiver
CVS	Coded Video Sequence
D	Dependency layer
DHT	Distributed Hash Table
DPB	Decoded Picture Buffer
EoSeq	End of Sequence
EoStream	End of Stream
EP	Erro Padrão
F1	Frequência de T1
F2	Frequência de T2
FB	Fragmentador de Blocos
FEC	Forward Error Correction
FGS	Fine-Grain Scalable coding
FIFO	First In First Out
FPS	Frames Per Second
FSM	Finite-State Machine
GAIMP	General Additive Increase / Multiplicative Decrease
GOP	Group Of Pictures
GPL	GNU General Public License
HHI	Heinrich-Hertz-Institute
HRD	Hypothetical Reference Decoder
IANA	Internet Assigned Numbers Authority

IDR	Instantaneous Decoding Refresh
IETR	Institute of Electronics and Telecommunications of Rennes
I-Frame	Intra-coded frame
IIAD	Inverse Increase / Additive Decrease
IQA	Initial Quality Adaptation
ISO/IEC JTC1	International Organization for Standardization / International Electrotechnical Commission Joint Technical Committee 1
JNA	Java Native Access
JOGL	Java Binding for the OpenGL API
JSVM	Joint Scalable Video Model
JVM	Java Virtual Machine
JVT	Joint Video Team
JXTA	Juxtapose
LWJGL	Lightweight Java Game Library
M0	Primeira região de memória compartilhada do RV
M1	Segunda região de memória compartilhada do RV
MANE	Media Aware Network Elements
MDC	Multiple Description Coding
MGS	Medium-Grain Scalable coding
MPEG	Moving Pictures Experts Group
NAL	Network Abstraction Layer
NALU	Network Abstraction Layer unit
NAT	Network Adaptation Technique
NS-2	Network Simulator 2
OpenGL	Open Graphics Library
P2P	Peer-to-Peer
PCP	Primary Coded Picture
PDTP	Peer Distributed Transfer Protocol
P-Frame	Predictively coded frame
POC	Picture Order Count
PPS	Picture Parameter Set
PQA	Progressive Quality Adaptation
Q	Quality Layer
QCIF	Um quarto de CIF (resolução 176 x 144)
QoS	Quality of Service
QP	Quantization Parameter
RB	Remontador de Blocos
RCP	Redundant Coded Picture
RD	Rate Distortion
RFC	Request For Comments

RGB	Representação de cor com as componentes: vermelho (R), verde (G) e azul (B)
RGBA	Representação de cor com as componentes: RGB e transparência (A)
RTP	Real-Time Protocol
RTT	Round-Trip Time
RV	Reprodutor de Vídeo
SEI	Supplemental Enhancement Information
SIMD	Square Increase / Multiplicative Decrease
SNR	Signal-to-Noise Ratio
SPS	Sequence Parameter Set
SQRT	Square-Root Increase / Decrease
SSM	Source-Specific Multicast
SVC	Scalable Video Coding
SWT	Eclipse SWT (Standard Widget Toolkit)
T	Temporal Layer
T1	Temporizador 1 do módulo RV
T2	Temporizador 2 do módulo RV
TCP	Transmission Control Protocol
TDD	Test Driven Design
TEAR	TCP-Emulation at Receiver
TFMCC	TCP-Friendly Multicast Congestion Control
TFRC	TCP-Friendly Rate Control
TFRCP	TCP-Friendly Rate Control Protocol
VBR	Variable Bitrate
VCEG	Video Coding Experts Group
VCL	Video Coded Layer
VoD	Video on Demand
YUV	Representação de cor com as componentes: brilho (Y) e cor (U e V)

1 - INTRODUÇÃO

1.1 - CONTEXTUALIZAÇÃO

As pesquisas em redes Par-a-Par (P2P, *Peer-to-Peer*) têm crescido rapidamente nos últimos anos, fato esse reflexo da importância cada vez mais evidente dos benefícios dos sistemas distribuídos. Estudos comparativos mostram a aplicabilidade, para diversos fins, que vem sendo conferida às redes P2P [1] [2] [3] [4] [5] [6] [7].

As redes P2P são uma solução promissora e eficaz para o desenvolvimento de aplicações de larga escala para a Internet [8]. Tal propriedade é resultado de uma característica fundamental dessas redes: a escalabilidade. Quanto maior o número de participantes, maior a capacidade da rede sem prejuízo sensível de outras características, como disponibilidade, tempo de resposta, etc. Essas qualidades tornam as redes P2P um importante substrato para distribuição de fluxo contínuo de vídeo em larga escala.

No processo de distribuição de fluxo contínuo de vídeo (vídeo *streaming*), a informação é enviada sem interrupção e o usuário a consome – reproduz o vídeo – tão logo recebe. A distribuição é classificada como Sob Demanda (VoD, *Video on Demand*), quando é permitido ao usuário selecionar o momento que deseja iniciar a recepção, ou Ao Vivo (*live streaming*), caso contrário. A distribuição ao vivo é o interesse deste trabalho.

Realizar distribuição de vídeo considerando uma arquitetura cliente-servidor convencional, que é baseada na comunicação ponto-a-ponto (*unicast*), é um modelo pouco escalável e, portanto, limitado em capacidade, pois com o aumento do número de participantes na rede, maior a banda passante exigida para transmissão a partir da fonte de vídeo. A abordagem alternativa que visa resolver esse problema é o modelo de comunicação multidestinatária (*multicast*), no qual a informação enviada pela fonte de vídeo é recebida por um grupo de usuários, o que não representa, necessariamente, aumento da banda passante exigida da fonte de vídeo.

O IP Multicast [9] é um protocolo de comunicação multidestinatária da camada de rede. Sua implantação tem sido lenta e complexa, pois requer adaptações e configurações que precisam ser aplicadas em toda a Internet. Surgem então, como soluções promissoras e de rápida instalação, inspiradas na pouca penetração do IP Multicast [10] [11] [12] [13] [14], as redes lógicas *multicast* criadas virtualmente sobre a Internet. Essa técnica recebe o nome de ALM (*Application Level Multicast*). A comunicação multidestinatária em redes P2P é aqui considerada um caso de ALM.

Cientistas têm dedicado esforços em criar arquiteturas bem definidas que permitam a instalação prática do modelo de comunicação par-a-par, permitindo então que sistemas de distribuição de vídeo obtenham maior penetração de mercado sem aumento considerável de custo, dado o benefício da escalabilidade dos sistemas P2P.

O sistema de rede P2P Pastry [15] associado com a ferramenta de comunicação multidestinatária Scribe [14] alcança, conforme [14], desempenho muito próximo do IP Multicast em custo computacional, em tempo de resposta e em consumo de banda de rede, provando ser uma solução eficiente, na maneira como soluciona o problema, e eficaz, no alcance do resultado propriamente dito.

Um desafio importante concernente à distribuição de vídeo é garantir os requisitos de qualidade de serviço (QoS, *Quality of Service*). Transmitir vídeo requer grande capacidade de banda passante na rede. Porém esse privilégio não é uma realidade de todas as aplicações. As aplicações robustas precisam estar cientes da heterogeneidade dos usuários na Internet, principalmente no que tange à capacidade de transmissão. No intuito de manter a melhor qualidade possível do conteúdo sendo transmitido, o sistema de distribuição de vídeo deve ser dotado da capacidade de adaptação. Nesse contexto, podem ser adotadas duas abordagens: (a) Adaptação da rede, ou NAT (*Network Adaptation Technique*), quando a adaptação diz respeito ao controle de alocação de recursos de rede; ou (b) Adaptação de conteúdo, CAT (*Content Adaptation Technique*), quando a informação sendo transmitida na rede é modificada em função da capacidade observada dos enlaces de comunicação.

No escopo amplo da Internet, praticar adaptação de rede (NAT) para manter a qualidade de vídeo remete ao mesmo problema vivenciado pelo IP Multicast: a solução teria baixa penetração devido ao grande impacto para implantação na Internet, vez que é necessária a intervenção nos nodos participantes do sistema.

No cenário de distribuição de vídeo, a aplicação de CAT é possível com o uso de técnicas de codificação escalável de vídeo. A codificação escalável permite o incremento e/ou decremento da qualidade do vídeo a um baixo custo computacional – sem necessidade de transcodificação de formato. A adaptação do vídeo pode ser realizada com base em protocolos que permitam a mensuração da capacidade do enlace de rede. Os protocolos de controle de congestionamento têm como resultado de sua função de decisão o valor da taxa de transmissão que deve ser obedecida pela fonte de transmissão. Essa taxa (*bitrate*) pode ser usada como referência para adaptação do vídeo codificado em técnica escalável. Dessa forma, então, o esquema CAT é implementado através do ajuste da qualidade do vídeo para que a taxa de transmissão esteja em concordância com aquela sugerida pelo protocolo de controle de congestionamento.

1.2 - CARACTERIZAÇÃO DO PROBLEMA

A Internet foi projetada originalmente para viabilizar aplicações simples, que não envolviam tráfego de tempo real. Posteriormente, o seu emprego veio a ser bastante difundido, incluindo aplicações de tempo real tais como *streaming* de vídeo e de áudio. O tráfego de vídeo representa um dos mais exigentes para a Internet, envolvendo requisitos como necessidade elevada de banda, sincronismo e adaptação da rede a diferentes taxas de transmissão, requisitos esses que se tornam de atendimento mais difícil em função do número crescente de usuários.

Neste contexto, arquiteturas baseadas em P2P apresentam-se como uma parte da solução para um dos problemas enfrentados pelas arquiteturas cliente-servidor, que é a baixa escalabilidade. Adicionalmente, novos esquemas de codificação de vídeo têm sido produzidos, dentre os quais se inclui o H.264 na sua extensão SVC que, face à possibilidade de adaptação de conteúdo, permite ajustar a relação necessária entre disponibilidade de recursos (banda) e a demanda oferecida pelas aplicações de *streaming* de vídeo. Ao lado de tais esquemas de codificação, destaca-se que a adoção de protocolos capazes de tratar possíveis situações de congestionamento na rede se mostra bastante relevante.

O presente trabalho busca então tratar o atendimento adequado às necessidades de transmissão de vídeo na Internet com base em redes P2P e no padrão H.264 SVC, demandando a definição e implementação de uma arquitetura capaz de integrar os componentes citados, favorecendo o controle de congestionamento e a melhoria da Qualidade de Experiência, especialmente nas transmissões realizadas ao vivo.

1.3 - OBJETIVOS

Como primeiro objetivo, este trabalho visa selecionar e, quando for o caso, complementar e/ou adaptar, tecnologias no âmbito de codificação de vídeo escalável, de redes par-a-par e de controle de congestionamento que sejam adequadas para realização do tema proposto: distribuição adaptativa de fluxo contínuo ao vivo de vídeo em larga escala.

O segundo objetivo é elaborar uma arquitetura que contemple a utilização das tecnologias selecionadas, propondo adaptações e/ou criando soluções necessárias para a integração de todas elas.

O terceiro objetivo deste trabalho é apresentar detalhes técnicos com respeito à implementação da arquitetura proposta, visando tornar públicos os caminhos de sucesso e de erro trilhados no percorrer do desenvolvimento da prova de conceito do sistema.

O quarto objetivo, por fim, envolve a realização de testes, em diferentes cenários, para aferir o desempenho da arquitetura implementada, bem como de opções adotadas para a implementação.

1.4 - CONTRIBUIÇÕES

- Levantamento de tecnologias P2P e de protocolos de controle de congestionamento, elencando as vantagens e as desvantagens das soluções mais apropriadas para o contexto de distribuição de fluxo de vídeo em tempo real;
- Definição do paradigma de compressão de vídeo em blocos independentes e investigação de sua aplicação no cenário P2P em conjunto com o controle de congestionamento e adaptação de conteúdo;
- Proposta de formato para encapsulamento e fragmentação dos blocos de vídeo em pacotes RTP bem como extensão do formato RTP para suporte ao piggybacking do TFMCC;
- Proposta de algoritmo para remontagem de pacotes;
- Soluções de implementação dos componentes da arquitetura, o que inclui, dentre outros, reprodução de vídeo baseada no uso de memória compartilhada;
- Proposta de esquema de retransmissão da camada base de blocos SVC.

1.5 - ORGANIZAÇÃO DO TRABALHO

O capítulo 2 trata da codificação escalável de vídeo, onde são apresentados o padrão H.264 SVC e suas principais características. No capítulo 3 é abordada uma visão geral de sistemas de rede par-a-par. É feita uma analise comparativa de sistemas projetados para distribuição de mídia. A rede Pastry é explanada com maiores detalhes, bem como o sistema de comunicação multidestinatária Scribe. O capítulo 4 introduz os protocolos de controle de congestionamento, com enfoque no TFMCC.

Uma proposta de arquitetura adaptativa para distribuição de vídeo é apresentada no capítulo 5, seguida de tópicos a respeito da implementação dessa arquitetura, apresentados no capítulo 6.

No capítulo 7 são mostrados e discutidos resultados da execução do sistema em diferentes cenários de teste. A dissertação é finalizada no capítulo 8, com as conclusões e trabalhos futuros.

2 - CODIFICAÇÃO ESCALÁVEL DE VÍDEO

2.1 - INTRODUÇÃO

Face ao tema escalabilidade de vídeo, existem diversas técnicas que permitem "codificar uma vez e decodificar muitas vezes" em diferentes qualidades (*encoding once and decoding many*). Essa expressão significa dizer que o custo computacional para converter o vídeo codificado para uma qualidade diferente da inicial é menor que o custo de transcodificação, ou seja, decodificar e codificar novamente o vídeo. As principais técnicas de escalabilidade são agrupadas em duas grandes classes: SVC (*Scalable Video Coding*) e MDC (*Multiple Description Coding*). Em ambos os esquemas o vídeo codificado é composto de camadas (*layers*).

No esquema MDC, as camadas (neste caso também denominadas de "partições") são independentes e contribuem igualmente no incremento de qualidade do vídeo decodificado. Quanto mais camadas utilizadas no processo de decodificação, melhor será o vídeo reconstruído na saída do decodificador.

No SVC, existe dependência entre as camadas e o incremento de qualidade não é garantidamente uniforme, como no MDC. A primeira camada representa o vídeo na sua pior qualidade.

Devido à independência entre as camadas, o MDC está, por natureza, associado a uma maior complexidade de decodificação, fato esse que lhe tem conferido menor atenção mercadológica com relação ao esquema SVC. Outra importante vantagem deste é a existência de padronização – o H.264 SVC – e de implementação de referência – o JSVM (*Joint Scalable Video Model*). Esses são os motivos da escolha do SVC como técnica fundamental para adaptação de conteúdo na arquitetura proposta neste trabalho.

2.2 - O PADRÃO H.264 SVC

2.2.1 - Introdução

A extensão SVC (*Scalable Video Coding*) [16] do formato H.264 foi originalmente proposta pelo instituto HHI (*Heinrich-Hertz-Institute*), que é parte do instituto Fraunhofer, Berlin. Sua atual versão é mantida pela ITU-T (*International Telecommunication Union*) em conjunto com o ISO/IEC JTC1 (*International Organization for Standardization* / *International Electrotechnical Commission Joint Technical Committee 1*).

No H.264 SVC, cada camada representa o incremento de uma ou mais das seguintes dimensões de escalabilidade:

(a) Temporal: taxa de quadros por segundo;
(b) Espacial: quantidade de pontos (*pixels*) por quadro;
(c) De qualidade: ruído de quantização ou SNR (*Signal-to-Noise Ratio*).

Essas três escalabilidades estão associadas, respectivamente, às técnicas de compressão de vídeo baseadas na exploração da redundância temporal, da redundância espacial e na quantização utilizada para representação os dados codificados.

A Compensação de Movimento diz respeito à compressão obtida com respeito à exploração da redundância temporal, ou seja, da correlação entre um determinado quadro e seus vizinhos. Em função da utilização dessa técnica, os quadros codificados são assim classificados:

- Quadros do tipo I (*Intra-coded frame*): codificados independentemente dos demais quadros;
- Quadros do tipo P (*Predictively coded frame*): codificados com base na diferença com relação a um quadro anterior;
- Quadros do tipo B (*Bi-directionally predicted frame*): codificados com base em um quadro anterior e um posterior.

Os quadros do tipo I determinam pontos de "decodificação instantânea" IDR (*Instantaneous Decoding Refresh*), no sentido de que não há dependência temporal para decodificação. A utilização de quadros P e principalmente B aumenta a eficiência da taxa de distorção (RD, *Rate Distortion*), ou seja, aumenta a relação entre qualidade do vídeo e taxa de transmissão (*bitrate*). Os quadros B, entretanto, resultam em atraso no processo de decodificação, vez que quadros futuros precisam ser transmitidos antes do quadro atual, devido à relação de dependência entre eles.

Os dados codificados em H.264 são armazenados e/ou transmitidos na forma de uma série ordenada de unidades NAL (*Network Abstraction Layer*). Na extensão SVC, a adaptação de taxa é conseguida com a remoção de unidades NAL relativas a uma ou mais camadas de incremento temporal, espacial ou de qualidade. A camada de abstração de rede (NAL) será apresentada em seção seguinte.

2.2.2 - Escalabilidade Temporal

A escalabilidade temporal é obtida quando o vídeo é particionado em uma camada temporal base T_0 e uma ou mais camadas temporais de melhoramento T_n, $n \geq 1$, sendo que: a remoção de todos os quadros pertencentes às camadas T maior ou igual a k, $k > 0$, resulta em um vídeo derivado válido, isto é, decodificável, porém com menor frequência de quadros (FPS, *Frames Per Second*).

Uma forma de obter escalabilidade temporal é restringindo a Predição por Compensação de Movimento Inter-Quadros (*inter-frame motion-compensated prediction*) de forma que o quadro sendo predito utiliza como referências apenas quadros cuja camada temporal T_n é menor ou igual à sua. Como consequência desse processo a relação de dependência temporal entre os quadros torna-se hierárquica.

A Figura 1 mostra um exemplo de GOP (*Group Of Pictures*) hierárquico de tamanho 4 e suas três possíveis configurações: vídeo contendo apenas a camada base (T_0), onde temos uma taxa de quadros genericamente representada por x; com duas camadas (T_0, T_1); e com as três camadas, situação de melhor qualidade temporal.

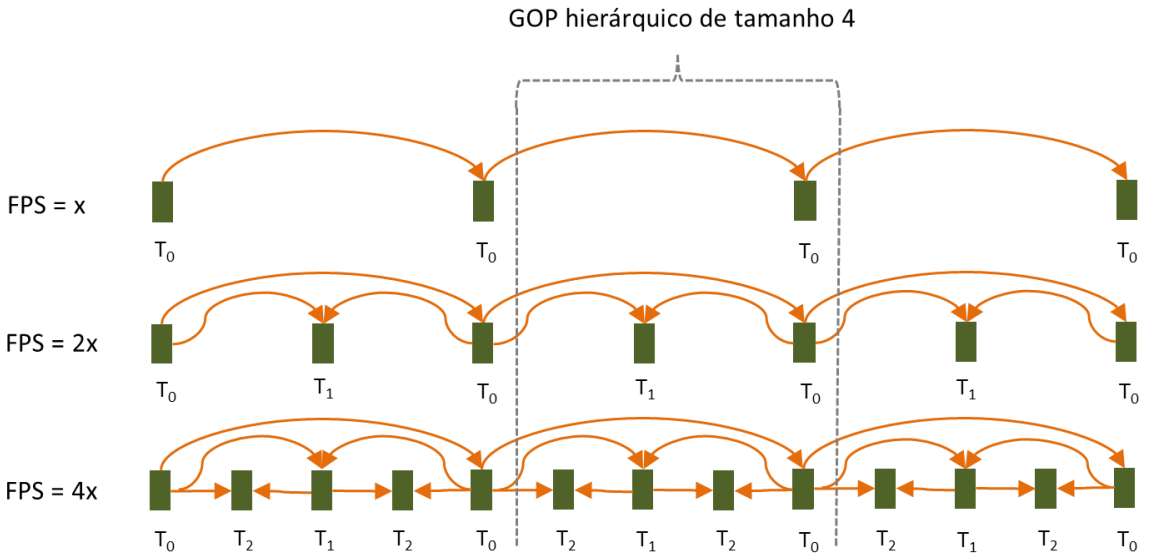


Figura 1 – Escalabilidade temporal de vídeo H.264 SVC

GOP é o menor conjunto de tipos I, P e B que se repete ao longo do vídeo. É importante salientar que a definição de GOP é especializada no SVC. Em geral diz-se que "GOP inicia por um quadro do tipo I" [17]. Porém, a definição usada neste trabalho, em concordância com a implementação de referência JSVM e com a norma H.264 SVC [16], é: "O GOP termina a cada quadro da camada temporal T_0". No H.264 SVC a camada temporal T_0 pode

ser formada por quadros do tipo I ou P. As camadas superiores têm livre escolha entre I, P ou B.

A Figura 1 é o superconjunto das possibilidades de relação de dependência temporal entre os quadros, onde na camada T_0 tem-se quadros P e nas camadas T_1 e T_2 tem-se B. Uma maneira de obter atraso zero no processo de decodificação é utilizar apenas quadros I ou P conforme Figura 2.

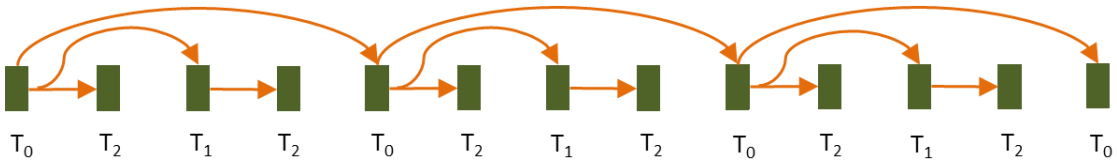


Figura 2 – Dependência temporal para GOP contendo quadros I ou P

Análises da escalabilidade temporal podem ser encontradas em [18] e [19].

2.2.3 - Escalabilidade Espacial

Com respeito à escalabilidade espacial, o H.264 SVC baseia-se na abordagem multicamadas empregada nos padrões H.262/MPEG-2 Video, H.263 e MPEG-4 Visual: dentro de cada camada espacial há predição por Compensação de Movimento e predição intra-quadro independentes das demais camadas. Contudo, tem-se a inclusão de um terceiro conceito: predição inter-camada (*inter-layer prediction*). Nesse processo, o quadro da camada inferior, de menor resolução, passa por um filtro para aumento de resolução (*upsampling*) e é utilizado como referência para melhoramento da eficiência de codificação.

Na Figura 3 as setas verticais indicam a predição inter-camada. D_0 (*dependency layer*) é a camada espacial base e D_1 é uma camada de melhoramento espacial (resolução maior).

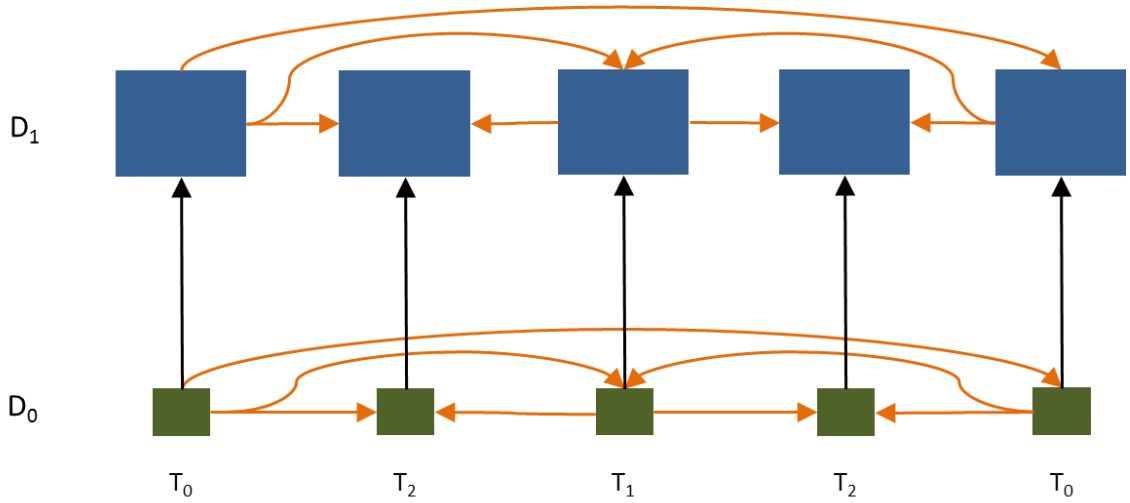


Figura 3 – Escalabilidade temporal e espacial de vídeo H.264 SVC

2.2.4 - Escalabilidade de Qualidade

A escalabilidade de qualidade, também chamada de escalabilidade perceptual ou escalabilidade de SNR, está associada ao uso de diferentes parâmetros de quantização (QP, *Quantization Parameter*). Ela pode ser entendida como um caso especial da escalabilidade espacial, quando a camada inferior possui a mesma resolução. Um melhoramento de qualidade é obtido via codificação da imagem residual em passos menores de quantização. Ou seja, a camada superior possui QP menor que a camada inferior.

Na Figura 4 as cores verde escuro e azul escuro representam camadas de qualidade base Q_0 (*quality layer*), cuja numeração de Q é iniciada em cada camada espacial D. As cores verde claro e azul claro são camadas de melhoramento de qualidade D_1. O contorno tracejado indica que qualquer uma das camadas de qualidade Q de uma determinada camada espacial D podem ser usadas para predição temporal na codificação ou decodificação de um quadro. Quanto maior a camada Q utilizada, melhor a eficiência da taxa de distorção (RD).

Camadas de melhoramento de qualidade podem ser transmitidas na forma de diferentes camadas D, esquema esse chamado de CGS (*Coarse-Grain Scalable coding*), ou na forma de diferentes camadas Q dentro de uma mesma camada D, que é o esquema MGS (*Medium-Grain Scalable coding*). Ambos os modos dizem respeito à sinalização, mas não mudam o processo de codificação. A Figura 5 mostra o resultado da estrutura em camadas do modo CGS e MGS.

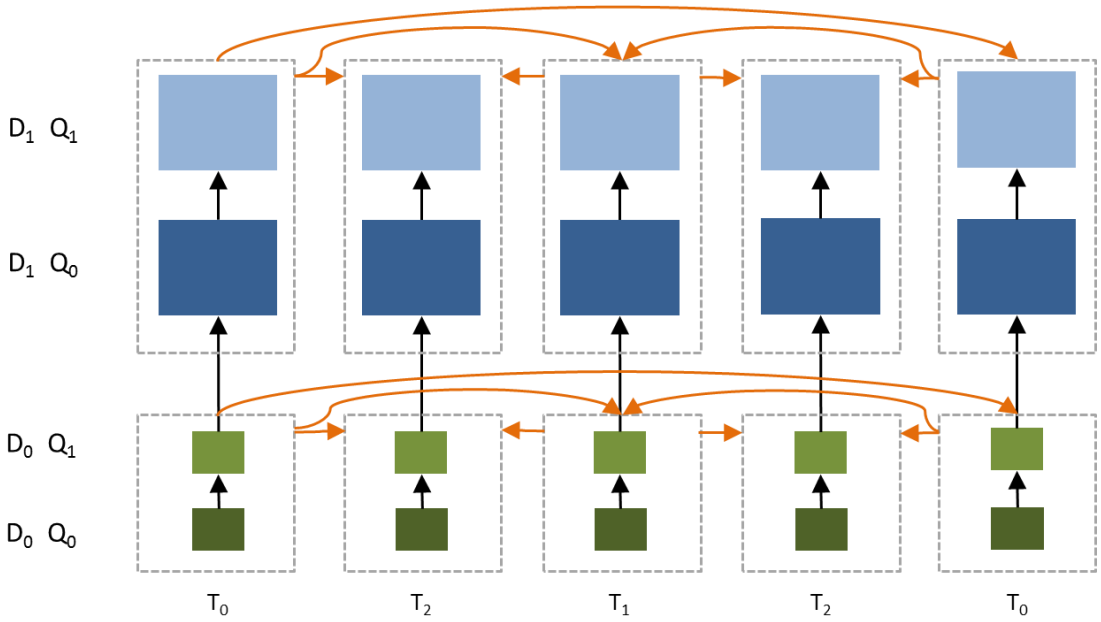


Figura 4 – Escalabilidade temporal, espacial e de qualidade de vídeo H.264 SVC

No processo de decodificação o salto em camada de dependência D pode ocorrer apenas em quadros do tipo I, que demarcam pontos de "decodificação instantânea" (IDR). O salto em camada de qualidade Q pode ocorrer, a depender da sinalização do vídeo, a qualquer momento. Daí o MGS ser uma alternativa interessante ao CGS, pois permite uma maior quantidade de possíveis taxas de transmissão.

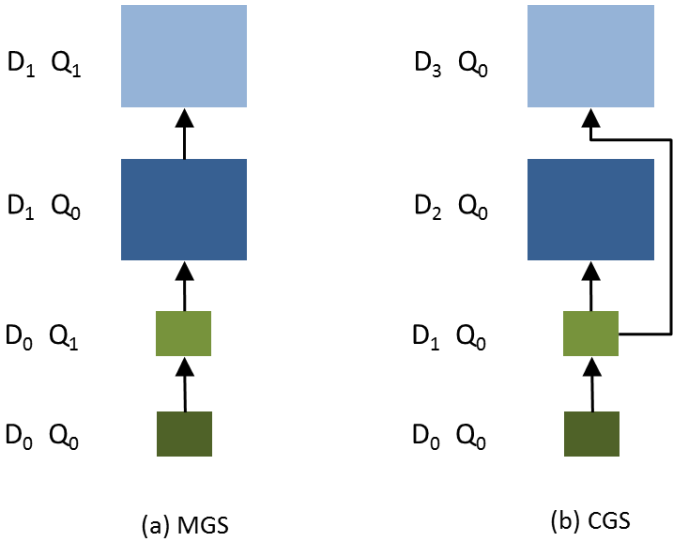


Figura 5 – Escalabilidade de qualidade nos modos MGS e CGS

Outra limitação do CGS diz respeito à predição inter-camada: o total de camadas D utilizadas como referência para camadas superiores é no máximo 2. Na Figura 5, D_0 e D_1 são camadas de referência (para D_1 e D_2 respectivamente). Portanto, D_2 não pode ser uma

camada de referência. Devido a essa limitação, a referência para D_3 pode ser, por exemplo, D_1. Tal limitação reduz a eficiência de compressão.

Existe um terceiro esquema de escalabilidade de qualidade – o FGS (*Fine-Grain Scalable coding*) – no qual o salto em camada D pode ocorrer a qualquer momento. A extensão SVC do H.264 ainda não normatizou esse esquema, vez que ele oferece baixa eficiência com respeito à taxa de distorção RD.

2.2.5 - Camada de Abstração de Rede

A Camada de Abstração de Rede, NAL (*Network Abstraction Layer*), é a parte do padrão H.264 SVC que determina um formato para armazenamento do vídeo codificado na forma de uma sequência de pacotes, chamados de unidades NAL ou apenas NALU (*Network Abstraction Layer unit*), tornando o vídeo SVC predisposto para transmissão em rede de pacotes (*network-friendly*).

A presente seção não trata de um resumo da literatura, mas sim de uma investigação da norma [16] para melhor entendimento da estrutura sintática do *stream* SVC. A análise da estrutura de pacotes NALU (que compõem o *stream*) é importante para elaboração da proposta de arquitetura deste trabalho.

As unidades NAL são classificadas conforme a seguinte taxonomia:

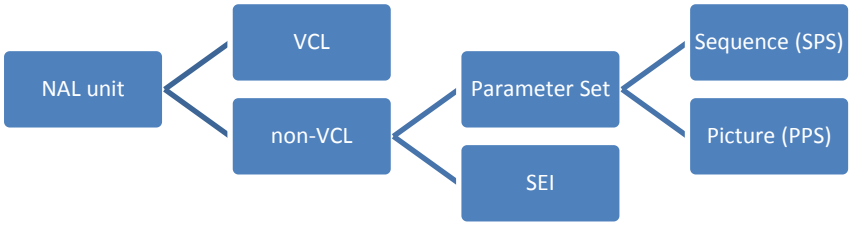


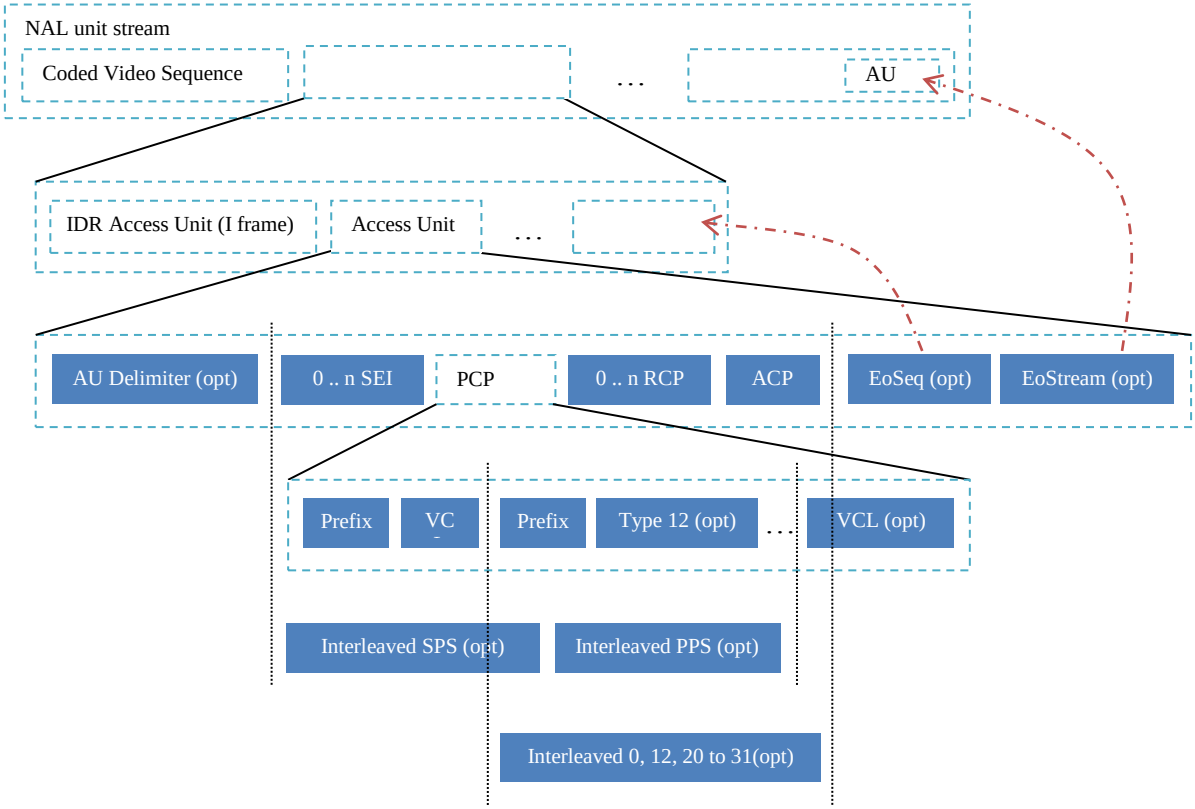
Figura 6 – Taxonomia das unidades NAL de vídeo SVC

As unidades VCL (*Video Coded Layer*) possuem informações do vídeo propriamente dito – os quadros comprimidos. As unidades non-VCL (não VCL) são informações de sinalização, controle e configuração que auxiliam a decodificação das unidades VCL.
As principais unidades non-VCL são:

- SPS (*Sequence Parameter Set*), contém parâmetros de codificação que se aplicam a uma sequência inteira de quadros. Exemplo: formato de cor, tipo de codificação de entropia, etc;
- PPS (*Picture Parameter Set*), contém parâmetros de codificação que se aplicam a quadros em específico;

- SEI (*Supplemental Enhancement Information*), contém informações adicionais não obrigatórias para a decodificação. São 36 tipos de mensagem SEI, sendo 11 deles introduzidos pelo SVC. O tipo 24, por exemplo, discrimina a lista de escalabilidades contidas no vídeo.

A estrutura de um *bytestream* SVC, após esmiuçar a norma, pode ser assim representada:



(opt) = Unidade NAL opcional.

Figura 7 – Estrutura do *bytestream* H.264 SVC

O vídeo SVC (*NAL unit stream*) é composto por um ou mais conjuntos CVS (*Coded Video Sequence*), que são sequências de decodificação independentes, ou seja, não há predição por Compensação de Movimento entre quadros de CVS diferentes.

Pacotes SPS determinam início de CVS. Verifica-se, na prática, que os pacotes PPS não são retransmitidos no início de cada CVS, tão pouco isso é feito para pacotes SEI, vez que o codificador SVC almeja, por padrão, a melhor SNR e, portanto, evita redundâncias.

Cada CVS é composto por um por uma ou mais AU (*Access Unit*). Cada AU está associada um único quadro. A primeira AU do CVS é chamada de IDR AU (*Instantaneous Decoding Refresh*), pois contém um quadro IDR. Uma IDR AU não determina início de CVS.

O sequenciamento das unidades NAL dentro de uma AU é mostrada na Figura 8.

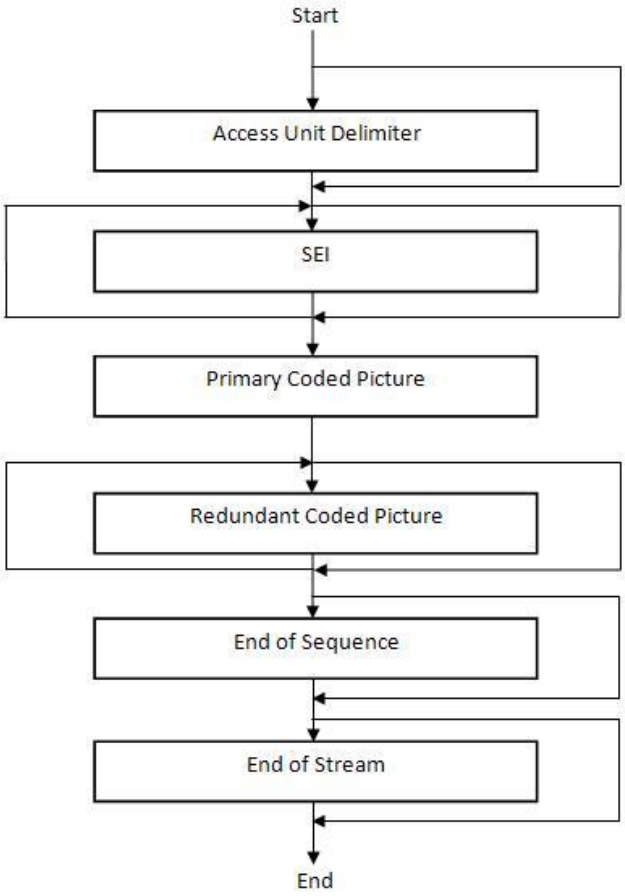


Figura 8 – Sequenciamento das unidades NAL de uma AU (*Access Unit*). Fonte: [16]

O conjunto PCP (*Primary Coded Picture*) contém unidades VCL que compõem o quadro. No SVC, todo VCL é precedido de uma unidade chamada SVC Prefix. Cada VCL pertence a uma determinada camada espacial D ou de qualidade Q, lembrando que a AU, como um todo, pertence a uma determinada camada temporal T.

O conjunto ACP (*Auxiliary Coded Picture*) é semelhante ao PCP e usado para outros fins suplementares, como, por exemplo, codificação do canal alpha (transparência).

Se o quadro ao qual se refere a AU for codificado em apenas um *slice*, teremos uma única unidade VCL para cada combinação das escalabilidades D e Q no PCP. Caso contrário, várias unidades VCL estarão presentes para cada combinação de D e Q. O procedimento de *slicing* diz respeito ao particionamento dos pontos (*pixels*) do quadro durante a codificação. O quadro do tipo P é aquele que possui pelo menos um *slice* P. O quadro do tipo B é aquele que possui pelo menos um *slice* B.

Após o PCP, informações de redundância podem ser adicionadas. Tais informações estão no conjunto chamado RCP (*Redundant Coded Picture*).

Unidades que delimitam final de CVS (EoSeq, *End of Sequence*) e final do vídeo (EoStream, *End of Stream*) são opcionais.

As informações de PPS e SPS são necessárias para decodificação dos *slices*. As regras para posicionamento dessas unidades no vídeo, bem como de outras unidades NAL de sinalização, são complexas. Nesse sentido, a Figura 7 utiliza o termo *interleaved* para indicar que a unidade pode estar presente em diferentes posições entre as linhas verticais pontilhadas. Infere-se, portanto, que os pacotes PPS e SPS, se existirem, devem estar localizados após o *AU Delimiter* (se este existir) e antes do último pacote VCL do PCP.

A norma SVC diz também que pacotes PPS e SPS determinam início de AU. Ainda, o primeiro pacote VCL após o último VCL do último PCP determina início de AU. Essas condições, além de diversas outras, estão documentadas em [16].

Outros pacotes de sinalização, de tipo 0, 12 ou 20 até 31, quando existirem, devem estar após o primeiro VCL do PCP e antes de EoSeq ou EoStream, se presentes. Os tipos de unidade NAL estão elencados na Tabela 1.

Tabela 1 – Tipos de unidade NAL. Fonte: [16]

Tipo	Descrição	Categoria conforme SVC
0	Não especificado	
1	Coded Slice non-IDR	VCL
2 a 4	Não aplicável ao SVC	
5	Coded Slice IDR	VCL
6	SEI – Supplemental Enhancement Information	non-VCL
7	SPS – Sequence Parameter Set	non-VCL
8	PPS – Picture Parameter Set	non-VCL
9	AU Delimiter	non-VCL
10	End of Sequence	non-VCL
11	End of Stream	non-VCL
12	Filler Data (padding 0xFF)	non-VCL
13	SPS extension	non-VCL
14	SVC Prefix	non-VCL se antes de tipo 12 ou VCL caso contrário
15	Subset SPS	non-VCL
16 a 18	Reservado	

19	Coded Slice ACP	non-VCL
20	Coded Slice SVC	VCL
21 a 23	Reservado	
24 a 31	Não especificado	

As unidades VCL, como dito, requerem informação de PPS e SPS para serem decodificadas. Essa dependência é mostrada na Figura 9. O decodificador SVC precisa, portanto, manter memória dos pacotes PPS e SPS.

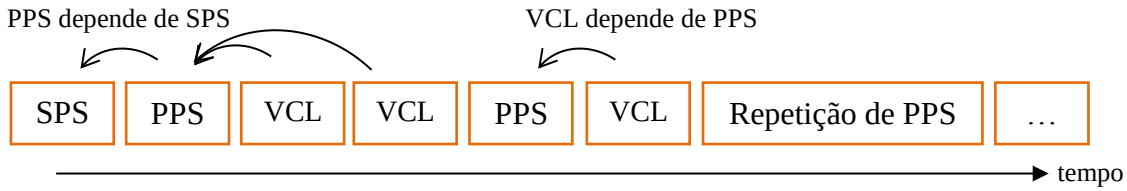


Figura 9 – Dependência entre unidades VCL, PPS e SPS

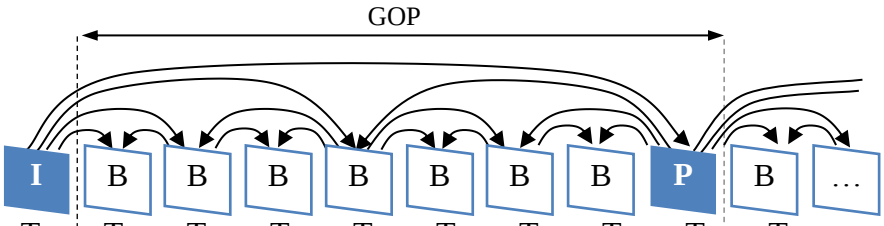
2.2.6 - Ordenamento de Quadros

O padrão H.264 define três tipos de ordenamento de imagens:

- (a) Ordem de decodificação (*decoding order*), ordem na qual as imagens são decodificadas no receptor;
- (b) Ordem de apresentação (*display order*), ordem na qual as imagens são exibidas para o usuário;
- (c) Ordem de referência (*reference order*), ordem na qual as imagens são organizadas para facilitar a predição inter-quadros.

A ordem de apresentação é calculada com base em parâmetros localizados em cabeçalhos de unidades VCL, que descrevem o POC (*Picture Order Count*). A ordem de decodificação está intimamente ligada à dependência inter-quadros regida pela predição de Compensação de Movimento mostrada na Figura 1. Essa ordem é elaborada de tal forma que a quantidade de imagens na memória DPB (*Decoded Picture Buffer*) do decodificador seja mínima. A ordem de referência está relacionada a estruturas internas do DPB – as listas 0 e 1 – usadas no processo de resolução da predição inter-quadros.

A Figura 10 mostra as relações de dependência temporal considerando GOP hierárquico de tamanho 8. Na figura, p_i é o quadro de número i na ordem natural do vídeo.



Camada temporal:	T_0	T_3	T_2	T_3	T_1	T_3	T_2	T_3	T_0	T_3
Ordem de apresentação:	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
Ordem de decodificação:	1º	5º	4º	6º	3º	8º	7º	9º	2º	13º
Ordem de transmissão:	p_0	p_8	p_4	p_2	p_1	p_3	p_6	p_5	p_7	p_{16}

Figura 10 – Ordenamento de quadros no H.264 SVC.

A ordem de decodificação está diretamente relacionada com a ordem de transmissão dos quadros no *stream*. Analisando o GOP composto dos quadros p_1 a p_8 e considerando as relações de dependência entre eles, observa-se que: p_1 requer p_0 e p_2. Considere que p_0 já fora transmitido no GOP anterior. Para decodificar p_2 é necessário p_4, que por sua vez requer p_8. Portanto, p_8 é o primeiro quadro a ser transmitido. Depois se transmite p_4, depois p_2 e, então, transmite-se p_1. Tem-se, portanto, que p_1 será decodificado com atraso de 3 quadros. Esse atraso é chamado de atraso estrutural (*structural delay*), que é inerente à estrutura hierárquica do GOP.

A Tabela 2 mostra os atrasos mínimos em cada fase de um sistema de transmissão de vídeo H.264 SVC considerando o GOP hierárquico de tamanho 8.

Tabela 2 – Atraso estrutural para GOP hierárquico de tamanho 8

GOP:	1º				2º								3º									
Fonte:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Codificado:									0	8	4	2	1	3	6	5	7	16	12	10	9
Transportado:										0	8	4	2	1	3	6	5	7	16	12	10
Decodificado:											0	8	4	2	1	3	6	5	7	16	12
Exibido:															0	1	2	3	4	5	6

tempo

Considere, na tabela acima, que cada célula corresponde ao período de 1 quadro.

O processo de decodificação ocorre quadro a quadro, ou seja, a cada interação de AU (*Access Unit*). O resultado na saída do decodificador, como mostrado na Figura 11, por ser zero ou mais quadros.

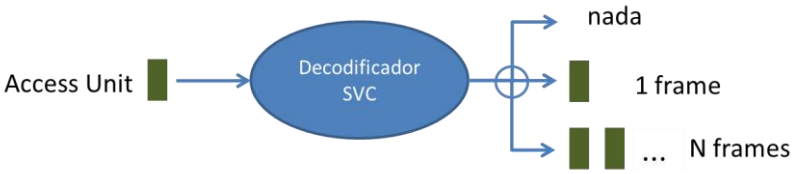


Figura 11 – Saída do decodificador SVC para cada interação de AU

Para entender essa propriedade, é necessário analisar as relações de dependência entre os quadros do GOP. A Figura 10 permite inferir que a cada interação de AU temos:

Tabela 3 – Saída do decodificador H.264 SVC a cada interação de AU

Interação	Entrada	Saída	Comentário
0	p_0	Nenhuma	Mantido no DPB, pois é referência para p_1, p_2, p_4, p_8.
1	p_8	Nenhuma	Mantido no DPB, pois é referência para p_9, p_{10}, p_{12}, p_{16}.
2	p_4	Nenhuma	Mantido no DPB, pois é referência para p_2, p_3, p_5, p_6.
3	p_2	Nenhuma	Mantido no DPB, pois é referência para p_1, p_3.
4	p_1	Nenhuma	p_1 não é referenciado, porém aguarda saída de p_0 (a saída do decodificador obedece a ordem de apresentação).
5	p_3	Nenhuma	p_1 não é referenciado, porém aguarda saída de p_0, p_1, p_2.
6	p_6	Nenhuma	Mantido no DPB, pois é referência para p_5, p_7.
7	p_5	Nenhuma	p_5 não é referenciado, porém aguarda saída de p_0 até p_4.
8	p_7	Nenhuma	p_7 não é referenciado, porém aguarda saída de p_0 até p_6.
9	p_{16}	Nenhuma	Mantido no DPB, pois é referência para p_{17}, p_{18}, p_{20}, p_{24}.
10	p_{12}	Nenhuma	Mantido no DPB, pois é referência para p_{10}, p_{11}, p_{13}, p_{14}.
11	p_{10}	Nenhuma	Mantido no DPB, pois é referência para p_9, p_{11}.
12	p_9	p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9	p_0 é liberado pois seus dependentes já foram processados. Demais são liberados conforme ordem de apresentação.
13	p_{11}	p_{10}, p_{11}	Liberados, pois seus dependentes já foram processados.
14	p_{14}	Nenhuma	Mantido no DPB, pois é referência para p_{13} e

			p_{15}.
15	p_{13}	p_{12}, p_{13}	Liberados, pois seus dependentes já foram processados.
16	p_{15}	p_{14}, p_{15}	Liberados, pois seus dependentes já foram processados.

2.2.7 - Implementações de H.264 SVC

Existem no mercado duas implementações *open-source* de H.264 SVC:

- JSVM (*Joint Scalable Video Model*): projeto criado pelo *Joint Video Team* (JVT) do *ISO/IEC Moving Pictures Experts Group* (MPEG) em conjunto com o *ITU-T Video Coding Experts Group* (VCEG) com o objetivo de ser uma prova de conceito do padrão SVC, sendo portanto considerado uma implementação de referência. Suporta as plataformas Windows e Linux. Sua versão atual é 9.19-7. É uma suíte que contém diversas ferramentas, na forma de programas executáveis, sendo as principais:
  - *H264AVCEncoderLibTestStatic*, o codificador SVC;
  - *H264AVCDecoderLibTestStatic*, o decodificador SVC;
  - *BitStreamExtractorStatic*, ferramenta que permite escalonar um vídeo através da extração de um *stream* que contém um subconjunto das camadas do *stream* original;
  - *DownConvertStatic*, reduz (*downscaling*) ou aumenta (*upscaling*) a resolução de uma imagem no formato YUV;
  - *PSNRStatic*, calcula a PSNR entre duas sequências de imagens YUV.
- *OpenSVCDecoder*: projeto criado pelo *Institute of Electronics and Telecommunications of Rennes* (IETR) em 2006. Sua versão atual é 1.12. É uma biblioteca de decodificação apenas.

Foi encontrada uma implementação de código proprietário:

- *MainConcept SVC*: Solução proprietária, de código fechado, da empresa MainConcept. Projeto criado em 2008. Suporta as plataformas Windows, Linux e OSX. Sua versão atual é 1.5. É uma suíte que contém o codificador e o decodificador SVC, ambos disponíveis na forma de programas executáveis, de bibliotecas ou de filtros para Microsoft DirectShow.

O decodificador JSVM apresentou desempenho extremamente ruim, da ordem de 300 vezes mais lento que a solução proprietária MainConcept, mostrando-se inviável para

aplicações práticas de tempo real. Para um vídeo SVC de 3 camadas, aproximadamente 500 Kbit/s, 4CIF, o JSVM demorou em média 1 segundo para decodificar cada quadro em um PC Intel Core2 Duo T7500 2,2 GHz, 2 GB RAM, Windows Vista 32 bits / Linux Ubuntu 9.4 32 bits. O OpenSVCDecoder gerou resultados inesperados, conforme mostrado à diante.

O resultado da Tabela 3 foi verificado experimentalmente com uso do MainConcept. Um ponto notório é: o atraso real foi maior que o atraso estrutural, pois o decodificador retém o quadro não apenas por motivos de dependência de decodificação, mas também para garantir a correta ordem de apresentação.

A seguir estão mostrados os resultados de testes do OpenSVCDecoder. Em todos os casos considere um vídeo de 9 quadros, codificado pelo JSVM com apenas a camada base nas configurações gerais da Tabela 4:

Tabela 4 – Configurações do vídeo para teste do OpenSVCDecoder

```
FrameRate               30                  # Maximum frame rate [Hz]
FramesToBeEncoded       9                   # Number of frames (at input frame rate)
GOPSize                 8                   # GOP Size (at maximum frame rate)
IntraPeriod             8                   # Intra Period
IDRPeriod               8                   # IDR period (should be (GOP size*N))
```

A Tabela 5 apresenta o resultado da codificação pelo JSVM (*trace* da saída) com a configuração de atraso estrutural zero. Nesse modo, o OpenSVCDecoder decodificou apenas 8 quadros (de 9).

Tabela 5 – Codificação SVC com atraso estrutural zero

```
MaxDelay                0.0                 # Maximum structural delay [ms]

AU     0: I    T0 L0 Q0   QP 31   Y 37.1313  U 38.1510  V 42.5956     46640 bit
AU     1: P    T3 L0 Q0   QP 37   Y 37.0521  U 34.1555  V 41.9423      7216 bit
AU     2: P    T2 L0 Q0   QP 36   Y 34.8244  U 34.2728  V 41.0963     13288 bit
AU     3: P    T3 L0 Q0   QP 37   Y 34.8438  U 34.0157  V 41.1558      8432 bit
AU     4: P    T1 L0 Q0   QP 34   Y 34.5797  U 35.4641  V 41.5462     26864 bit
AU     5: P    T3 L0 Q0   QP 37   Y 34.6718  U 34.4839  V 41.1642      8336 bit
AU     6: P    T2 L0 Q0   QP 36   Y 33.9451  U 34.3392  V 40.6019     14416 bit
AU     7: P    T3 L0 Q0   QP 37   Y 34.1796  U 34.1215  V 40.5600      8320 bit
AU     0: I    T0 L0 Q0   QP 31   Y 37.6130  U 37.4899  V 40.7416     48104 bit
```

A Tabela 6 apresenta o resultado para atraso estrutural de 70 ms, que equivale a 2 quadros. O OpenSVCDecoder decodificou apenas 7 quadros (de 9).

Tabela 6 – Resultados de teste do OpenSVCDecoder: atraso estrutural de 70ms

```
MaxDelay                70.0                # Maximum structural delay [ms]

AU     0: I    T0 L0 Q0   QP 31   Y 37.1313  U 38.1510  V 42.5956     46640 bit
AU     1: P    T3 L0 Q0   QP 37   Y 37.0521  U 34.1555  V 41.9423      7216 bit
AU     4: P    T1 L0 Q0   QP 34   Y 34.5797  U 35.4641  V 41.5462     26864 bit
AU     2: B    T2 L0 Q0   QP 36   Y 35.1595  U 34.3468  V 40.9890     11552 bit
AU     3: B    T3 L0 Q0   QP 37   Y 34.9816  U 34.6199  V 41.4432      7008 bit
AU     5: P    T3 L0 Q0   QP 37   Y 34.7240  U 34.4492  V 41.0702      8376 bit
```

```
AU     0: I    T0 L0 Q0   QP 31   Y 37.6130  U 37.4899  V 40.7416     48104 bit
AU    -2: P    T2 L0 Q0   QP 36   Y 33.8609  U 35.4736  V 41.1876     13280 bit
AU    -1: B    T3 L0 Q0   QP 37   Y 34.7127  U 36.6284  V 42.0545      5816 bit
```

A Tabela 7 apresenta o resultado para atraso estrutural de 200 ms, que equivale a 6 quadros. O OpenSVCDecoder decodificou apenas 3 quadros (de 9).

Tabela 7 – Codificação SVC com atraso estrutural de 200ms

```
MaxDelay                200.0                  # Maximum structural delay [ms]

AU     0: I    T0 L0 Q0   QP 31   Y 37.1313  U 38.1510  V 42.5956     46640 bit
AU     1: P    T3 L0 Q0   QP 37   Y 37.0521  U 34.1555  V 41.9423      7216 bit
AU     0: I    T0 L0 Q0   QP 31   Y 37.6130  U 37.4899  V 40.7416     48104 bit
AU    -4: P    T1 L0 Q0   QP 34   Y 34.8066  U 35.7895  V 41.4933     22200 bit
AU    -6: P    T2 L0 Q0   QP 36   Y 34.6092  U 34.3751  V 40.3228     15544 bit
AU    -5: B    T3 L0 Q0   QP 37   Y 35.2628  U 35.1025  V 41.3216      7320 bit
AU    -2: B    T2 L0 Q0   QP 36   Y 34.8940  U 35.5208  V 41.5174     10584 bit
AU    -3: B    T3 L0 Q0   QP 37   Y 35.4030  U 35.1718  V 41.4602      7400 bit
AU    -1: B    T3 L0 Q0   QP 37   Y 35.0729  U 36.5981  V 42.3230      5808 bit
```

A Tabela 8 apresenta o resultado para atraso estrutural de 1000 ms, que equivale a 30 quadros. O OpenSVCDecoder decodificou apenas 2 quadros (de 9).

Tabela 8 – Codificação SVC com atraso estrutural de 1000ms

```
MaxDelay               1000.0                  # Maximum structural delay [ms]

AU     0: I    T0 L0 Q0   QP 31   Y 37.1313  U 38.1510  V 42.5956     46640 bit
AU     0: I    T0 L0 Q0   QP 31   Y 37.6130  U 37.4899  V 40.7416     48104 bit
AU    -4: P    T1 L0 Q0   QP 34   Y 34.8066  U 35.7895  V 41.4933     22200 bit
AU    -6: P    T2 L0 Q0   QP 36   Y 33.6732  U 34.4999  V 40.5495     16032 bit
AU    -7: P    T3 L0 Q0   QP 37   Y 33.6237  U 34.5122  V 40.8412      9776 bit
AU    -5: B    T3 L0 Q0   QP 37   Y 34.6464  U 35.3123  V 41.3740      7424 bit
AU    -2: B    T2 L0 Q0   QP 36   Y 34.8940  U 35.5208  V 41.5174     10568 bit
AU    -3: B    T3 L0 Q0   QP 37   Y 35.3930  U 35.4206  V 41.5180      6968 bit
AU    -1: B    T3 L0 Q0   QP 37   Y 35.1198  U 36.5413  V 42.2575      6000 bit
```

Conclusão: O próprio vídeo de exemplo do IETR gera problema de decodificação. São entregues 488 de 500 quadros. No arquivo de configuração de exemplo no *Wiki* do IETR temos MaxDelay diferente de zero, mas o decodificador parece não suportar esse recurso.
Na configuração mais simples, onde MaxDelay vale 0, o decodificador entrega 8 quadros. A versão 1.03 decodificou 8 quadros, porém versões mais recentes decodificaram apenas 2.

No *Wiki* do IETR temos uma tabela que informa que o decodificador não suporta codificação de entropia CABAC (*Context-Adaptive Binary Arithmetic Coding*) nem quadros B na camada base. Porém, no próprio exemplo do tutorial do IETR, temos uma configuração de camada base com uso explícito do CABAC e atraso maior que 0, o que resulta em quadros B.

O gráfico da Figura 12 mostra os resultados de testes do MainConcept SVC em um PC Intel Core2 Duo T7500 2,2 GHz, 2 GB RAM, Windows Vista 32 bits.). As margens de erro dizem respeito ao intervalo de confiança de 95% relativo aos dados.

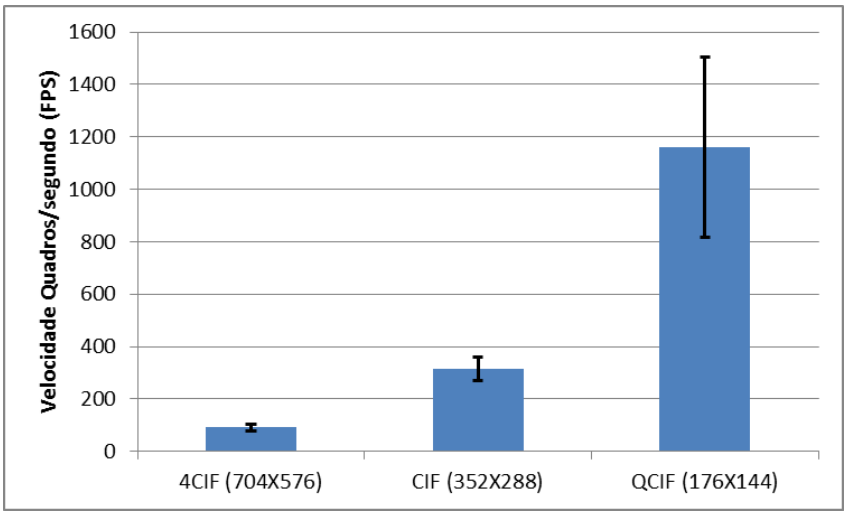


Figura 12 – Desempenho do MainConcept SVC Decoder

3 - DISTRIBUIÇÃO DE VÍDEO SOBRE REDES P2P

3.1 - INTRODUÇÃO

Esse capítulo apresenta, inicialmente, uma visão geral de redes par-a-par. Em seguida, é dedicado um espaço para estudo comparativo dos sistemas projetados para distribuição de fluxo contínuo em larga escala. O *Pastry* é apresentado com mais detalhes juntamente com o *Scribe* – sistema de gerência de grupos em topologia árvore criado sobre a rede *Pastry*.

3.2 - REDES PAR-A-PAR (P2P, Peer-to-Peer)

Para iniciar o estudo de distribuição de vídeo sobre redes par-a-par (P2P), é importante antes definir o significado do termo P2P. Algumas arquiteturas são claramente P2P e outras claramente Cliente-Servidor. Porém algumas arquiteturas podem ser classificadas como P2P ou como Cliente-Servidor a depender a definição do termo sendo considerada.
Em uma rede P2P os nós participantes possuem atribuições iguais ou muito semelhantes e a comunicação ocorre de muitos para muitos, não havendo uma evidência clara de cliente e servidor. A existência de entidade centralizada não descaracteriza um sistema P2P. Entretanto, caso a maioria dos nós comportem-se como clientes o sistema não pode ser considerado P2P [1].

A principal característica das redes par-a-par é a escalabilidade: quanto maior a quantidade de nós participantes, maior a capacidade da rede em prover o serviço que lhe foi proposto.
Funções pertinentes a todo sistema P2P:

- Descoberta de nó (*Peer discovery function)*: para ingresso na rede, as entidades interessadas (“nós externos”) precisam antes conhecer algum ou alguns dos membros da rede. A função de descoberta permite que essas entidades possam localizar nós da rede para então realizar a conexão;
- Inscrição na rede (*Enrollment function)*: função que trata da autenticação e da autorização de novos nós à rede. A afiliação (*join*) diz respeito à obtenção de credenciais para ingresso na rede.

Além das funções acima, o sistema P2P pode também prover funções como: indexação de dados, armazenamento de dados, computação, transporte de mensagem, dentre outras.
Os sistemas P2P são caracterizados em três dimensões de análise [2]:

- Auto-organização (*self-organization*), que diz respeito ao tratamento de falhas, de entrada e de saída de nós;
- Comunicação simétrica, vez que os nós atuam como clientes e servidores;

- Controle distribuído, no que tange aos protocolos e algoritmos descentralizados que mantêm o sistema P2P.

Redes notáveis amplamente destacadas na literatura são: Gnutella [20], Freenet [21], Pastry [15], Tapestry [22], Chord [23], Content Addressable Network (CAN) [13], pSearch [24], Scribe [14], Bayeux [12], Edutella [25]. Essas e outras redes serão mais bem apresentadas nas seções seguintes, com maior enfoque nas características relevantes à adequação para transferência de fluxo contínuo de mídia.

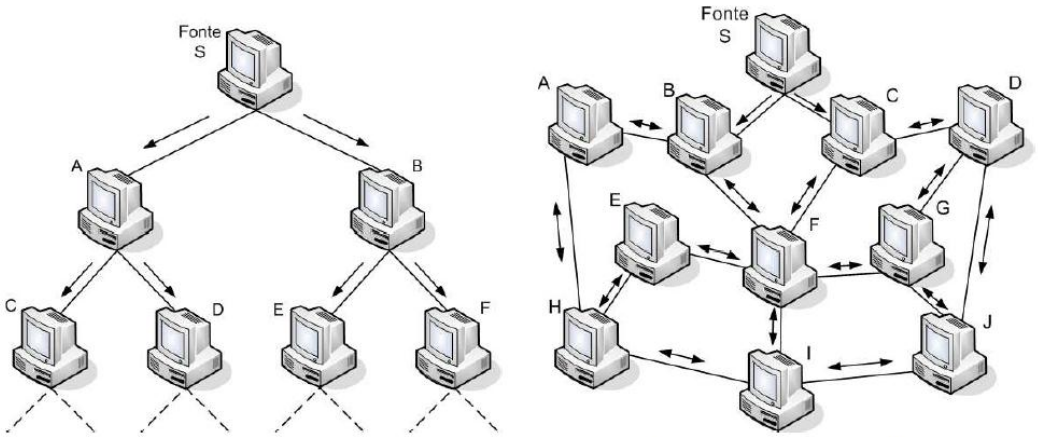
3.3 - REDES P2P PARA DISTRIBUIÇÃO DE VÍDEO DE FLUXO CONTÍNUO

Existem duas importantes topologias de sistemas P2P utilizadas para distribuição de fluxo contínuo vídeo: árvore (*tree-based*) e malha (*mesh*).

A diferença reside na forma como os nós se organizam para encaminhar o fluxo de dados. Na primeira são formadas relação de pai e filho entre os nós, de forma que o pai encaminha informação de vídeo para seus filhos e este recebe informação apenas do seu pai.

Na segunda arquitetura, em malha, os nós compartilham pedaços (*chunks*) de vídeo que estão espalhados na rede. Os nós não possuem função específica e podem receber e enviar informação de vídeo para quaisquer outros nós.

A figura abaixo exemplifica as arquiteturas em árvore e em malha.



(a) A arquitetura em árvore. (b) A arquitetura em malha.

Figura 13 – Topologias P2P para distribuição de vídeo. Fonte: [8]

A arquitetura em árvore é semelhante ao modelo adotado pelo IP Multicast [9]. Nessa arquitetura o fluxo de vídeo é "empurrado" (*push*) para os nós receptores, no sentido do pai para os filhos. A maior vantagem da arquitetura em árvore é a baixa latência e a baixa

sobrecarga de controle quando a taxa de entrada e saída de nós é baixa, pois, uma vez construída a árvore, a comunicação dar-se-á de pai para filho sem necessidade de roteamento.

Nas arquiteturas em malha o esquema de transmissão convencionalmente adotado é o *pull*, no qual cada nó "puxa" a informação que deseja receber de um ou mais nós vizinhos. A diferença notável entre a arquitetura em árvore e a arquitetura em malha é que esta última não mantém uma estrutura explícita para encaminhamento do fluxo de vídeo. Por esse motivo, as redes em malha são também chamadas de enxame (*swarm*). Elas toleram melhor a dinâmica de entrada e saída de nós participantes, penalizando, entretanto, o custo de encaminhamento do fluxo de vídeo dentro da rede devido à sobrecarga de controle do esquema *pull*.

Uma arquitetura híbrida possível seria a composição de múltiplas árvores disjuntas para um mesmo conjunto de nós, permitindo redundância de conexão e balanceamento de carga. O encaminhamento do fluxo de vídeo sobre múltiplas árvores torna a distribuição mais robusta à saída e à falha de nós. Busca-se, nesse esquema, explorar a diversidade de caminhos existentes na rede. Cada árvore transmite parte (*stripe*) do fluxo de vídeo.
Podemos elencar as seguintes características das arquiteturas em árvore única (*single-tree*), múltiplas árvores e malha [4] [3].

Arquitetura em árvore única:

- Topologia estática;
- Muito vulnerável à dinâmica de entrada e saída de nós (*peer churn*);
- Os nós folhas não colaboram no processo de encaminhamento do fluxo (*push*), pois eles apenas recebem dados, mas não enviam;
- O número de níveis da rede (profundidade da árvore) não pode ser alto para não causar grande atraso (*delay*) na transmissão ao vivo;
- A largura de cada nível (*fan-out*) é limitada pela capacidade de envio (*upload*) do nó pai;
- A construção e manutenção da árvore por ser à moda centralizada ou distribuída. À moda centralizada pode ser uma opção bem justificada pela existência necessária do servidor central de distribuição do vídeo (raiz da árvore).

Arquitetura em múltiplas árvores:

- O nó pode ter diferentes posições em diferentes árvores da rede;
- Os nós folhas podem colaborar no processo de encaminhamento do fluxo, vez que eles podem ser nós internos (não folha) em outra árvore;

- Melhor aproveita a capacidade de banda passante disponível nos enlaces da rede, pois o número de árvores em que o nó pertence pode ser ajustado conforme sua capacidade de transferência de dados.

Arquitetura em malha:

- A topologia dinâmica e a maior quantidade de adjacências (múltiplos vizinhos) permitem melhor robustez à dinâmica de entrada e saída de nós (*peer churn*);
- A dinamicidade na atualização da topologia da rede torna a eficiência da distribuição de vídeo menos previsível. Os pacotes podem chegar por diferentes rotas. O atraso (*delay*) pode variar rapidamente, resultando em *jitter* maior;
- O nó pode enviar e receber vídeo de múltiplos vizinhos simultaneamente. Se um vizinho falha, o vídeo ainda pode ser obtidos dos vizinhos remanescentes;
- A aplicação simples do esquema *push* gera alguns problemas:
    - O nó pode, cegamente, "empurrar" um pedaço de vídeo para um nó que já possui a informação;
    - Pode ainda ocorrer de dois nós "empurrarem", simultaneamente, o mesmo pedaço para um mesmo nó destino.

Como solução, os nós podem fazer o agendamento do "empurrão".

- A utilização do esquema *pull*, que é feito com auxílio da troca de mapa de memória (*buffers maps*), é uma alternativa ao *push*, porém gera grande sobrecarga (*overhead*) de controle, além da sobrecarga própria do processo de solicitação (*pull*) de pedaços de vídeo.

A estratégia "requisição-resposta" do *push* é naturalmente pouco interessante para transmissão de fluxo contínuo ao vivo. Ainda, é conhecido na literatura que o SVC introduz uma complexidade na distribuição de *chunks* em malha devido à dependência entre as camadas. Por esses motivos e considerando também o baixo *jitter* e a maior estabilidade da arquitetura em árvore, a arquitetura sendo proposta baseia-se neste esquema.

Sistemas P2P utilizados para fluxo contínuo de mídia:

Anysee [26] é uma rede em malha proposta em 2006. Rotas de transmissão são configuradas sobre a malha, de forma que não há processo de *pull*. Cada nó mantém em memória o percurso ativo para transmissão de dados além um conjunto de percursos reservas a serem utilizados caso o percurso ativo falhe na entrega do pedaço de vídeo. A

falha é detectada se a transmissão não ocorrer dentro de um intervalo de tempo. Não foi encontrada implementação de referência.

BitTorrent [4] é uma rede em malha proposta por Bram Cohen para compartilhamento de arquivos. É considerado um dos protocolos mais comuns de P2P. Em fevereiro de 2009, 27 a 55% de todo o tráfego de internet foi ocupado por transmissão BitTorrent. A versão oficial do protocolo pode ser encontrada em http://www.bittorrent.org/beps/bep_0003.html. Devido à simplicidade do protocolo original, diversos melhoramentos e extensões têm sido propostos e estão compilados no endereço eletrônico http://wiki.theory.org/index.php/BitTorrentSpecification. A maioria dos aplicativos de BitTorrent são aderentes a esta última especificação. A tecnologia tem sido aplicada também à transmissão de fluxo contínuo de mídia. O servidor de registro (*tracker*) informa quais nós na rede possuem o conteúdo desejado. É feita a conexão com um ou mais desses nós e, com base em várias heurísticas, realiza-se o *pull* para obtenção dos *chunks* de vídeo.

PPLive [27], também baseado na arquitetura em malha, é um sistema proprietário popular na China. Foi proposto em 2007. Possui atraso de quase um minuto devido ao processamento de *buffer* nos nós da rede e no reprodutor de vídeo (*player*). O sítio do PPLive está publicado no endereço eletrônico http://www.pplive.com.

O sistema Coolstreaming/DONet [28], baseado em malha, foi proposto em 2005. É similar ao PPLive no que tange ao registro de nó, descoberta e distribuição de *chunks*. Escrito em Python. Baseia-se em algoritmos de fofoca (*gossip algorithms*) para construir grafos de conexão para distribuição de conteúdo. Parou sua atividade ainda em 2005 devido a problemas de *copyright*. Foi o primeiro sistema P2P a conseguir um milhão de usuários. Tornou-se alicerce para produtos da empresa Roxbeam Corp, a qual lançou canais de conteúdo ao vivo junto com Yahoo Japan em 2006. Existe uma implementação alternativa em Java disponível no endereço http://code.google.com/p/coolstreaming, porém não há documentação do código-fonte.

SopCast [29], também em malha, proposto em 2007, possui tempo de arranque que varia entre 15 segundos até 3 minutos. O tempo de arranque, *start-up delay* [4] ou *bootstrap time* [6] é o tempo transcorrido desde que o usuário sintoniza/escolhe o conteúdo até que a reprodução do vídeo inicie. Sua implementação é gratuita, porém não *open-source*. A documentação é escassa e está disponível apenas em chinês.

SplitStream [30] é uma proposta de sistema em arquitetura de múltiplas árvores. Propõe resolver o problema da não contribuição dos nós folhas em redes de árvore única. É parte integrante do projeto FreePastry e basea-se no sistema Pastry [15], criado por Antony

Rowstron, Principal Researcher do centro de pesquisa Microsoft Research, Cambridge, UK. Rowstron atualmente trabalha com Windows DRT (*Distributed Routing Table*), semelhante à API (*Application Program Interface*) KBR [31] do Pastry, e possui contribuições no projeto dos sistemas LiveStation, proprietário, e Scribe [14].

ChunkySpread [32], proposto em 2006, e CoopNet [33], proposto em 2004, seguem arquitetura em múltiplas árvores e propõem resolver a forte dependência do nó filho com seu nó pai. Não foi encontrada implementação de referência na internet a respeito do ChunkySpread. O CoopNet foi projetado para codificação de vídeo com múltiplos descritores (MDC). Sua arquitetura considera servidor central e redundância de fonte de conteúdo. Assemelha-se às abordagens do tipo CDN (*Content Distribution Network*). O sistema vale-se do modelo clássico cliente-servidor quando das condições normais da rede. Os nós podem, entretanto, trocar informações em caso de congestionamento do servidor central. O projeto está hospedado na Microsoft sob o endereço eletrônico http://research.microsoft.com/en-us/um/people/padmanab/projects/coopnet, porém o status é inativo.

ESM (*End System Multicast*) [11] cria uma rede em malha através da seleção de enlaces baseada no RTT (*Round-Trip Time*). Sobre a malha é construída uma árvore para distribuição do fluxo de vídeo. O ESM é referência em vários estudos. O código-fonte é escrito em C/ObjectiveC e publicado sob a licença BSD. O projeto, entretanto, foi publicamente descontinuado e está sob domínio da empresa Conviva. Não há informações técnicas a respeito da nova vertente do projeto.

Nice [34], também orientado a arvore, baseia-se na informação de RTT para definir uma hierarquia entre os nós da rede. Os nós mantêm informações detalhadas dos vizinhos próximos (em termos de hierarquia). Informação global não é necessária. Não foi encontrada implementação de referência.

Os sistemas PPStream (http://www.ppstream.com), proprietário, Overcast, baseado em árvore única, Prime e Chainsaw, ambos baseados em malha e Bullet, baseado em múltiplas árvores, são citados em [3], porém não foi encontrada implementação de referência dessas redes.

Outras redes consideradas na literatura porém com pouca ou nenhuma informação técnica a respeito de implementação são: Zattoo, proprietária; PROMISE (2003); CollectCast (2003); CFS (2001); Chord (2001); PAST (2001); CAN (2001); Pixie (2002); ZigZag (2003); Narada; QQLive, proprietária; TVUnetworks (2005), proprietária; Tribler, escrita em Python e projetada para VoD; IceShare, escrita em Python e descontinuada; MediaBlog, sob GPL, traz idéias do PeerCast e BitTorrent, projeto parado desde 2006;

ForceP2P, proprietária; Alluvium (2003), GPL (*GNU General Public License*), em malha, com servidor central e idéias baseadas na abordagem CDN, sítio eletrônico indisponível; PeerCast (2002), inativo deste 2007; FreeCast (2004), escrito em Java, parado desde 2007; Joost (2006), comprado em 2009.

PULSE [35], baseado em malha, com suporte a incentivos semelhante ao BitTorrent, é projetado para transmissão ao vivo de vídeo. Existe código-fonte de referência, escrito em Python e Zope, disponível no endereço eletrônico http://www.napa-wine.eu/cgi-bin/twiki/view/Public/PULSE.

A empresa Oracle possui também a especificação JXTA (*Juxtapose*) composta por uma suíte de protocolos relacionados a P2P, a citar: Peer Resolver Protocol, Peer Information Protocol, Rendezvous Protocol, Peer Membership Protocol, Pipe Binding Protocol, Endpoint Routing Protocol. O projeto TVP2P implementa JXTA. A especificação mudou de nome para JXSE e foi lançada para compatibilidade com Java 5 em 2007. A grande desvantagem desta solução é que sua abordagem é ampla, no sentido de definir um sistema genérico de P2P, não havendo, portanto, otimização para o escopo de transmissão de fluxo contínuo.

DistribuStream é uma implementação, em Ruby, GPL, do protocolo PDTP (*Peer Distributed Transfer Protocol*, http://distribustream.org/specification), similar ao BitTorrent. Versão em C, de 2003, está disponível em http://libpdtp.sf.net/. Iniciativas de interfaces gráficos foram iniciadas em 2003 (http://skyfire.sf.net) e 2004 (http://squall.sf.net) porém não concluídas até então.

FreePastry é um projeto criado em 2001 e mantido por Jeff Hoye, um dos idealizadores. Vem sendo usado em projetos recentes, como [36]. É composto da implementação de várias propostas que unidas formam um sistema P2P único, integrado e dotado de diversas funcionalidades. Sua evolução é evidente e os melhoramentos estão registrados no sítio do projeto - http://www.freepastry.org. Os projetos que compõem o sistema FreePastry são:

- Pastry [15]: infraestrutura para formação de rede na arquitetura em malha, de alta escalabilidade e completamente descentralizada;
- Scribe [14] [37]: rede de sobreposição em arquitetura árvore construída sobre o Pastry;
- SplitStream [30]: rede de sobreposição em múltiplas árvores; construída sobre o Scribe para permitir melhor aproveitamento dos nós folhas da árvore Scribe;
- Squirrel: cachê web cooperativo construído sobre o Pastry;
- PAST: banco de dados distribuído em arquitetura DHT (*Distributed Hash Table*) sobre o Pastry;

- POST: sistema cooperativo de distribuição de mensagem instantânea; construído sobre o Pastry e útil para criação de soluções de *e-mail*, calendários compartilhados, *whiteboards*, dentre outras, sem necessidade de servidor central;
- Scrivener: compartilhamento de arquivos baseado em incentivos. Também construído sobre o Pastry;
- PASTA: sistema de arquivos escalável; construído sobre o Pastry;
- Pastiche: ferramenta de backup seguro sobre o Pastry;
- Glacier, GCPast, etc.

O FreePastry é implementado em Java sob licença BSD (*Berkeley Software Distribution*). Todo o código-fonte é orientado a eventos e baseado no padrão de desenvolvimento (*design pattern*) de nome *Continuation*, o qual define uma interface comum para criação de estruturas de invocação assíncrona de métodos. A programação, também orientada a teste (TDD, *Test Driven Design*), usa o *framework* jUnit. Todo o código-fonte está razoavelmente documentado em JavaDoc. Tutoriais técnicos a respeito da implementação e da API do Pastry, do Scribe e do SplitStream estão disponíveis em https://trac.freepastry.org/wiki/FreePastryTutorial. As publicações referentes aos sistemas embarcados no FreePastry estão no sítio da Microsoft http://research.microsoft.com/en-us/um/people/antr/PASTRY/pubs.htm.

Conforme apresentado, observa-se que o sistema Pastry é bastante flexível na sua qualidade de possibilitar a implementação das mais diversas funcionalidades, como replicação, banco de dados, balanceamento de carga, cachê, dentre outras, mostrando ser uma solução genérica e robusta para os mais variados fins. Soma-se também outras importantes características do Pastry como descentralização completa, escalabilidade, auto-organização e tolerância a falhas. Essas características serão detalhadas em seção seguinte. Todas essas vantagens, aliada à disponibilidade de código-fonte *open-source*, atualizado e documentado, elege o Pastry a opção a ser considerada na arquitetura deste trabalho.

3.4 - VIDEO ESCALÁVEL EM REDES P2P

A proposta [38] trata a transmissão de Vídeo sob Demanda (VoD, *Video on Demand*) codificado no formato *aceSVC* – SVC baseado em *wavelet* – sobre uma rede P2P BitTorrent, cuja implementação utilizada é o *Tribler*. O vídeo é divido em pedados contendo um GOP. A adaptação de taxa é obtida com base no controle de *buffering* através de janela deslizante.

Em [39], é proposta uma transmissão de vídeo dividido em pedaços, *chunks*, sobre uma rede em malha. Define-se *QS* o conjunto de todas as possíveis escalabilidades. Para cada escalabilidade são mapeadas as seguintes características:

- Resolução
- Complexidade de processamento
- Taxa (*bitrate*)

O esquema adaptativo ocorre em duas fases:

1ª Fase - Initial Quality Adaptation (IQA):

- Passo 1) Retirar elementos de *QS* onde: resolução > resolução do dispositivo;
- Passo 2) Retirar elementos de *QS* onde: processamento > processamento disponível no dispositivo;
- Passo 3) Retirar elementos de *QS* onde: *bitrate* > *bitrate* disponível para o dispositivo;
- Passo 4) Selecionar um elemento (escalabilidade) do conjunto restante conforme preferência do usuário.

2ª Fase - Progressive Quality Adaptation (PQA):

- Passo 1) Retirar elementos de *QS* onde: *bitrate* > *bitrate* disponível para o dispositivo;
- Passo 2) Retirar elementos de *QS* onde: processamento > processamento disponível no dispositivo.

A escalabilidade espacial é fixada no inicio da transmissão, pois se considera que a variação desta provoca grandes distorções. Apenas blocos (*chunks*) pertencentes às escalabilidades selecionadas pelos algoritmos IQA/PQA são considerados. A prioridade é dada pela fórmula

$$\text{Priority}(\text{Block}(t, D, T, Q)) = -A\ t - B(a\ D + b\ T + c\ Q)$$

onde *t* é o tempo de *playback* (que é altamente priorizado por A). Nós que querem *playback* mais suave podem usar *A* mais alto. Nós que querem melhor qualidade SNR podem usar B mais alto.

Nessa proposta, existe um controlador (*tracker*) semelhante ao BitTorrent. Cada nó se conecta no *tracker* e obtém potenciais vizinhos. Para suportar SVC, o *tracker* agora informa o nível (*layer*) que cada nó está transmitindo.

Uma desvantagem da proposta [39] reside no cálculo prático da complexidade computacional da decodificação (isto é, do custo de processamento) de cada nível de

escalabilidade. Ainda, capacidade de processamento é uma característica muito dinâmica dos dispositivos da rede P2P. Entrar no escopo da análise de recursos computacionais dos nós traz à tona vários tópicos como memória disponível, *codecs* suportados, etc.

A arquitetura proposta neste trabalho visa dispensar esforços no estudo da escalabilidade com relação à capacidade de transmissão da rede P2P e não com relação aos dispositivos que compõem a rede.

Em [40] tem-se a abordagem com uso de FEC (*Forward Error Correction*) e retransmissão de pacote como forma de otimizar a performance de vídeo SVC em rede P2P organizada na forma de múltiplas árvores.

Em [41], os autores fazem um estudo do ganho na qualidade do vídeo SVC quando este é transmitido em rede P2P através de grupo *multicast* organizado em árvore. É apresentada uma descrição do algoritmo de formação da árvore assumindo que as bandas de *upload* e de *download* de cada nó são conhecidas. O artigo compara a qualidade do vídeo em termos de taxa média recebida (em bps) em cenário em que os nós são dispostos aleatoriamente com outro cenário onde os nós são dispostos na ordem do algoritmo – os de maior banda no início da árvore, ambos sem entrada e saída (*churn*) de membros. Os testes são realizados com emprego do simulador NS-2, considerando 500 nós. O segundo cenário apresentou uma taxa de recepção 40% maior. A solução foi considerada escalável em ambos os cenários. Todavia, a proposta não incorpora o controle de congestionamento.

3.5 - A REDE PASTRY

3.5.1 - Introdução

O Pastry foi criado por Antony Rowstron e Peter Druschel, pesquisadores membros da Microsoft Research. Projetado para larga escala, tem aplicações em variadas áreas, como armazenamento distribuído de dados (*global data storage*), compartilhamento de arquivos (*data sharing*) e comunicação em grupo. Pastry é completamente descentralizado, escalável e auto-organizável. Ele adapta-se de acordo com a entrada, saída e falha de nós.

Roteamento

O Pastry usa uma tabela de roteamento inspirada no modelo de Plaxton [42]. Dada uma mensagem e uma chave, o roteamento entrega a mensagem para o nó cujo *nodeId* é numericamente mais próximo da chave. O *nodeId* é o número único que identifica o nó na rede. Esse número é composto de l símbolos (dígitos), onde cada símbolo possui uma quantidade b de bits.

A cada iteração do roteamento, ou seja, a cada salto (*hop*) na rede, resolve-se um ou mais dígitos do *nodeId*. O número máximo de saltos é $log_{2^b}N$, onde *N* é o número máximo de nós ($N = 2 \cdot b \cdot l$). Assim, com ausência de falhas, o roteamento no Pastry ocorre em passos *O(log N)*.

Todo nó possui as seguintes informações de estado, conforme mostrado na Figura 14:

- *Leaf set*: Lista de nós para encaminhamento direto. Caso a chave da mensagem coincida com um dos *nodeIds* deste conjunto, então é realizada a entrega direta;
- *Routing table* (R): Tabela para encaminhamento com mais de um salto;
- *Neighborhood set*: Lista de nós para conversação objetivando atualização de estado.

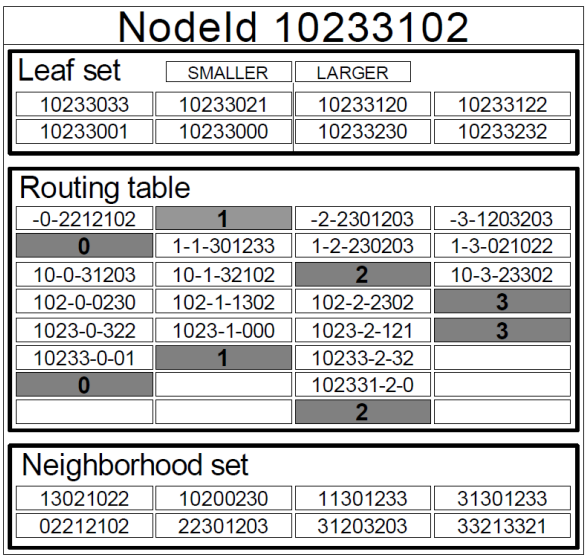


Figura 14 – Estado hipotético de um nó Pastry de *nodeId* 10233102, base 4 (*b* = 2) e *l* = 8.
Fonte: [15]

A cada salto, o nó deve resolver um ou mais dígitos da chave. A tabela de roteamento é organizada para facilitar essa resolução. Suponha que um determinado nó precise resolver o **terceiro** dígito da chave 10**1**43001. O nó em questão selecionará a **terceira linha** da tabela de roteamento. Dado que o dígito a ser resolvido é o símbolo **1**, que corresponde ao **segundo** símbolo da base 4 (*b* = 2 bits), então seleciona-se a **segunda coluna** da tabela de roteamento. A célula resultado, cujo, no exemplo, tem *nodeId* = 10132102, indica o próximo salto no processo de roteamento. Caso a célula esteja vazia, o nó em questão considera que ele é o destinatário final, ou seja, ele é o nó de *nodeId* numericamente mais próximo da chave, finalizando-se assim o processo de roteamento.

Não está sendo mostrada na tabela de roteamento da Figura 14 a identificação física (IP, por exemplo) dos nós na rede sobre a qual se estabelece o Pastry.

A quantidade de linhas da tabela de roteamento é exatamente o número de símbolos do *nodeId*. A quantidade de colunas é a base utilizada para representar o símbolo. Vale observar que, portanto, a tabela de roteamento não cresce com o aumentar do número de nós presentes da rede. Essa característica garante alta escalabilidade da solução.

O conjunto *neighborhood set* possui a função de manter a tabela de roteamento atualizada. Com determinada frequência, o nó contata os nós do *neighborhood set* para troca das informações de estado. Ao obter as tabelas de roteamento dos vizinhos, faz-se então um processo de mesclagem para obter uma tabela resultado mais eficiente conforme o critério de seleção. O Pastry, por padrão, seleciona *nodeId* de menor RTT, que é um critério de proximidade baseado em distância temporal.

Localidade

Uma propriedade consequente do processo de roteamento acima descrito é a localidade: a cada salto, o próximo nó está mais distante conforme o critério de proximidade escolhido. Se o critério for RTT, então, em média, cada salto vai para um nó geograficamente mais distante. Esse princípio é verdadeiro se os *nodeIds* são distribuídos uniformemente e para um valor de N suficientemente alto.

Na primeira linha da tabela de roteamento qualquer *nodeId* é válido. Na segunda linha somente os *nodeIds* cujo primeiro símbolo coincide com o do nó em questão são válidos. Na terceira linha somente os *nodeIds* cujos dois primeiros símbolos coincidem com o do nó em questão são válidos. Assim, no processo de atualização da tabela de roteamento através da troca de contexto entre os nós do *neighborhood set*, o nó tem uma liberdade maior de escolha de *nodeId* quanto menor o número da linha na tabela de roteamento. Dessa forma, para uma quantidade grande de nós no sistema, é provado que os vizinhos de menor RTT tendem a estar nas primeiras linhas da tabela de roteamento. Também, os vizinhos de maior RTT tentem a estar nas últimas linhas. Isso resulta numa propriedade interessante do esquema de roteamento do Pastry: os saltos estão associados a distâncias (conforme a métrica de proximidade) cada vez maiores no decorrer do processo de encaminhamento da mensagem, conforme é mostrado na Figura 15. Esse comportamento é contrário à distância numérica entre os *nodeIds* a cada salto, conforme mostrado na Figura 16.

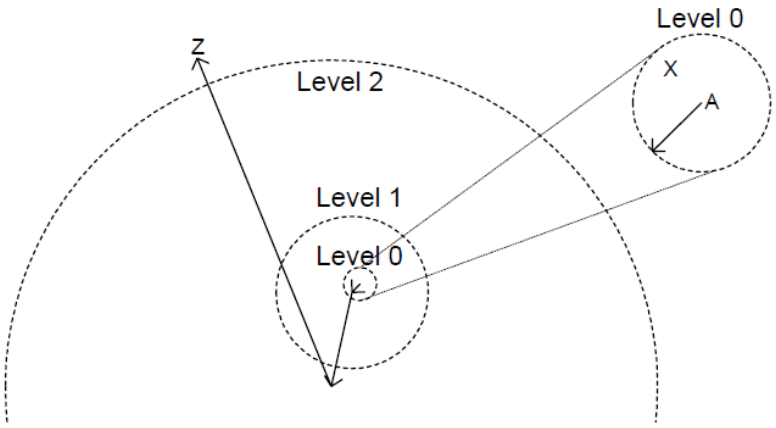


Figura 15 – Representação da distância geográfica em cada salto (*level*) do processo de roteamento do Pastry. Fonte: [15]

Roteamento representado no espaço de distribuição circular dos *nodeIds*:

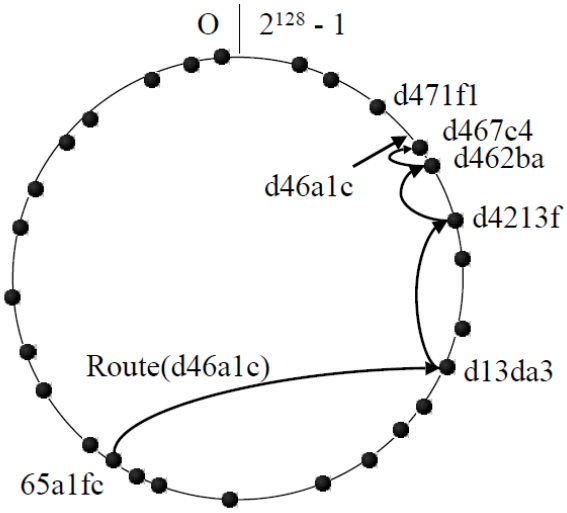


Figura 16 – Exemplo de roteamento da mensagem originada no nó 65a1fc com chave d46a1c, base = 16 (*b* = 4). Fonte: [14]

Ingresso de nó na rede

Considere que o nó *X* deseja ingressar em uma rede Pastry formada por vários nós, dentre eles *A*, *B*, *C* e *D*. É necessário que *X* conheça um membro da rede para iniciar o processo de ingresso, que, neste exemplo, é *A*.

Inicialmente *X* sorteia um número aleatório para ser seu *nodeId*. A política de concessão de *nodeIds* não é escopo do Pastry.

Seja X_{Id} o *nodeId* de *X* e seja *route(msg, key)* a instrução do Pastry que faz o roteamento da mensagem *msg* para o nó de *nodeId* numericamente mais próximo da chave *key*. Para iniciar o ingresso na rede, *X* envia para *A* a instrução *route*($join_X$, X_{Id}), sendo $join_X$ uma mensagem que representa o desejo de *X* de fazer parte da rede.

A instrução *route*($join_X$, X_{Id}) será encaminhada na rede conforme as regras de roteamento explanadas. A mensagem $join_X$ apenas adiciona a semântica de que durante o roteamento cada nó enviará um retorno (*feedback*) diretamente para *X* contendo informações de seu estado. Suponha que o roteamento de $join_X$ ocorra conforme os saltos mostrados na Figura 17, sendo *D* o nó final, ou seja, D_{Id} é numericamente próximo de X_{Id}.

De posse então das tabelas de roteamento de *A*, *B*, *C* e *D*, o nó *X* compõe sua tabela de roteamento da seguinte maneira: seleciona-se a primeira linha da tabela de *A*, a segunda linha da tabela de *B*, a terceira de *C* e o restante de *Z*. O conjunto *neighborhood set* é obtido de *A* e o *leaf set* é obtido de *D*.

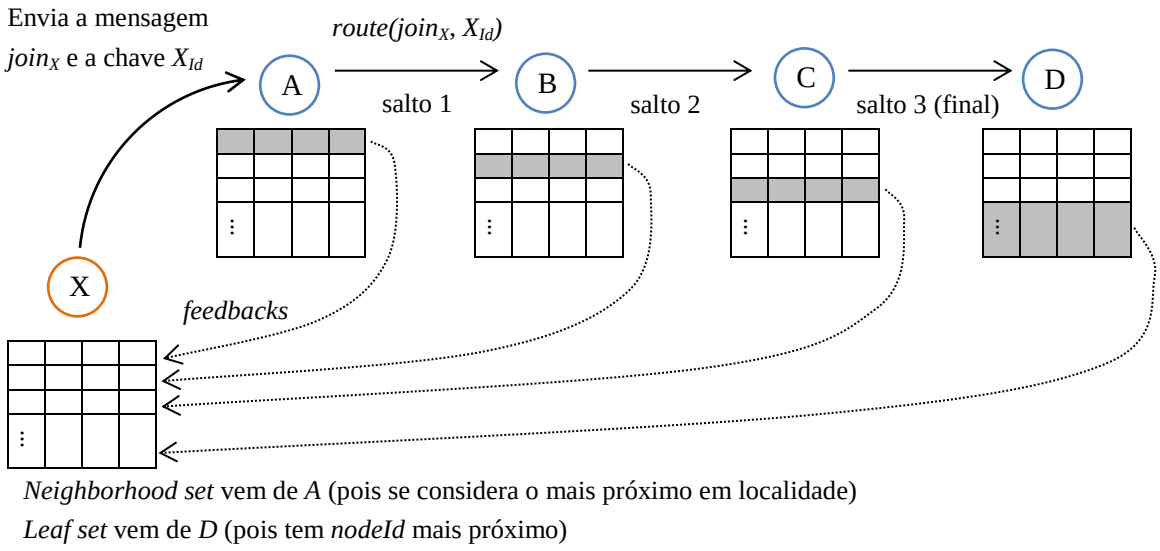


Figura 17 – Fluxo de entrada do nó *X* e construção de sua tabela de roteamento Pastry.

É fácil observar que: se *A* encaminhou a mensagem para *B*, então em *B*, a segunda linha da tabela de roteamento contém *nodeIds* cujo primeiro símbolo coincide com o *nodeId* de *X*, sendo portanto uma linha válida para a segunda linha da tabela de roteamento de *X*. Essa ideia é valida para todo o processo. Criar o estado inicial do nó *X* é quase que um processo trivial no Pastry.

Considera-se que o Pastry, juntamente com o Tapestry, Chord e CAN, formam uma segunda geração de redes P2P, pois eles garantem uma resposta definitiva para a busca na

rede com o número máximo de saltos determinado, mantendo a escalabilidade da rede (como faz o FreeNet) e a auto-organização (como faz o Gnutella e o FreeNet). Pastry e Tapestry são similares à proposta de Plaxton (1999) e Landmark (1988). Estes, porém, não são auto-organizáveis. Chord faz roteamento com base na diferença numérica entre *nodeIds* (ao invés do casamento do prefixo). Chord não trata localidade. CAN faz um roteamento em espaço d-dimensional. Sua tabela de rotas não cresce, porém o número de saltos é maior que $log_{2^b}N$.

3.5.2 - Gerência de Grupos: *Scribe*

O Scribe é uma infraestrutura de notificação de eventos orientada a tópicos construída sobre o Pastry. No Scribe, a difusão da informação ocorre segundo a topologia árvore. O Scribe é tolerante a falhas, escalável e auto-organizável. Sua proposta permite a criação de grupos *multicast* dentro da rede Pastry.

Alguns resultados importantes do Pastry são fundamentos para o projeto Scribe:

- Simulações mostram que, dada uma rede com topologia baseada no modelo Georgia Tech (1996), a distância média trafegada por uma mensagem é menor que 66% da maior distância entre a fonte e destino na rede subjacente (na qual há a sobreposição Pastry);
- Assume-se que dois nós com distância *d* entre eles transmitem mensagens com a mesma chave, tal que a distância entre cada nó e o *nodeId* numericamente mais próximo da chave é muito maior que *d*. Simulações mostram que a distância média trafegada por cada uma das mensagens antes da rota convergir é aproximadamente igual à distância *d* entre as respectivas fontes.

O Scribe não trata da confiança na entrega nem do ordenamento de eventos. No objetivo proposto neste projeto, que é de transmissão ao vivo de vídeo, o não tratamento desses dois requisitos é vantagem para arquitetura, vez que o Scribe propõe-se a não causar sobrecarga de processamento no controle de mensagens. Ambos os serviços de confiança e ordenamento de eventos não são interessantes para distribuição de fluxo contínuo de mídia, principalmente quando é tratada a abordagem ao vivo.

O Pastry é usado para criação de tópicos (grupos), para ingresso de assinantes (membros do grupo) e para difusão de eventos.

Os assinantes (*subscribers*) para ingressarem no grupo Scribe enviam uma mensagem (*subscribe)* via Pastry usando o *topicId* como chave, sendo *topicId* o *hash* do nome do grupo (*topic name*). O registro é memorizado em cada salto ao longo do roteamento dessa

mensagem. A lista de assinantes é mantida no nó de *nodeId* numericamente próximo do *topicId*. Este é o nó raiz do grupo, chamado de *rendezvous point*.

O publicador de conteúdo (aquele que deseja fazer difusão no grupo), envia uma mensagem de dados (*publish*) usando o *topicId* como chave. Essa mensagem chegará, obviamente, no *rendezvous point*, o qual encaminhará o dado através da árvore formada pelos caminhos reversos partindo dele (o raiz) até os nós assinantes, processo este chamado de *reverse path forwarding.*

O processo de saída de um assinante é semelhante ao ingresso *subscribe*. Para sair, o membro do grupo envia a mensagem *unsubscribe* com a mesma chave *topicId*. A única mudança é que, neste caso, o registro do caminho reverso é retirado da memória em cada salto ao longo do roteamento dessa mensagem.

A árvore de *multicast* formada pelos caminhos reversos do roteamento pode ser genericamente representada pela Figura 18, onde o círculo de maior raio é o *rendezvous point.*

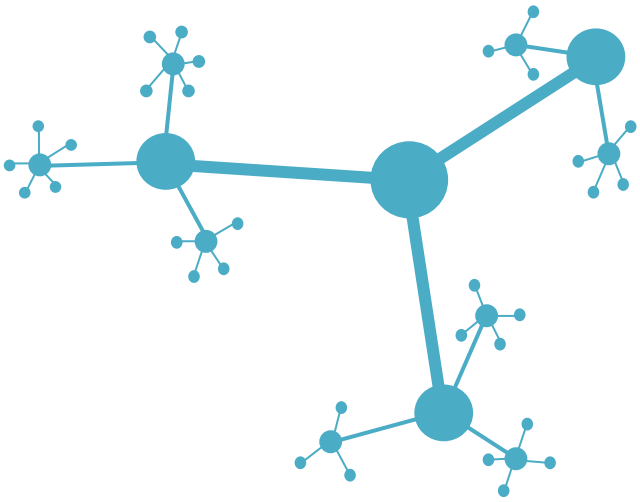


Figura 18 – Topologia árvore hipotética criada pelo Scribe

Manutenção da rede: periodicamente, cada nó não folha envia uma mensagem especial (*heartbeat message*) para seus nós filhos. Um filho detecta falha do nó pai quando não recebe esta mensagem por determinado período de tempo. Após detecção da falha, o nó executa novamente a instrução *subscribe* cuja chave é o *topicId* do grupo. O Pastry roteará a mensagem até o *rendezvous point*, reparando a árvore *multicast*.

O Scribe também é tolerante a falhas. As informações de gestão do grupo (lista de assinantes) são replicadas nos saltos próximos do *rendezvous point*. Se o *rendezvous point* falhar, automaticamente o nó de *nodeId* numericamente mais próximo do *topicId* será o

próximo *rendezvous point*. Esse resultado é natural do próprio algoritmo de roteamento do Pastry.

Se entrar no grupo um nó cujo *nodeId* é numericamente ainda mais próximo do *topicId* com relação ao *rendezvous point*, esse novo assumirá automaticamente o papel de *rendezvous point*.

O Scribe é comparado com os esquemas Narada e Overcast. A maior diferença é que o Pastry, utilizado como substrato para o Scribe, pode crescer para uma quantidade extremamente grande de nós devido à sua escalabilidade. Isso concede escalabilidade também para o Scribe.

O Bayeux, comparado com o Scribe, tem dois problemas de escalabilidade. (1) A raiz, no Bayeux, mantém informações de todos os membros. (2) Grande sobrecarga na rede, pois todo gerenciamento do grupo passa pelo nó raiz.

Análise de desempenho do Scribe, realizada em [14], mostra que o Scribe é, no pior dos casos, duas vezes mais lento que o IP Multicast, segundo a métrica RAD (*ratio between the average delay using Scribe and the average delay using IP Multicast*). De acordo com a métrica RMD (*ratio between the maximum delay using Scribe and the maximum delay using IP Multicast*), o Scribe é, no máximo, 4,26 vezes mais lento que o IP Multicast. A função de distribuição da sobrecarga de enlace é muito próxima do IP Multicast.

Uma maneira de melhorar o Scribe no sentido de obter árvores em que os nós de melhor banda estejam próximos à raiz é apresentada em [43]. O artigo analisa o comportamento de importantes jogos, incluindo World of WarCraft (WOW), o jogo online mais popular atualmente. O esquema baseia-se na formação de uma árvore Scribe de "primeiro nível" formada por nós considerados importantes, fisicamente distribuídos em longas distâncias e dotados de boa capacidade de banda. Árvores de "segundo nível" ficam penduradas em nós da árvore principal e são formadas por nós fisicamente próximos, como, por exemplo, dentro de um mesmo domínio. Esses nós possuem menor capacidade de banda. São utilizados algoritmos de *clustering* para identificação dos grupos de segundo nível. Nesses grupos, o algoritmo de seleção por proximidade do Pastry é ajustado para inferir a distância física entre os nós. O artigo não trata do problema de distribuição de banda na árvore de primeiro nível, entretanto a ideia mostra-se bastante eficiente, chegando a obter melhoramento de taxa em até 27 vezes.

4 - CONTROLE DE CONGESTIONAMENTO MULTIDESTINATÁRIO

4.1 - INTRODUÇÃO

Esse capítulo conceitua os protocolos de controle de congestionamento *TCP-Friendly* com respeito à taxonomia, agressividade, responsividade e justiça. Em seguida, o protocolo TFMCC é eleito como insumo para o módulo adaptativo da arquitetura proposta. Suas características e algoritmos são apresentados.

Tratar de transmissão de dados em larga escala é, de fato, considerar o escopo de uso da rede Internet. Nesse cenário, qualquer que seja o protocolo de transporte a ser utilizado no sistema, certamente o fluxo de dados concorrerá com o fluxo de conexões TCP (*Transmission Control Protocol*), vez que é um dos protocolos de maior penetração na Internet.

Já que a proposta de arquitetura é definir um sistema que possa ser utilizado na abrangência da rede mundial de computadores, então é conveniente que a camada de transporte coordene o fluxo de vídeo no sentido de obter comportamento *justo* quando da transmissão concorrente com fluxos TCP. O termo *justo* significa que a banda ocupada pelo fluxo de vídeo seja tão próxima quanto à de um fluxo TCP sob as mesmas condições de rede, a considerar perda de pacotes e RTT (*Round-Trip Time*) [44]. Os protocolos cuja taxa de transferência converge para o TCP sob mesmas condições de rede são ditos *TCP-Friendly.* Conforme definido em [45], um protocolo *TCP-Friendly* é aquele que objetiva consumir uma largura de banda de rede menor ou igual à do TCP.

A taxa de transferência controlada pela estratégia AIMD (*Additive Increase / Multiplicative Decrease*) no TCP muda severa e frequentemente, o que não é adequado para sistemas multimídia de fluxo contínuo de tempo real [45], tal como conversas de áudio e/ou vídeo. Vários esquemas semelhantes e baseados no AIMD têm sido propostos no intuito de minimizar esse problema. Tais esquemas podem ser classificados, conforme sua aderência ou proximidade de comportamento quando defrontados com o TCP, como *TCP-compatibility*, *TCP-equivalent* e/ou *TCP-equal share*.

A RFC (*Request For Comments*) número 2309 [46] introduz a definição de *TCP-compatibility*: um fluxo que, quando da rede em estado estacionário, deve consumir no máximo a mesma largura de banda de um fluxo TCP sob condições comparáveis de RTT e perda de pacotes. Uma rede em estado estacionário é aquela cujas variações de RTT e de perda de pacotes são desprezíveis durante um intervalo arbitrariamente longo de tempo.

Em termos práticos, para o escopo de Internet, arbitrariamente longo é entendido como a duração de alguns minutos. Observa-se, segundo [47], que o fluxo de dados na Internet experimenta múltiplas regiões de estado estacionário com duração na escala de alguns minutos e com intervalos de espaçamento mutuamente independentes e uniformemente distribuídos.

Um fluxo *TCP-equivalent* é aquele que obtém taxa igual ao fluxo TCP quando das mesmas condições de rede nos termos de RTT e de perda de pacotes.

A classificação *TCP-equal share*, criada por [45], define uma condição ainda mais rígida: um fluxo que obtém taxa igual quando *coexistir* com o fluxo TCP no mesmo enlace ou barramento.

Resultados [45] mostram que o protocolo TFRC (*TCP-Friendly Rate Control*), além de atender ambos os critérios *TCP-equivalence* e *TCP-equal share*, apresenta equilíbrio satisfatório de agressividade *(aggressiveness)*, responsividade *(responsiveness)* e justiça *(fairness)* quando comparado com diversas outras propostas, como GAIMP (*General Additive Inc. / Multiplicative Dec.*), IIAD (*Inverse Inc. / Additive Dec.*), SQRT (*Square-Root Inc. / Dec.*), SIMD (*Square Inc. / Multiplicative Dec.*), AIAD/H (*Additive Inc. / Dec. with History*), TFRCP (*TCP-Friendly Rate Control Protocol*) e TEAR (*TCP-Emulation at Receiver*).

A agressividade representa a característica do protocolo que diz respeito ao aumento da taxa de transmissão até antes do próximo evento de perda de pacote. Um protocolo muito agressivo é aquele que busca atingir a capacidade máxima de transferência da rede tão rápido quanto possível. A Figura 19 mostra que o TFRC e TEAR têm comportamento bastante suave e o que primeiro converge mais rapidamente à taxa média do fluxo TCP.

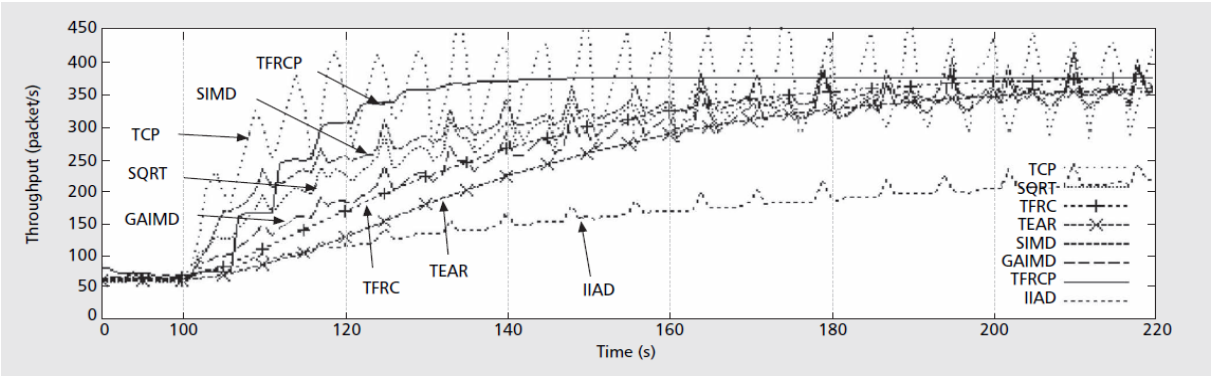


Figura 19 – Comparação de agressividade entre protocolos TCP e *TCP-Friendly*. Fonte: [45].

A responsividade indica o quão cauteloso é o protocolo com respeito à mudança nas condições da rede. Um protocolo de baixa responsividade não muda sua taxa de transferência imediata ou bruscamente logo após evento de perda de pacote (ou mudanças na medida do RTT). A Figura 20 mostra que o TFRC é responsividade porém sem perder a suavidade no controle da taxa de transmissão.

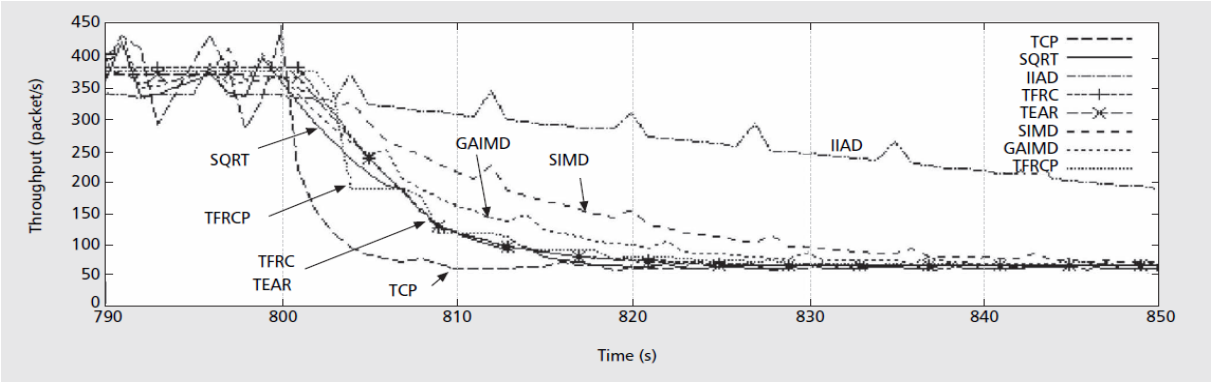


Figura 20 – Comparação de responsividade entre protocolos TCP e *TCP-Friendly*. Fonte: [45].

A política de justiça descreve a capacidade do protocolo em obter taxa de transferência equivalente ao TCP no estado estacionário da rede.

Os protocolos acima citados são todos projetados para enlaces ponto-a-ponto (*Unicast*). O protocolo TFMCC é uma versão multidestinatária do TFRC e sugerido em [48] para aplicações de cunho multimídia. Ainda, dos protocolos *TCP-Friendly* supra referenciados, o TFRC é o que obtém menor taxa média de perda de pacote. Essa característica é importantíssima para um sistema de transmissão ao vivo, onde não há espaço para retransmissão de pacotes perdidos. É, portanto, desejável que o protocolo de controle de congestionamento procure sempre evitar perdas.

O trabalho de [49] também considera o TFMCC como fundamento, devido à sua condição de maduro, testado e validado. Zhu cita os desafios de manter a justiça do protocolo quando concorrente com fluxos TCP e o problema de manter uma taxa mínima – requisito para aplicações multimídia – que é desprezado pela maioria dos protocolos de controle de congestionamento. Nesse trabalho é adotada uma solução intercamadas (*cross-layer*) através do ajuste da taxa de compressão do vídeo, considerando-se uma fonte com codificação em tempo real. Ou seja, é efeito um controle de fluxo associado ao controle de congestionamento oferecido pelo TFMCC, de maneira a evitar sobrecarga (*overflow*) ou esvaziamento de memória (*underflow*) no transmissor. O protocolo apresenta uma ideia interessante: durante a suavização da adaptação de fluxo, é permitido ao TFMCC violar temporariamente sua característica *TCP-Frindly*. Um algoritmo de compensação é adotado durante esse intervalo. Os resultados, simulados através do aplicativo “NS-2 simulator”,

mostram que a solução oferece PSNR média de 1 dB acima quando comparada com TFMCC normal, 29 dB. A PSNR máxima do vídeo testado é 40 dB. É tratado um fluxo *multicast* concorrente com 15 fluxos TCP. Esse trabalho é focado em codificação MDC, não sendo diretamente adequado ao SVC.

4.2 - O TFMCC

O protocolo TFMCC [48], proposto sob a RFC experimental de número 4654, de 2006, é uma adaptação do TFRC para o cenário multidestinatário. O TFRC atende satisfatoriamente o critério de TCP-Equal share [45].

O TFMCC é baseado em taxa (RB ou *Rate-based*), o que lhe confere maior suavidade na política de ajuste da taxa de transferência quando comparado com aqueles baseados em janela de contenção (WB ou *Window-based*).

O TFMCC é desenhado para controle de congestionamento multidestinatário em ambos os modelos ASM (*Any-Source Multicast*), em que o *feedback* dos receptores é enviado para o transmissor e também para todos os demais receptores do grupo, e SSM (*Source-Specific Multicast*), em que o *feedback* dos receptores é enviado de maneira *unicast* para o transmissor, apenas. O protocolo será estudado no modo SSM.

Para prover escalabilidade, o TFMCC dedica a maior carga de processamento para os nodos receptores.

Da responsividade

Em geral o TFMCC, segundo [48], é bastante cauteloso na variação da taxa de transmissão, o que torna sua aplicação desejável em sistemas que requerem taxa de envio relativamente suave, tal como *media streaming.* O TFMCC é classificado como pouco responsivo a mudanças na largura de banda disponível. Esse comportamento é adequado para o contexto tratado neste trabalho, pois a responsividade é traduzida, na prática, pela redução no fator de ½ da taxa de transmissão depois de percebido um evento de perda de pacote. Cabe observar que essa penalidade é evitada pelo TFMCC.

Da justiça

O TFMCC é projeto para ser *razoavelmente justo* quando coexistindo concorrentemente com fluxos de TCP. Um tráfego multidestinatário é dito *razoavelmente justo* se sua taxa de envio vale no máximo duas vezes a taxa do fluxo TCP entre o transmissor e o receptor mais lento do grupo, sob as mesmas condições de perda de pacote e RTT [48].

Da restrição de largura de banda mínima

Segundo os próprios autores, o TFMCC não deve usado junto com tráfego TCP se a capacidade do enlace é menor que 30 Kbit/s. Por sorte, o projeto aqui tratado nem convém ser utilizado em redes de tão baixa capacidade, vez que o objeto de estudo é transmissão vídeo.

Da restrição de tamanho fixo de pacotes

O TFMCC é proposto para aplicações que utilizam pacotes de tamanho fixo e que se valem do ajuste da taxa de envio de pacotes em resposta a um congestionamento ou aumento da banda de rede.

Este trabalho baseia-se no envio de blocos de vídeo em uma taxa constante. Alterar essa taxa em tempo de execução significa alterar a duração (com respeito à reprodução) dos blocos de vídeo. Esse procedimento, todavia, requer o processo de transcodificação de vídeo, que: (a) contraria a noção de vídeo escalável, no qual o ajuste de qualidade deve ocorrer sem necessidade de transcodificação; (b) sugeriria uma capacidade computacional extremamente alta em todos os nós da rede.

O requisito do TFMCC pode, contudo, ser atendido considerando que os blocos de vídeo serão fragmentados em unidades pequenas, de tamanho fixo, e que a transmissão dos fragmentos será distribuída ao longo do período do bloco. O fragmento pode ser dimensionado para o tamanho do menor bloco de vídeo – aquele contendo apenas a camada base SVC.

5 - PROPOSTA DE ARQUITETURA ADAPTATIVA PARA TRANSMISSÃO E EXECUÇÃO DE VÍDEO SVC

5.1 - INTRODUÇÃO

Os capítulos anteriores apresentaram as três principais áreas temáticas deste trabalho: Vídeo Escalável, Redes P2P e Controle de Congestionamento. Esses conceitos são agora casados para compor a proposta de arquitetura de sistema.

A arquitetura inicia-se por considerar a utilização do esquema de distribuição de dados conforme o modelo em árvore. A Figura 21 mostra um grupo de comunicação em árvore sobre uma malha P2P. O nó raiz está identificado por 1 (um). No segundo nível hierárquico da árvore estão os nós 2, 3 e 4. Por fim, no último nível hierárquico estão 5 e 6. A fonte de vídeo é um nó externo à árvore e abstrai todo o processo de captura e codificação de vídeo.

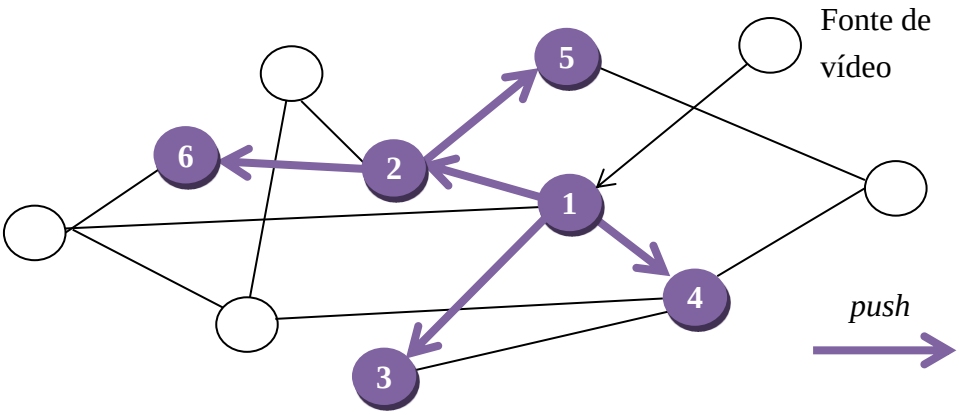


Figura 21 – Árvore de distribuição construída sobre uma P2P

As prerrogativas de baixa latência e de pouca sobrecarga de comunicação de controle são importantes para o cenário de distribuição ao vivo de vídeo.

A baixa latência permite que um usuário não receba, por exemplo, o evento “gol !” de uma partida de futebol com grande atraso com relação a outro usuário que esteja, possivelmente, recebendo a transmissão por um enlace de baixíssimo atraso. A boa experiência do usuário é chave crucial para o sucesso da proposta.

A pouca sobrecarga de controle possibilita o uso mais eficiente da banda de rede disponível. Dado que o sistema trata de fluxo contínuo, o usuário não tem muito tempo para esperar as informações de melhoramento de qualidade do vídeo (camadas SVC).

Assim, quanto maior a banda disponível para envio do vídeo propriamente dito (*payload* dos pacotes de rede), melhor será a qualidade vivenciada pelo usuário.

Para substrato de infraestrutura P2P em malha é selecionado o Pastry. Sobre este, a rede lógica em árvore é construída pelo Scribe.

Uma instância do controle de congestionamento TFMCC dar-se-á para cada conjunto das arestas que saem de um nó, conforme mostra a Figura 22:

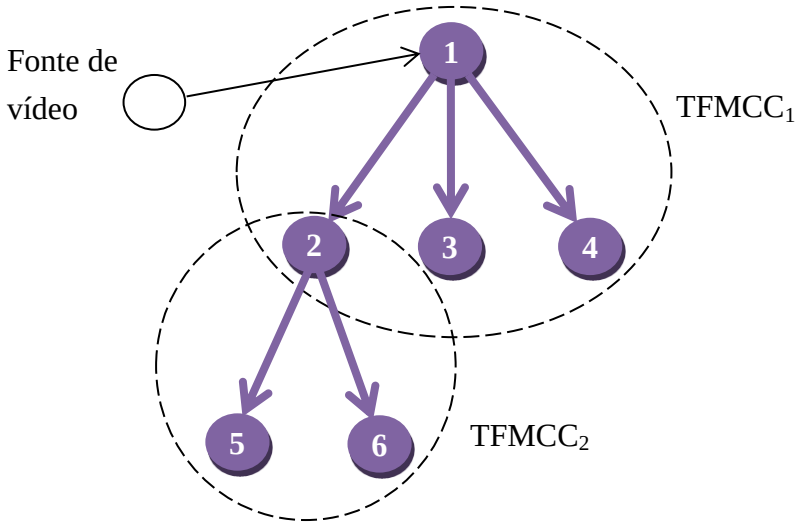


Figura 22 – Instâncias do controle de congestionamento na árvore P2P

De forma a manter o sistema escalável, cada instância do controle de congestionamento tem a visão limitada ao nível hierárquico da árvore na qual ela atua, considerando apenas os enlaces (arestas) que originam no mesmo nó. Com isso o protocolo TFMCC pode ser utilizado na sua forma original.

Como efeito da estrutura apresentada, uma redução da qualidade do vídeo em determinado ponto da árvore afeta transitivamente todos os nós filhos vinculados. Por exemplo: má condição no enlace 1-3 causaria redução da qualidade do vídeo oferecido pelo nó 1, o que afetaria os nós 2, 3 e 4 e, por conseguinte, os nós 5 e 6. É necessário, portanto, atenção especial nos primeiros níveis hierárquicos da árvore, assunto esse discutido em capítulo posterior.

O vídeo gerado pela fonte é comprimido no padrão H.264 SVC segundo o *paradigma de compressão em blocos*, explicado em seção seguinte. Esse paradigma permite a simplificação do transporte, tornando a solução interessante no sentido da implementação e do custo computacional graças à abstração da complexidade de análise do *stream* SVC nos

processos de adaptação e de reprodução (*playback*) do vídeo. O paradigma não traz prejuízo às possibilidades de escalabilidade oferecidas pelo referido padrão.

O resultado da compressão é uma sequência de blocos em que cada bloco contém um intervalo de tempo de vídeo. Esses intervalos possuem comprimento constante durante toda a transmissão. Os blocos de vídeo SVC são então fragmentados em unidades de tamanho fixo e cada unidade é encapsulada em um pacote RTP (*Real-Time Protocol*) e transmitida para o nó raiz da árvore (*rendezvous point*).

O nó raiz, bem como todos os demais nós da árvore, executam os seguintes passos:

- Recebe os fragmentos encapsulados em pacotes RTP;
- Remonta o bloco de vídeo SVC;
- Envia uma cópia do bloco de vídeo para a fila de reprodução em tela;
- Procede com a adaptação do bloco realizada com auxílio de informações oferecidas pelo controle de congestionamento TFMCC que é estabelecido entre o nó corrente e os seus nós imediatamente filhos. O resultado da adaptação é um bloco de menor tamanho devido à remoção de camadas SVC. Essa etapa não se aplica caso o nó corrente não possua filhos ou caso o TFMCC não esteja sugerindo redução da taxa de transmissão;
- Fragmenta o bloco adaptado em unidades de tamanho fixo;
- Encapsula os fragmentos em pacotes RTP fazendo também *PiggyBack* do TFMCC através do cabeçalho de extensão do RTP;
- Transmite os pacotes RTP para os nós filhos segundo a política *push* da árvore.

Cada nó da árvore pode ser representado pelo diagrama de componentes da Figura 23.

Em concordância com as premissas de distribuição e de descentralização, todos os nós da rede são iguais, ou seja, contêm os mesmos componentes de arquitetura e executam os mesmos algoritmos. Os nós são ditos MANEs (*Media Aware Network Elements*) pois eles realizam a adaptação de qualidade com ciência do conteúdo (que são os blocos SVC).

A configuração do protocolo de transporte em tempo real RTP é apresentada na seção 5.3. Na seção 5.4 é apresentado o esquema para implantação do controle de congestionamento, de duas partes: (a) Especificação da sintaxe e semântica da extensão de cabeçalho RTP; (b) Especificação da sintaxe e semântica dos pacotes de *feedback* - sinalização de retorno que é enviada dos filhos para o pai em cada instância do TFMCC.

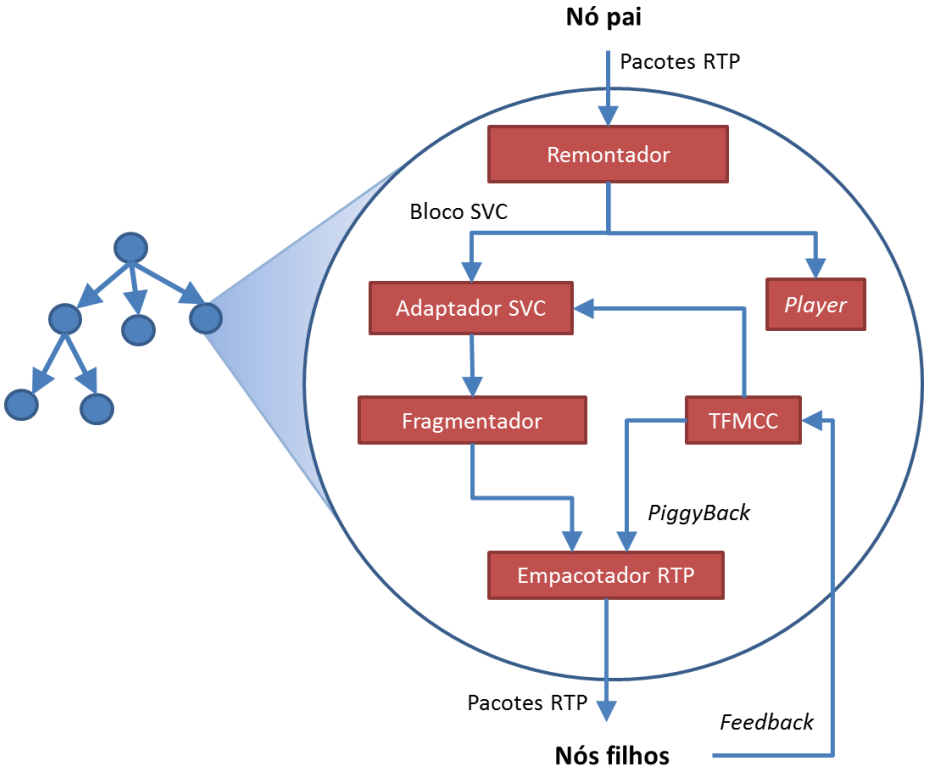


Figura 23 – Componentes de um nó da árvore de distribuição

O esquema de fragmentação é apresentado na seção 5.5. Definem-se a sintaxe e semântica do *payload* RTP bem como o algoritmo para remontagem espelhado no que faz o IP. É acrescentado o suporte a remontagem de múltiplos pacotes em paralelo. As definições referentes à fragmentação são estabelecidas para atendimento ao protocolo TFMCC.

No diagrama da Figura 23 temos os seguintes módulos:

Remontador de Blocos (RB)

Implementa o algoritmo de remontagem de fragmentos conforme explanado na seção 5.6. As premissas do remontador são:

- Atraso mínimo: O pacote montado é entregue à aplicação tão logo sua remontagem é concluída com sucesso;
- Tolerância a falhas de ordenação de fragmentos provida pelos múltiplos compartimentos (*slots*) de remontagem de pacote. Observa-se que o algoritmo proposto no protocolo IPv4, por exemplo, considera apenas 1 compartimento de remontagem de pacote, de forma que a chegada antecipada de um fragmento do próximo pacote resulta no descarte dos fragmentos atualmente em memória no

remontador, causando a perda do pacote corrente. Neste caso, o remontador com multiplos compartimentos evita o descarte mencionado.

Deve-se configurar o número compartimentos de remontagem conforme a capacidade de memória do nó. Sugere-se uma configuração inicial de dois compartimentos.

***Player* ou Reprodutor de Vídeo (RV)**

Responsável pela reprodução do vídeo para o usuário final. Esse módulo interpretará os blocos de vídeo SVC restaurando a sequência de quadros e temporizando-os corretamente para exibição em tela.

O Reprodutor de Vídeo, simplesmente chamado de RV, possui cinco componentes: Fila de blocos, decodificador, exibidor e dois temporizadores de sincronização.

A Figura 24 mostra o diagrama interno deste módulo.

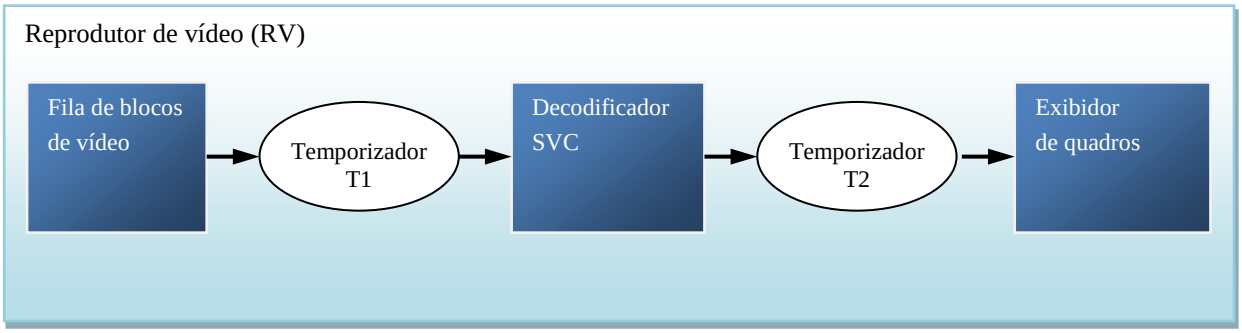


Figura 24 – Unidade de reprodução de vídeo

O componente Fila possui as seguintes funções:

- Ordenação dos blocos: A Fila, de posse do número de sequência do bloco, tem a responsabilidade de entregar os blocos para o primeiro temporizador na ordem correta de exibição;
- Sincronização de partida: A Fila determina o momento de início da reprodução do vídeo através do congelamento temporário de sua saída até que exista uma quantidade suficiente blocos em memória. A Fila provoca esse atraso na reprodução do vídeo com o intuito de possibilitar o cancelamento de *jitter* (variação do atraso na chegada dos blocos) e permitir, também, tolerância com respeito a intervalos de tempo (não longos) de indisponibilidade de serviço (não chegada de pacotes).

A Fila inicia no estado “parado”, na qual ela descarta qualquer bloco recebido e mantém sua memória interna vazia. Quando o usuário solicita o início da produção (*play*), a Fila transita para o estado “realizando carga” (*buffering*), no qual ela recebe e armazena os blocos entregues pelo RB. Após alcançar a carga configurada como nível de partida, a Fila emite sinal (evento) para que o T1 transite do estado “parado” para o estado “executando” e comece, então, a consumir os blocos da Fila. Esse primeiro temporizador é um relógio de baixa escala (“lento”), cuja velocidade é de acordo com a frequência de decodificação de bloco (F1). F1 é uma constante do sistema. O temporizador T1 tem, portanto, a função de interagir com o decodificador SVC em intervalos síncronos e igualmente espaçados, conforme a frequência F1 do sistema.

A Figura 25 e a Figura 26 mostram, respectivamente, o diagrama estendido de estado finito (FSM, *Finite-State Machine*) [44] da Fila e do temporizador T1.

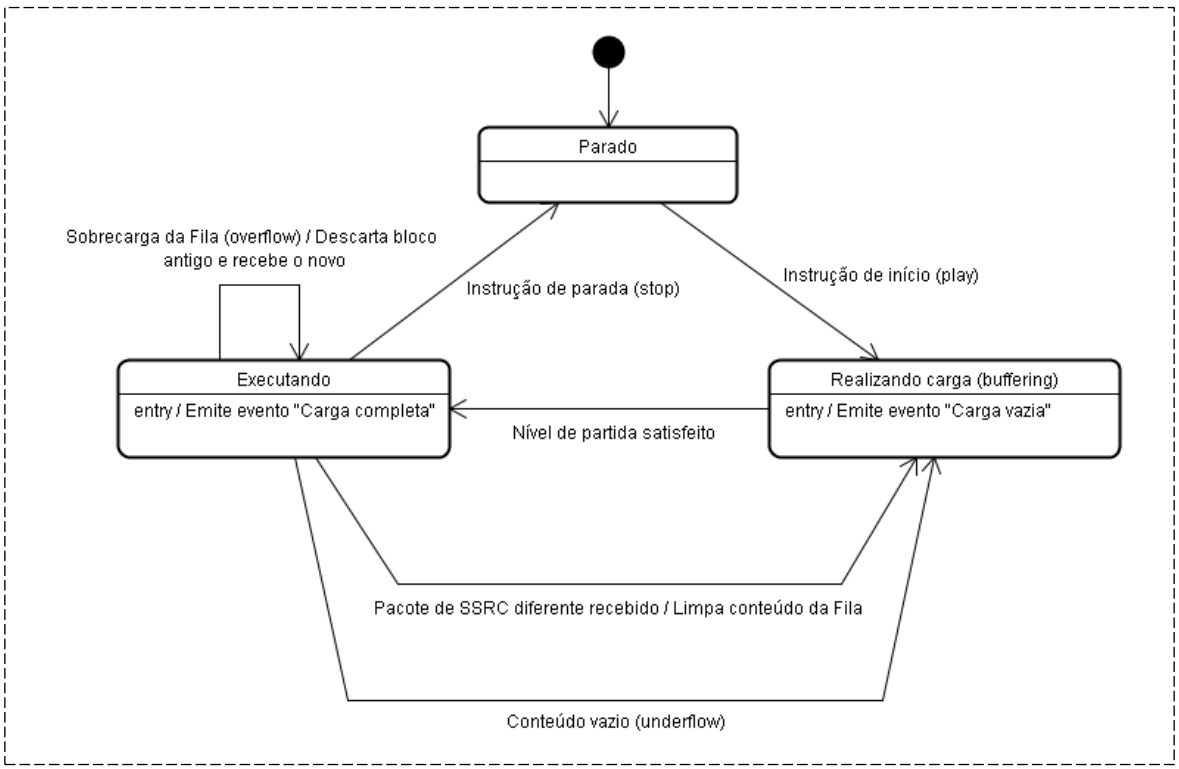


Figura 25 – Diagrama de estados da Fila de blocos SVC

O decodificador SVC, após realizar seu trabalho, o qual ocorre durante o estado “ocupado”, retorna para o estado “disponível” e mantém, em sua interface de saída, os quadros de vídeo disponíveis para leitura pelo temporizador T2.

O temporizador T2 captura os quadros (cada bloco de vídeo contém 1 ou vários quadros) e transfere-os, um a um, conforme a frequência de exibição de quadro (F2). Essa frequência

é comumente medida em FPS (*Frames Per Second*). A frequência F2, na arquitetura em questão, varia em função da escalabilidade temporal de cada bloco SVC.

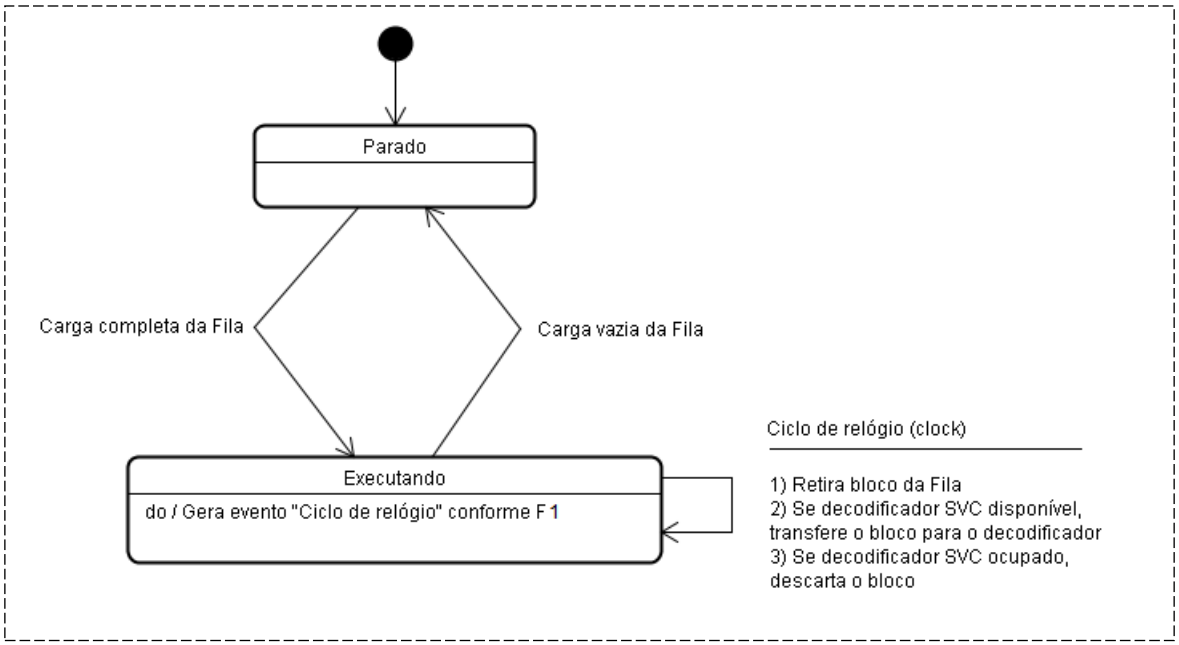


Figura 26 – Diagrama de estado do temporizador de decodificação de bloco (T1)

Os diagramas de estado do decodificador SVC e do T2 são mais simples e estão abaixo apresentados na Figura 27 e Figura 28, respectivamente:

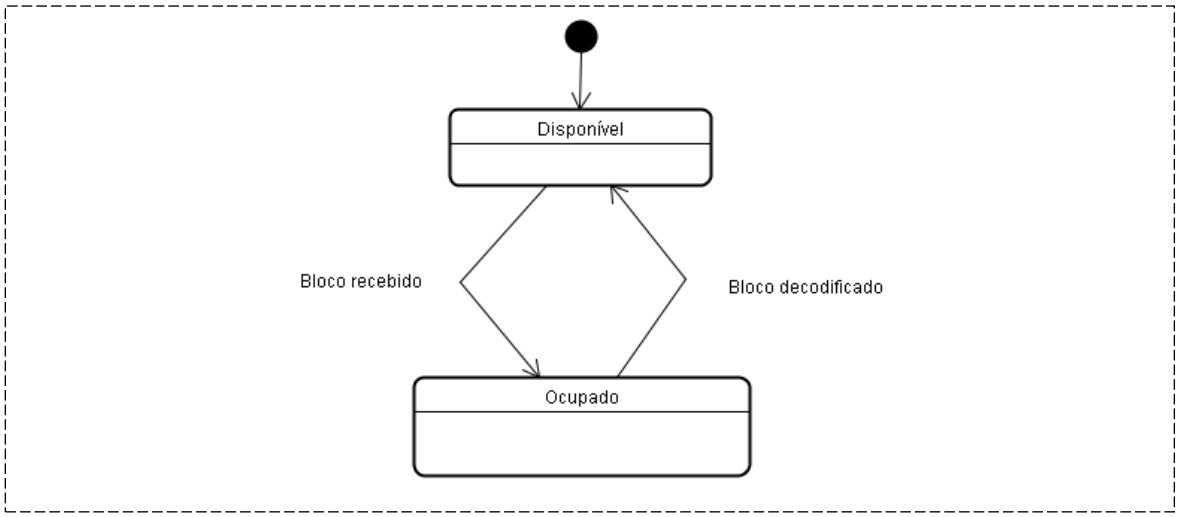


Figura 27 – Diagrama de estados do decodificador SVC

O evento “fim do bloco” ocorre quando todos os quadros do bloco já foram exibidos.

Os diagramas de T1 e T2 abstraem detalhes que convêm serem evidenciados no capítulo sobre a implementação. Um exemplo é o descarte por atraso: se o exibidor está livre (“disponível”) e o quadro está muito atrasado para o tempo correto de sua exibição, então ocorrerá o descarte. O descarte ocorre para que o T2 avance imediatamente para o próximo ciclo de relógio. Esse cenário exemplificado é comum quando o nó não dispõe de recursos computacionais suficientes para manter a sincronia correta dos relógios T1 e T2 nas frequências F1 e F2, respectivamente. De fato, existe então um *jitter* gerado em nível de aplicação dentro dos próprios temporizadores T1 e T2. Se o erro no tratamento do ciclo de relógio é maior que um período completo, então necessariamente convém descartar a informação e avançar para o próximo ciclo.

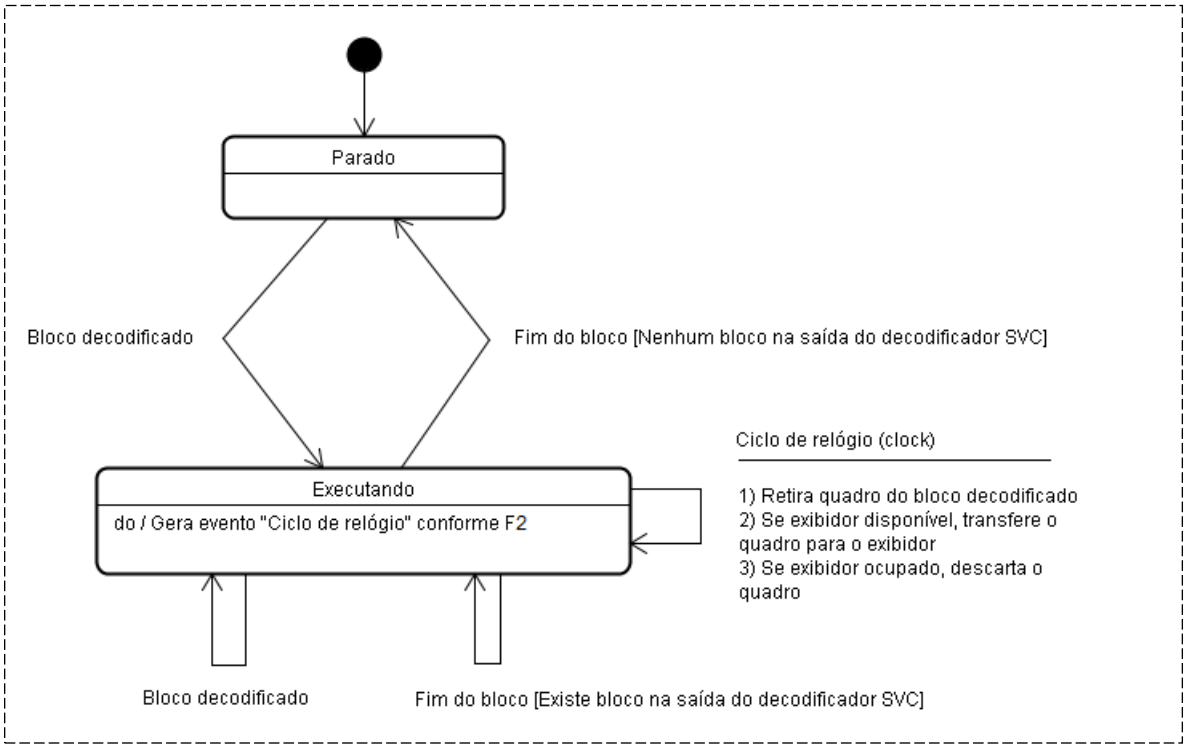


Figura 28 – Diagrama de estado do temporizador de exibição de quadro (T2)

5.2 - PARADIGMA DE COMPRESSÃO EM BLOCOS

A compressão e consequente transmissão de vídeo podem ocorrer em dois paradigmas distintos conforme mostra a Figura 29:

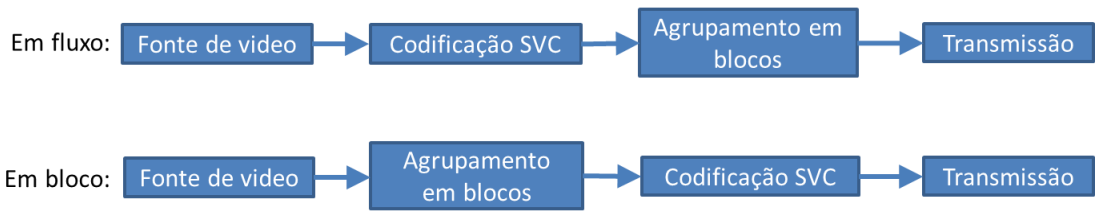


Figura 29 – Paradigmas de compressão em fluxo e em blocos

Compressão em Fluxo

No paradigma convencional, comprime-se um vídeo longo e transmitem-se pedaços, também chamados de *chunks*, desse vídeo. Esse modelo é classificado, neste trabalho, como compressão em fluxo.

Nesse esquema, conjuntos de unidades NAL do formato SVC seriam transmitidos na rede dentro de pacotes. Esses pacotes seriam recebidos e processados para exibição e para adaptação.

Algumas dificuldades práticas tornam esse esquema pouco interessante:

Do Transporte

O fragmentador de pacotes precisaria interpretar o *bytestream* SVC para realizar a fragmentação sem corromper unidades NAL, vez que os decodificadores existentes não são capazes de tratar esse cenário – *bytestream* corrompido. Esse procedimento requer um processamento de camada alta, o que exige grande poder de processamento. Esse problema, se não tratado no fragmentador, precisaria necessariamente ser resolvido em camada mais alta do sistema desde que antes da interação com o decodificador SVC.

Da Reprodução

As unidades NAL possuem estrutura sintática complexa, conforme mostrado no capítulo 2. Vários cabeçalhos podem não estar presentes conforme valores de outros cabeçalhos da mesma unidade NAL ou até mesmo de unidades NAL anteriores.

O componente Reprodutor de Vídeo receberia pacotes contendo uma ou mais unidades NAL. Os pacotes seriam concatenados até ocorrer a detecção e separação em lotes que possam ser interagidos com o aplicativo decodificador de SVC (*decoder* ou *codec*).

Numa primeira abordagem o lote de interação com o decodificador seria uma unidade AU (*Access Unit*). Separar unidades AU exige a realização dos algoritmos de detecção de início e de fim de AU. Tais algoritmos requerem análise em nível de cabeçalho de unidades VCL, sendo, portanto, um processo custoso.

Os cabeçalhos de unidades VCL dependem de valores de cabeçalhos anteriores e, por vezes, de outros pacotes em memória como SPS e PPS. Manter esses pacotes em memória

requer entendimento do modelo HRD (*Hypothetical Reference Decoder*), que trata do armazenamento e descarte de quadros na memória DPB (*Decoded Picture Buffer*).

A implementação de HRD é necessária para o processo de codificação, porém é redundante para ser tratada também no módulo RV do sistema, pois envolve entender grande parte do processo de interpretação do *bytestream* SVC.

Existe dependência entre as unidades AU devido à estrutura de predição inter-quadro. Essa dependência é sinalizada no SVC. A ordem de posicionamento das AUs no SVC está expressa na sequência POC (*Picture Order Code*). A extração do número POC de cada AU depende da análise de cabeçalhos dos pacotes VCL presentes na AU e dos pacotes PPS e SPS relacionados. O Apêndice A exemplifica esse cálculo.

A necessidade de interpretação, com memória, dos cabeçalhos de unidades NAL, processamento este que é uma redundância, vez que já é realizado dentro do decodificador SVC, remete a um custo computacional significativo em cada salto na rede P2P.

As dificuldades acima explanadas são mantidas considerando a abordagem em que cada lote contém um GOP inteiro, pois existe também dependência entre unidades NAL de GOPs consecutivos.

O cenário de canal não confiável, onde há perda de pacotes, que é o caso abordado neste trabalho, o cálculo do ordenamento POC fica comprometido e inviabiliza a correta marcação de tempo dos quadros emitidos na saída do decodificador SVC. A saída do deste não obedecerá ao previsto na Tabela 3, podendo variar conforme implementação.

Nas implementações de decodificador SVC citadas no capítulo 2 não é possível embarcar metadados nas unidades AU de forma que a informação seja repetida na saída (quadro não comprimido). Se possível fosse, a carimbação prévia de tempo, a ser feita nas unidades AU, resolveria o problema.

Se o FPS não fosse dinâmico no SVC, a marcação de tempo na saída do *decoder* seria trivial e o calculo de POC poderia ser desprezado. FPS dinâmico não é um problema simples. As arquiteturas padrões de mercado não estão preparadas para tratar essa questão e nem mesmo a ordem POC. Testes feitos no Microsoft Media Player 10, VLC 2 e Apple QuickTime 7.7, ao executarem vídeo SVC de única camada, com GOP hierárquico, codificado em modo de compatibilidade AVC, resultou na reprodução dos quadros em ordem errada.

Compressão em Blocos

Nessa abordagem, conjuntos formados por 1 ou mais GOPs completos seriam codificados na fonte de maneira independente. Ou seja, ao invés de termos um vídeo longo transmitido em pedaços, teríamos vários vídeos curtos transmitidos atomicamente. Assim, os módulos de Reprodução e de Adaptação não precisam realizar qualquer processamento de camada alta que tenha relação com a análise de cabeçalhos de unidades NAL, identificação de início e fim de AU ou mesmo de correção de ordem POC.

Vantagens da proposta:

- Independência da técnica de compressão. Não é necessário tratar da interpretação de características como POC, HRD, etc;
- Simplificação da interação com o decodificador SVC, pois cada bloco é um *bytestream* completo e válido;
- Decodificação passível de paralelização, pois os blocos não possuem dependência entre si;
- Não há restrições ao uso das características do SVC;
- Sincronização de tempo dos quadros decodificados torna-se trivial, vez que a taxa de blocos é constante.

Desvantagens da proposta:

- O atraso em cada nó da árvore é de pelo menos a duração de um bloco inteiro;
- Os blocos precisam ser fragmentados para atender à restrição de tamanho fixo de pacote do TFMCC.

5.3 - TRANSPORTE EM TEMPO REAL

A camada de transporte é baseada no protocolo RTP conforme a seguinte configuração de cabeçalhos:

Tabela 9 – Sintaxe e semântica do pacote RTP

Bit offset	0-1	2	3	4-7	8	9-15	16-31
0	Ver.	P	X	CC	M	PT	SeqNum
32	TimestampInMillis						
64	SSRC						
96	Payload…						

- Ver.: versão do protocolo. Valor fixado no valor “10”;
- P (*Padding*): indica preenchimento adicional do pacote. Valor fixado em “0”;
- X (*Extension header*): indica se o cabeçalho estendido está presente. É específico de cada aplicação. Na seção 5.4 é apresentado o cabeçalho de extensão proposto para a presente arquitetura. Valor fixado no valor “1”;
- CC (*CSRC count*): contador de identificadores CSRC para fluxos de múltiplas fontes. Valor fixado em “0”;
- M (*Marker*): indica que o *payload* tem relevância especial para a aplicação. Valor fixado em “0”;
- PT (*Payload type*): tipo de conteúdo encapsulado no RTP, também chamado de *profile RTP*. Devido ao uso do cabeçalho de extensão (*Extension header*) e da proposta de *payload* apresentada na seção 5.5, é necessário utilizar um novo valor PT. No presente trabalho esse valor é arbitrário, porém, para fins de uso real na internet, será necessário aprovação do novo valor de PT pela IANA (*Internet Assigned Numbers Authority*);
- SeqNum (*Sequence number*): número de sequência do pacote;
- TimestampInMillis: momento em que o pacote foi enviado;
- SSRC: identificação do canal de mídia. Único em toda a sessão.

5.4 - CONTROLE DE CONGESTIONAMENTO

O cabeçalho de extensão RTP é utilizado para *PiggyBack* do TFMCC, conforme sintaxe e semântica abaixo. A utilização do cabeçalho *Extension header* para este fim está em concordância com a RFC 3550 [50], que diz: “the header extension may be ignored by other interoperating implementations that have not been extended.”. De fato o controle de congestionamento é uma ferramenta acessória, não necessária para entendimento do conteúdo do pacote RTP.

Tabela 10 - Sintaxe e semântica do *PiggyBack* TFMCC (cabeçalho de extensão RTP)

Bit offset	0-7	8	9-31
0	ts_i		
32	RTT_max		
64	fb_nr	is_CLR	reserved
96	r		
128	tr_r_line		
160	X_supp		
192	Payload...		

- ts_i: momento de envio em milissegundos;
- RTT_max: RTT máximo dentre todos os receptores TFMCC;
- fb_nr (*feedback round*): contador de iteração do TFMCC, incrementado pelo transmissor;
- is_CLR (*Current Limiting Receiver*): indica se o receptor identificado por *r* é o CLR;
- reserved: reservado para uso futuro;
- r: identificação do receptor referente aos atributos *tr_r_line* e *is_CLR*;
- tr_r_line: retorno (*echo*) do momento em que o transmissor recebeu o último *feedback* do receptor identificado por *r* (*tr_r*) somado ao atraso decorrido até o envio do pacote (*ts_d*), i.e., tr_r_line = tr_r + ts_d;
- X_supp: taxa de supressão do TFMCC em bits por segundo.

5.5 - ENCAPSULAMENTO E FRAGMENTAÇÃO DOS BLOCOS

Os blocos SVC são encapsulados em fragmentos conforme a estrutura a seguir.

Tabela 11 – Sintaxe e semântica do encapsulamento de bloco

Bit offset	0-1	2-4	5-8	9	10-11	12-14	15	16-19	20-23	24-31
0	Ver.	Reserved		EH	NumAH		LF	FragOffset		
32	BlockNum									
64	MaxWidth							MaxHeight		
96	AspectN			AspectD				DurationInMillis		
128	Layer		SizeFactor			D		T	Q	NumFrames
160	Layer		SizeFactor			D		T	Q	NumFrames...
	Payload...									

- Ver.: versão do protocolo. Valor fixado em “01”;
- *Reserved*: reservado para uso futuro;
- EH (*Extra header*): indica se existe cabeçalho extra. Os cabeçalhos extras são: *MaxWidth, MaxHeight, AspectN, AspectD, DurationInMillis.* Pode valer zero a partir do segundo fragmento*;*
- NumAH (*Number of adaptation headers*): número de cabeçalhos de adaptação. Cada cabeçalho detalha uma camada SVC presente no bloco para a qual este pode ser escalonado. Por restrições diversas, algumas camadas podem não estar declaradas. O conjunto formado pelos cabeçalhos *NumAH*: *Layer*, *D, T, Q, NumFrames* e *SizeFactor* se repete conforme a quantidade *NumAH*, que pode valer zero a partir do segundo fragmento. A lista é ordenada da camada mais alta para a

mais baixa de forma que o primeiro conjunto da lista é a qualidade corrente do bloco;

- LF (*Last fragment*): indica se é o último fragmento do bloco;
- FragOffset (*Fragment offset*): deslocamento do fragmento em múltiplo de 256 Bytes;
- BlockNum (*Block number*): número sequencial do bloco de vídeo;
- MaxWidth (*Maximum width*): largura máxima, em pixels, do quadro que o fluxo pode atingir na melhor qualidade espacial da fonte de vídeo. Importante para que o receptor possa pré-alocar recursos;
- MaxHeight (*Maximum height*): altura máxima, em pixels, do quadro que o fluxo pode atingir na melhor qualidade espacial da fonte de vídeo. Importante para que o receptor possa pré-alocar recursos;
- AspectN: numerador da fração de proporção de quadro;
- AspectD: denominador da proporção de quadro;
- DurationInMillis: duração do bloco. Constante durante toda a sessão RTP;
- Layer: número da camada SVC. A camada base tem valor zero;
- SizeFactor: fator de redução da resolução com respeito a *MaxWidth* e *MaxHeight*. Valor 2, por exemplo, significa metade da largura e altura do quadro;
- D: parâmetro *Dependency Layer* da camada SVC;
- T: parâmetro *Temporal Layer* da camada SVC;
- Q: parâmetro *Quality Layer* da camada SVC;
- NumFrames (*Number of frames*): número de quadros da camada SVC.

5.6 - REMONTAGEM DOS BLOCOS

A remontagem dos blocos SVC a partir dos fragmentos recebidos via RTP é espelhada no procedimento adotado pelo IPv4. É feita uma extensão para que múltiplos blocos sejam remontados em paralelo.

Os fragmentos recebidos são colocados em uma fila ordenada pelo número de sequência RTP. Cada fragmento refere-se a um bloco de número B_i. A quantidade máxima permitida de números distintos de bloco determina a capacidade C de remontagem paralela. Tão logo se recebe um próximo número de bloco B_j, os fragmentos pertencentes a blocos de número $i <= N - j$ são expirados (removidos da fila).

O trecho abaixo propõe um algoritmo de remontagem que implementa essa lógica:

Tabela 12 – Algoritmo de remontagem de bloco

```
Processa (fragmento)
{
```

```
Bloco de seção crítica (monitor com trava de exclusão mútua na fila ordenada)
{
   Se o SSRC mudou
   {
      Limpa a fila ordenada
   }

   Se o fragmento é o único do bloco
   {
      Entrega fragmento à aplicação
      Fim
   }

   Se o número do bloco associado ao fragmento é menor que o menor número de bloco
   presente na fila a menos de uma constante C
   {
      Descarta o fragmento
      Fim
   }

   Adiciona fragmento na fila ordenada usando número de sequência RTP como chave

   Faz busca por remontagem concluída
   {
      Cria lista FIFO temporária

      Inicia expectedFragOffset em 0
      Inicia currentBlockNum como sendo o primeiro número de bloco da fila ordenada

      // currentBlockNum  = número do bloco que espera-se remontar
      // expectedFragOffset = valor esperado para o do próximo fragmento da iteração

      // Se as três condições abaixo forem verdadeiras, então se tem um bloco
      // inteiro na lista FIFO
      // 1- currentBlockNum igual ao blockNum da iteração
      // 2- expectedFragOffset igual ao fragOffset da iteração
      // 3- lastFrag da iteração é verdadeiro

      Percorre os fragmentos da fila ordenada
      {
         // Se as condições (1) OU (2) acima forem violadas então se tem uma
         // tentativa de reagrupamento mal sucedida. Assim limpa-se o
         // conteúdo da lista FIFO. Volta-se expectedFragOffset para 0 pois o primeiro
         // fragmento de todo bloco tem fragOffset = 0

         Se não condicao1 OU não condicao2
         {
            Limpa lista FIFO temporária
            currentBlockNum = número do bloco do fragmento da iteração
            expectedFragOffset = 0
         }

         // Se as condições (1) e (2) acima forem verdadeiras então o fragmento é o
         // próximo fragmento esperado.

         Se condicao1 E condicao2
         {
            Adiciona fragmento na lista FIFO temporária

            // Se a condição (3) for verdadeira, tem-se um bloco completo.

            Se condicao3
            {
               blocoCompleto = concatenação sequencial do payload dos
                               fragmentos da lista FIFO temporária

               pacoteRemontado = novo pacote RTP onde o payload é o
                                 blocoCompleto, o cabeçalho LF vale verdadeiro,
                                 o cabeçalho fragOffset vale zero e demais
                                 cabeçalhos é a união dos cabeçalhos dos
                                 fragmentos da lista FIFO temporária
```

```
                    Subtrai da fila ordenada todos os fragmentos da lista FIFO

                    Guarda referência de pacoteRemontado
                }

                // Se a condição (3) for falsa, ajusta-se o valor de expectedFragOffset

                Se não condicao3
                {
                    expectedFragOffset = expectedFragOffset +
                                ( tamanho em bytes do payload do fragmento da iteração / 256 )
                }
            }
        }
    }

    Expira fragmentos antigos
    {
        Percorre os fragmentos da fila ordenada (objeto de iteração: old)
        {
            Se old.blockNum < fragmento.blockNum – C
            {
                Remove old da fila ordenada
            }
        }
    }

    Se existe referência de pacoteRemontado, entrega-no à aplicação
  }
}
```

5.7 - ADAPTAÇÃO DA QUALIDADE DE VÍDEO

O módulo Adaptador SVC é responsável pelo escalonamento dos blocos para que a taxa de transmissão resultante esteja o mais próximo possível da taxa sugerida pelo módulo TFMCC.

O algoritmo proposto é simples: escolhe-se os parâmetros D, T e Q (*Dependency Layer*, *Temporal Layer* e *Quality Layer*) tal que o *bitrate* associado seja o maior possível e menor que taxa sugerida pelo TFMCC.

O conjunto das tuplas (D, T, Q, *bitrate*) está presente no pacote de vídeo para auxiliar o algoritmo acima.

A convergência do algoritmo está condicionada ao que se segue: sejam L_i as tuplas (D, T, Q, *bitrate*) ordenadas crescentemente pelo *bitrate*, onde $0 <= i <= k$. É necessário que, para toda tupla L_j, onde $0 <= j < k$, a relação $bitrate(L_{j+1}) - bitrate(L_j) < 2 \cdot bitrate(L_{j+1}) - M$ seja verdadeira.

A condição se deve ao fato de que o TFMCC aumenta a taxa conforme aquela recomendada pelo nó receptor CLR (*Current Limiting Receiver*) e este, por sua vez, sugere uma taxa que vale o dobro da taxa recebida medida. Inerente a qualquer processo de medição, existe um erro de precisão. Nesse sentido, o parâmetro M, positivo, é um fator de

tolerância adicionado à equação de forma a resolver o erro de precisão associado à medição de taxa realizada pelos receptores TFMCC.
Consequente à condição acima, tem-se que os saltos em *bitrate* podem seguir, no máximo, progressão geométrica de razão 2.

5.8 - CONFIGURAÇÃO DA CODIFICAÇÃO SVC

Recomenda-se a codificação de vídeo SVC no modo MGS (*Medium-Grain Scalable coding)* devido às vantagens deste esquema com respeito ao modo CGS (*Coarse-Grain Scalable coding*). Essas vantagens foram explanadas no capítulo 2.

No modo MGS a ferramenta de extração de *bytestream* da suíte JSVM possui comportamento mais interessante para uso em rede P2P: é possível escalonar o bloco SVC para uma camada L_i mantendo-se todas as camadas inferiores L_j, $j <= i$. No modo CGS não é garantido que todas as camadas L_j possam ser utilizadas para decodificação da camada L_i. Com isso, é possível que o *bytestream* resultante possua *gaps*, ou seja, ausência de camadas intermediárias entre a base e a camada mais alta. Por consequência o próximo nó da rede terá menos possibilidades para escalonamento do vídeo, culminando, portanto, numa rápida degradação da qualidade do vídeo nos saltos ao longo da árvore de distribuição. Esse efeito é mostrado na
Figura 30.

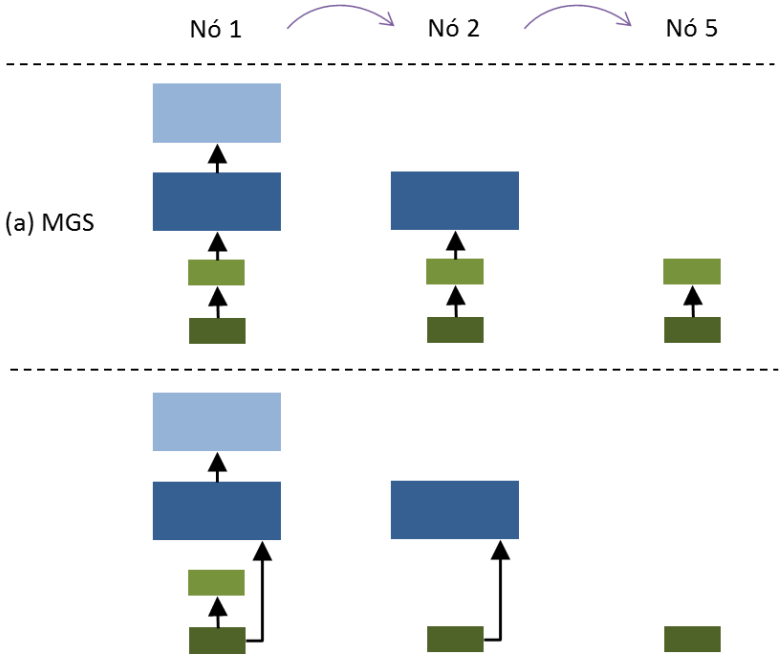


Figura 30 – Degradação do CGS e do MGS durante a comutação da árvore P2P

Investigações sobre o comportamento do JSVM Extractor estão no Apêndice B.

Propõe-se que as camadas de melhoramento L_i resultem em saltos de taxa (*bitrate*) em progressão geométrica de razão R, $1 < R < 2$. O limite 2 fora anteriormente justificado.

Sabe-se que a percepção de qualidade com respeito ao ruído de compressão de um vídeo SVC está intimamente relacionada com o parâmetro de codificação QP (*Quantization Parameter*). O manual do JSVM recomenda que cada camada de melhoramento de qualidade Q seja dada por um decremento linear de pelo menos duas unidades no QP. Por exemplo, se camada Q = 0 tem QP = 30, então Q = 1 poderia ser gerada com QP = 28 e Q = 2 com QP = 26. Vale lembrar que QP representa o passo de quantização (distância entre os valores quantizados), de forma que menor QP resulta em melhor imagem. Da técnica de compressão SVC tem-se que QP possui uma relação aproximadamente exponencial com o *bitrate* do *bytestream* gerado. Com isso, a escolha da progressão geométrica é uma forma de aproximar o crescimento exponencial esperado para o *bitrate* bem como atender ao padrão de aumento multiplicativo de taxa (*Multiplicative Increase*) adotado pelo TFMCC.

Uma propriedade relevante do vídeo SVC é: se retirada a restrição de limite de atraso estrutural, ou seja, se permitida a utilização de quadros do tipo B, tem-se, de modo geral, que a variação do *bitrate* em função da camada temporal T (mantidas as camadas D e Q) é baixíssima. O incremento no *bitrate* da camada T_i, $i > 0$, é menor quanto maior o i. A razão é que a compensação de movimento na predição inter-quadro torna-se mais eficiente devido à proximidade temporal das imagens. Neste cenário, sem limite de atraso estrutural, não compensa escalonar os blocos de vídeo na dimensão temporal T. Recomenda-se manter todas as camadas T e ajuste apenas D e Q. A redução da dimensão T resulta em pouca diminuição do *bitrate* e em grande impacto na experiência do usuário, pois a redução do FPS causa a sensação de que o vídeo está “travando”. Ainda, o cenário em questão dificulta a conciliação com a proposta de salto geométrico no *bitrate*.

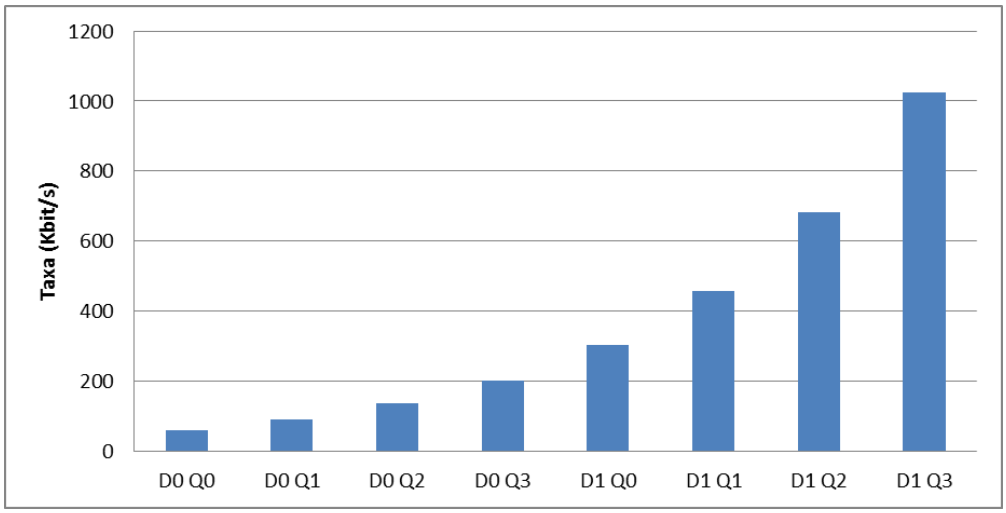


Figura 31 – Progressão do *biterate* das camadas SVC. *D = Dependency layer, Q = Quality layer*

Para R = 1,5, L_0 = 60 Kbit/s e 8 camadas SVC considerando dois níveis de escalabildiade espacial D e 4 níveis de escalabilidade de qualidade Q, tem-se o resultado mostrado na Figura 31.

É importante que o *bitrate* de todas as camadas seja mantido constante ao longo da transmissão. Uma forma de atender esse requisito de CBR (*Constant Bitrate*) é ajustando-se dinamicamente o QP de cada camada durante a codificação de cada bloco SVC. Vale ressaltar que a PSNR é aproximadamente inversamente proporcional ao QP.

O trecho abaixo propõe um algoritmo para obtenção de CBR em SVC:

Tabela 13 – Algoritmo de compressão CBR para SVC

```
Define NUM_CAMADAS = número de camadas SVC.

Define MISMATCH    = erro percentual tolerado para o bitrate real obtido em cada
                     camada. Padrão: 5.

Define MAX_ITR     = número máximo de iterações para calculo do QP cada camada.
                     Padrão: 15.

Define INITIAL_QP  = valor inicial de QP no processo de iteração. Padrão: 40.

Define MIN_QP      = valor mínimo de QP que pode ser associado a uma camada. Padrão: 5.

Define MAX_QP      = valor máximo de QP que pode ser associado uma camada. Padrão: 50.
                     Verifica-se, na prática, que valores acima de 50 não resultam
                     em diminuição do bitrate.

Define DELTA_QP    = base de cálculo para o incremento do QP em iteração. Padrão: 8.

Define MIN_DQP     = incremento mínimo do QP em cada iteração. Padrão: 2.

CodificarCBR (bloco)
{
   Cria lista targetQP de tamanho NUM_CAMADAS. Essa lista conterá resultado do cálculo
   de obtenção de QP para cada camada. O primeiro elemento da lista é referenciado
   por targetQP[0]
```

```
   Repete para cada numLayer de 1 a NUM_CAMADAS, inclusivo
   {
      Atribui targetRate = taxa desejada para a camada numLayer

      Atribui minTargetRate = targetRate * (100 - MISMATCH) / 100
      Atribui maxTargetRate = targetRate * (100 + MISMATCH) / 100
      Atribui minRate   = 0
      Atribui maxRate   = 0
      Atribui maxQP     =  1000
      Atribui minQP     = -1000
      Atribui currQP    = INITIAL_QP

      Repete para cada itr de 0 a MAX_ITR, exclusivo
      {
         Atribui targetQP[numLayers - 1] = currQP

         Codifica o bloco com número de camadas SVC limitado a numLayer e utilizando
         como parâmetros de QP o(s) valor(es) respectivo(s) em targetQP. O bitrate
         resultante é atribuído à variável rate

         Se (currQP >= MAX_QP ou currQP <= MIN_QP ou
             (rate >= minTargetRate e rate <= maxTargetRate))
         {
            Fim da repetição corrente
         }

         Se (rate < targetRate)
         {
            Atribui minRate = rate
            Atribui maxQP = currQP
         }
         Caso contrário
         {
            Atribui maxRate = rate
            Atribui minQP   = currQP
         }

         Se (maxQP diferente de 1000 e minQP diferente de -1000)
         {
            Atribui deltaQP = maxQP - minQP
            Atribui deltaQP = deltaQP * logarítmo(maxRate / targetRate)
            Atribui deltaQP = deltaQP / logarítmo(maxRate / minRate)
            Atribui currQP  = minQP + deltaQP
         }

         // reduz o QP
         Caso contrário, se (maxQP diferente de 1000)
         {
            Atribui deltaQP = DELTA_QP * logarítmo(rate / targetRate)
            Atribui deltaQP = máximo entre deltaQP e MIN_DQP
            Atribui currQP  = currQP - deltaQP
            Atribui currQP  = máximo entre currQP e MIN_QP
         }

         // aumenta o QP
         Caso contrário
         {
            Atribui deltaQP = DELTA_QP * logarítmo(rate / TARGET_RATE[numLayers - 1])
            Atribui deltaQP = máximo entre deltaQP e MIN_DQP
            Atribui currQP  = currQP + deltaQP
            Atribui currQP  = mínimo entre currQP e MAX_QP
         }
      }

   }

   Retorna a lista targetQP
}
```

No esquema CBR o vídeo terá *bitrates* bem definidos e constantes para cada camada SVC. O protocolo TFMCC fica então com o objetivo de selecionar o melhor *bitrate* para a largura de banda disponível no canal. Uma vez feito seu papel (o TFMCC), a qualidade

PSNR do vídeo irá variar ao longo dos blocos (devido à variação do QP) de forma a manter constante o *bitrate* sugerido pelo controle de congestionamento. Esse esquema é conveniente, pois garante a estabilidade do protocolo TFMCC.

Na abordagem VBR (*Variable Bitrate*), que pode ser implementada mantendo-se QP constante ou PSNR constante, tem-se que o *bitrate* pode variar bruscamente de um bloco para outro em função, por exemplo, da variação de entropia do vídeo. Uma consequência desse processo é que, mesmo o módulo Adaptador SVC alternando de camada a cada bloco, ocorra ainda variação significativa e rápida da taxa de transmissão, impedindo que o TFMCC estabilize sua decisão sobre a taxa sugerida com base na análise da taxa de recepção medida nos nós filhos. Ou seja, é estabelecido um desvanecimento rápido do *bitrate* quando comparado com os períodos de análise do TFMCC.

5.9 - CONTROLE DA FILA DE REPRODUÇÃO

O módulo Reprodutor de Vídeo (RV) possui o componente Fila, apresentado na Figura 24, responsável por resolver a sincronização de partida, correção de *jitter*, tolerância a flutuações na capacidade computacional do nó e ordenamento dos blocos.

A Fila possui duas operações fundamentais: *Adicionar(bloco)*, responsável por adicionar um bloco na Fila; *Retirar()*, responsável por obter o bloco mais antigo. Os trechos abaixo propõem algoritmos para esses procedimentos:

Tabela 14 – Algoritmo para sincronização da fila de blocos

```
Adicionar (bloco)
{
   Bloco de seção crítica (monitor com trava de exclusão mútua na fila ordenada)
   {
      Se SSRC mudou
      {
         Guarda novo SSRC
         Remove todos os blocos da fila ordenada
         Define buffering = verdadeiro
         Define lastSeqNumDelivered = bloco.seqNum - 1
      }

      Se tamanho da fila > MAX_QUEUE_LEN
      {
         Remove o bloco mais antigo da fila ordenada
         Adiciona o bloco recebido na fila ordenada
         Fim
      }

      Se bloco.seqNum < lastSeqNumDelivered
      {
         Descarta o bloco (muito atrasado)
         Fim
      }

      Adiciona o bloco recebido na fila ordenada

      Se tamanho da fila >= BUFFERING_THRESHOLD e buffering = verdadeiro
      {
         Define buffering = falso
```

```
            Marca o tempo zero do playback (initialTimestamp = bloco.timestamp)
            Acorda as threads do monitor da fila ordenada
        }
    }
}
```

```
Retirar ()
{
   Bloco de seção crítica (monitor com trava de exclusão mútua na fila ordenada)
   {
      Enquanto (buffering = verdadeiro) ou (fila ordenada vazia)
      {
         Suspende a thread com monitor na fila ordenada
      }

      Retira bloco mais antigo da fila ordenada

      Incrementa o bloco.timestamp em (-initialTimestamp)

      Define lastSeqNumDelivered = bloco.seqNum

      Se tamanho da fila ordenada = 0
      {
         Define buffering = verdadeiro
      }

      Retorna o bloco
   }
}
```

5.10 - RETRANSMISSÃO EM CAMADA BASE

Propõe-se aqui um esquema de retransmissão de bloco que objetiva reduzir os efeitos causados pela perda de pacotes.

O esquema inspira-se na seguinte ideia: se a taxa de transmissão decresce abruptamente em função do comportamento do TFMCC, então significa que há perda razoável de pacotes (fragmentos) e, por consequência, de blocos inteiros no(s) receptor(es). Faz-se, então, retransmissão da camada base SVC dos últimos blocos enviados, conforme descrito a seguir.

Considera-se para efeito de configuração que um decréscimo abrupto da taxa ocorre quando esta é reduzida pra menos que 70% de seu valor anterior. O sistema é pouco sensível a este fator, pois: o protocolo TFMCC *sender* costuma, em caso de detecção de problemas no enlace, decrescer a taxa para 50% de seu valor. O protocolo TFMCC *receiver* sugere, em caso de detecção de perda de pacotes, uma taxa baseada na função de aproximação do Reno TCP e esta função tem comportamento bastante agressivo. De modo geral, na presença de perda de pacotes, a taxa costuma ser reduzida para 50% ou menos de seu valor.

Ocorrido o decréscimo abrupto da taxa, gera-se então o evento aqui chamado de "sinalização de retransmissão rápida". Este evento dispara a retransmissão dos últimos blocos enviados para os nós filhos. Recomenda-se uma retransmissão retroativa de 3 blocos. É interessante que esta quantidade represente um trecho de vídeo de duração de aproximadamente igual ao *buffer* de *playback* subtraído do RTT médio.

Os fragmentos de retransmissão são marcados com cabeçalho *FastRetransmission* e enviados imediatamente. Pode ser utilizado qualquer um os bits entre 2 e 8 da Tabela 11 para esta sinalização.

O Remontador de Blocos, ao receber os fragmentos marcados com *FastRetransmission*, entrega-os imediatamente para a aplicação e não expira, caso houver, o compartimento de remontagem referente ao bloco. Assim, a perda de pacotes continuará sendo computada normalmente.

6 - IMPLEMENTAÇÃO

6.1 - INTRODUÇÃO

Esse capítulo é dedicado a uma breve apresentação de soluções técnicas adotadas durante a implementação da arquitetura proposta.

Os trabalhos coletados, referências para este trabalho, se baseiam todos em simulação. É utilizado, de modo geral, o *software* NS-2 (*Network Simulator 2*). A simulação é uma metodologia válida para mostrar ou demonstrar uma proposta. Neste trabalho, entretanto, deseja-se verificar resultados através de execuções reais, sem auxílio de simulações, de forma que a transmissão do vídeo de tempo real ocorra, de fato, em tempo real. O objetivo dessa metodologia é reduzir falhas conceituais muitas vezes percebidas quando o esquema é implementado em sua completude e interage com condições reais de ambiente no momento de sua implantação e execução.

A quase inexistência de códigos abertos relacionados a protocolos *TCP-Friendly multicast*, incluindo o TFMCC, e, principalmente, a pouca oferta de codificadores SVC de bom desempenho são fatores que dificultaram substancialmente a construção da prova de conceito.

Todos os módulos da arquitetura, com exceção do codificador SVC, JSVM Extractor e do substrato P2P, foram implementados sem códigos de terceiros. A Tabela 15 resume a abordagem considerada em cada módulo.

Tabela 15 – Implementação dos módulos na prova de conceito

Módulo	Abordagem técnica
Reprodutor de Vídeo (RV)	Implementado em Java com auxílio de OpenGL (*Open Graphics Library*), da biblioteca LWJGL (*Lightweight Java Game Library*), da biblioteca Eclipse SWT (*Standard Widget Toolkit*) e da linguagem C para integração do decodificador SVC via JNA (*Java Native Access*).
Remontador de Blocos (RB)	Implementado puramente em Java.
Fragmentador de Blocos (FB)	Implementado puramente em Java.
Adaptador SVC	Implementado em Java com auxílio do *software* JSVM Extractor.
TFMCC (sender e receiver)	Implementado puramente em Java.
Empacotador RTP	Implementado puramente em Java.

Substrato P2P	Utilização do Pastry e do Scribe, com adaptações.

6.2 - MÓDULO REPRODUTOR DE VÍDEO

O módulo RV é composto de três importantes componentes – a Fila de blocos SVC, o Decodificador SVC e Exibidor de quadros – e dois temporizadores de sincronização – T1 e T2, conforme já apresentados na Figura 24.

Adota-se uma estrutura *multi-thread* para implementação do RV. O fluxo de dados entre os componentes pode ser representado por uma esteira de fábrica ou *pipeline:*

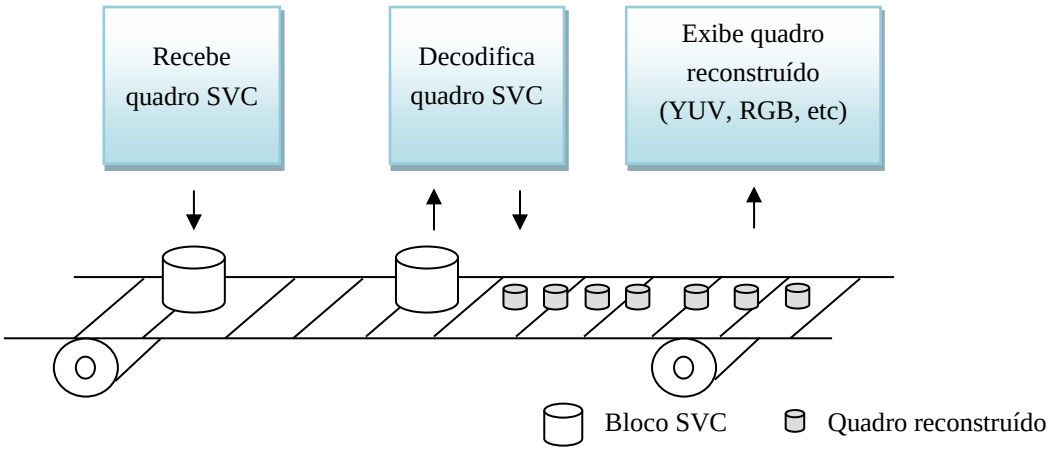


Figura 32 – Reprodução paralelizada (*pipeline*)

No esquema *pipeline* cada um dos três componentes é executado em paralelo com respeito aos demais. Enquanto um quadro é exibido no último componente, o próximo quadro já está em fase de decodificação no segundo componente e, por sua vez, outro novíssimo quadro pode estar sendo entregue para o primeiro componente. A esteira se movimenta conforme a velocidade de reprodução do vídeo, sendo responsável, então, pela sincronização entre recebimento, decodificação e exibição.

Existem dois gargalos:

(1) O Decodificador SVC, que exige alto custo de processamento. O tempo gasto do bloco dentro deste componente não pode ultrapassar o período de 1 bloco, ou seja, 1/F1, onde F1 é a frequência de blocos do sistema;
(2) O Exibidor de quadros, que exige alto custo de memória, tanto no sentido de quantidade quanto no sentido de velocidade de leitura e de escrita.

Para exemplificar o gargalo (2) supracitado, considere o cenário da Tabela 16:

Tabela 16 – Cenário de configuração do sistema

Parâmetro	Valor
F1	3
Duração do bloco	1 / 3 = 333ms
Número máximo de quadros por bloco	8 (tamanho do GOP na qualidade máxima SVC)
Qualidade espacial máxima	4CIF 704 x 576 (405.504 pixels)
Representação do quadro reconstruído	RGBA (4 bytes ou 32bits / pixel)

A taxa de quadros por segundo quando da qualidade máxima SVC e em condições normais da transmissão vale:

$$FPS = 3 * 8 = 24 \text{ quadros/segundo}$$

O tamanho do quadro reconstruído é:

$$S = 405504 * 32 = 12976128 \approx 13 \text{ Mbit}$$

E a taxa de transferência de dados vale:

$$R_T = 24 * S = 311427072 \approx 311 \text{ Mbit/s}$$

Essa taxa é de fato crítica, pois o próprio acesso de escrita na memória RAM tangencia esse limite de velocidade.

A implementação, portanto, deve ser otimizada para evitar duplicação dos dados nas interfaces entre os componentes do RV ou até mesmo dentro deles. Por exemplo, se a informação é copiada do decodificador para o exibidor, tem-se então uma taxa de escrita na memória de 311 * 2 = 622 Mbps.

No sentido de reduzir o custo de acesso à memória, propõe-se uso de memória compartilhada entre o decodificador SVC e o exibidor, conforme a seguir: duas regiões compartilhadas são dedicadas para armazenamento de bloco de vídeo reconstruído. Essas regiões são chamadas de M0 (primeira memória) e M1 (segunda memória). O tamanho das regiões é ajustado dinamicamente para comportar o maior bloco de vídeo reconstruído até então. Existem dois semáforos de exclusão mútua que garantem que: enquanto memória M_i é lida pelo exibidor de quadros, a outra, M_{1-i}, é gravada pelo decodificador SVC. Uma chave lógica, de domínio do temporizador de decodificação T1, controla o ponteiro de gravação. Outra chave lógica, de domínio do temporizador de decodificação T2, controla o ponteiro de leitura. Essas chaves trabalham de maneira alternada sem intersecção, isto é,

não há momento em que ambos os ponteiros apontam para uma mesma região M_i. O módulo RV baseado em memória compartilhada está representado pela Figura 33.

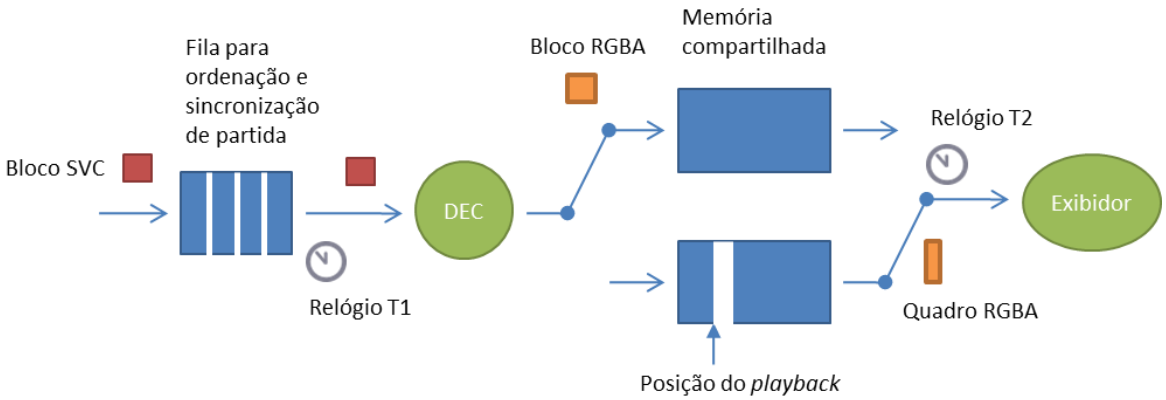


Figura 33 – Implementação do RV com memória compartilhada

A fila de blocos acumula carga até atingir determinado limiar. Ocorrido esse evento, o relógio de baixa resolução é acionado para que se inicie o consumo da fila. Os pacotes serão retirados por esse temporizador conforme a taxa de transmissão de blocos de vídeo (F1).

Ocorrendo sobrecarga da fila de pacotes (*overflow*), os novos pacotes são recebidos e os antigos são descartados. Essa sobrecarga pode ser proveniente de falha no agendamento de *threads* pelo sistema operacional. Outro motivo seria a baixa capacidade de processamento do nó, o que gera lentidão na decodificação do vídeo e consequentemente a não liberação da memória compartilhada para acesso de gravação no próximo período 1/F1.

O relógio de alta resolução sincroniza o tempo de exibição de cada quadro. Esse relógio, em cada ciclo, acessa, na memória compartilhada, o trecho relativo ao quadro "do momento" para exibição em tela.

Além dos dois relógios T1 e T2, no esquema de reprodução baseado em OpenGL existe um terceiro relógio, responsável por atualizar a imagem na tela. A frequência deste relógio deve ser superior ao maior FPS possível no sistema, de forma que os quadros não sejam descartados inutilmente após decodificação. Esse relógio do OpenGL é constante e não depende de T2, o qual varia conforme a escalabilidade temporal SVC. O casamento de resolução de ambos os relógios resulta no efeito chamado de *pull-down*, mostrado Figura 34.

No primeiro gráfico da Figura 34 tem-se a representação das memórias M0 e M1 e o intervalo de tempo em que cada uma esteve alocada para que o decodificador SVC gravasse o bloco reconstruído.

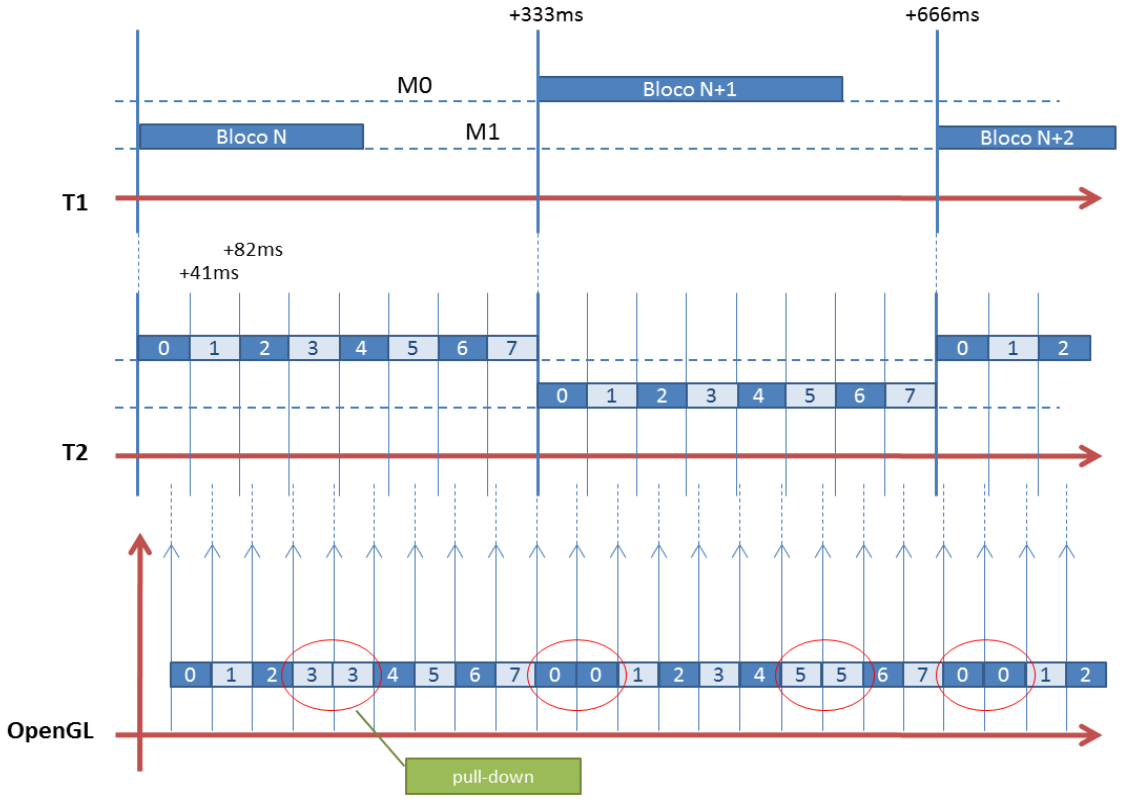


Figura 34 – Efeito *Pull-down* no casamento do relógio T2 com o do OpenGL

No segundo gráfico da Figura 34 tem-se os intervalos de tempo em que as memórias foram lidas pelo temporizador T2, lembrando que o bloco reconstruído pelo decodificador em determinado período de T1 será lido pelo T2 somente no próximo período T1. Isso se deve ao chaveamento alternado dos ponteiros de leitura e de escrita de M0 e M1.

No terceiro gráfico da Figura 34 tem-se a amostragem feita pelo OpenGL no último quadro obtido por T2. Observa-se que alguns quadros serão exibidos duas vezes, que é o efeito *pull-down*.

A Figura 35 mostra o sincronismo entre todos os temporizadores envolvidos no RV considerando os estados da Fila de blocos.

Com respeito ao exibidor de quadros, as implementações baseadas em Swing, Java 2D e JOGL (*Java Binding for the OpenGL API*) resultaram em lentidão. A solução satisfatória foi com o uso da biblioteca Eclipse SWT em conjunto com LWJGL.

A integração do decodificador SVC com Java foi experimentada nas seguintes técnicas:

(a) Integração por biblioteca nativa em C via JNA (*Java Native Access*);
(b) Integração através de arquivos *Pipe* (FIFO) do sistema operacional;
(c) Integração por entrada e saída padrão do *kernel*;

(d) Integração via socket.

A opção (b) é inviável em função da alta taxa de transferência do vídeo reconstruído. As opções (c) e (d) envolvem complexidade computacional adicionada pelo sistema operacional, não sendo, portanto, soluções muito eficazes. A melhor integração obtida foi a opção baseada em JNA, pois permite que o C e o Java façam acesso direto à uma memória compartilhada. É possível que a alocação de memória seja feita previamente pelo Java ou dinamicamente pela biblioteca C.

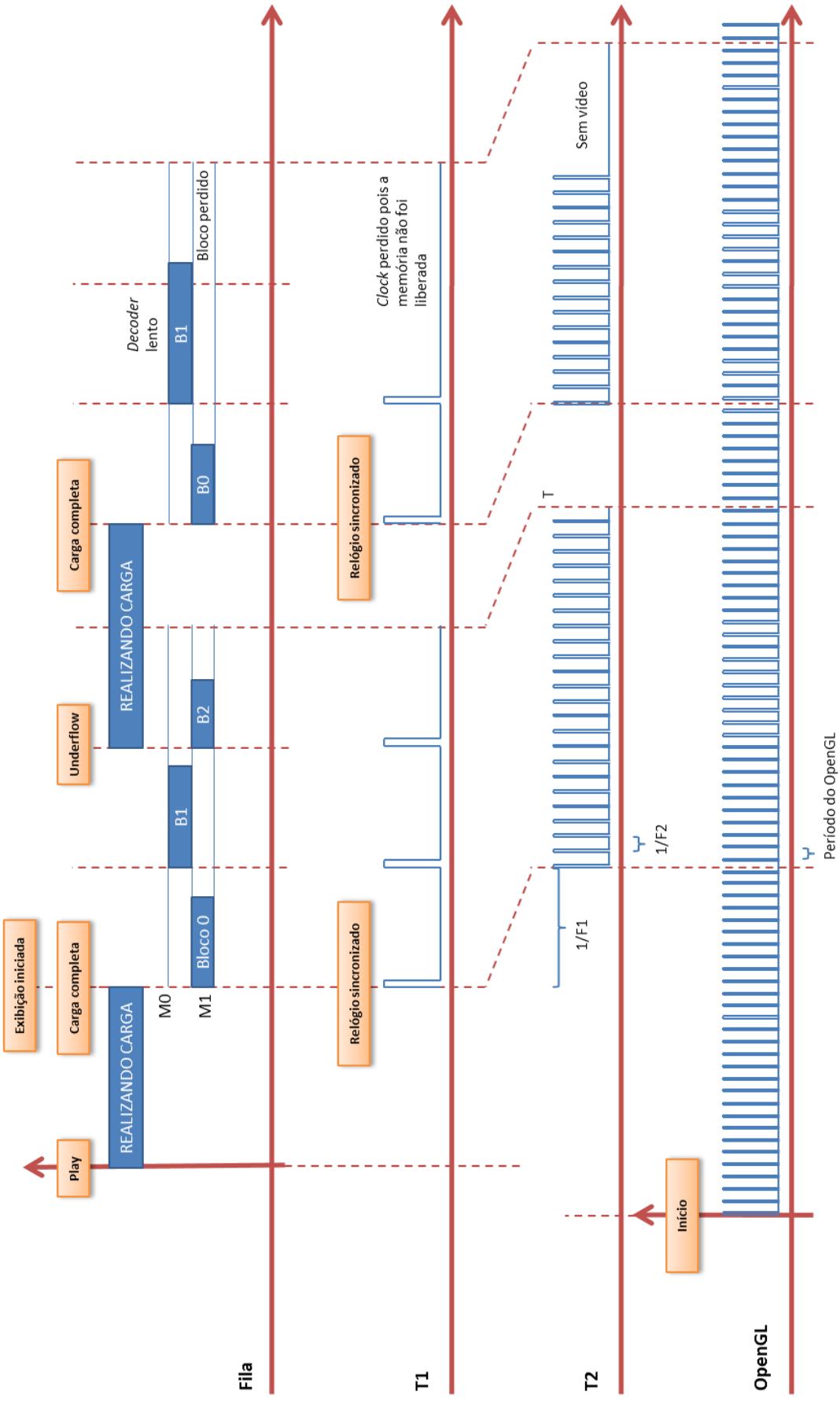


Figura 35 – Representação na linha do tempo do RV

A Figura 12 mostra o resultado de desempenho decodificação do MainConcept sendo executado através de JNA. Utilizou-se pré-alocação de memória via Java. A alocação de memória compartilhada via C com alinhamento de ponteiro para otimização SSE2 e várias outras diretivas de otimização de compilação não resultou no mesmo desempenho do Java. Observa-se que, para a resolução 4CIF, o FPS médio obtido foi de 91 e mínimo de 62. Ou seja, o custo de processamento especificamente dentro do componente de decodificação SVC do módulo RV é responsável por consumir grande parte do recurso de processamento do PC considerado em questão. Se o vídeo tivesse taxa de 60 FPS, por exemplo, não havia tempo disponível para executar as demais tarefas do sistema, como: exibir os quadros em tela, gerir a rede P2P, remontar e fragmentar pacotes, controlar a sincronização *threads*, responder a eventos do usuário, etc. Daí a criticidade em elaborar uma arquitetura que evite ao máximo processamento adicional do *stream* de vídeo.

É importante citar que o módulo RV foi implementado também com uso das ferramentas GStreamer e Microsoft Direct Show. Elas incluem funcionalidades para gerência de fluxo de dados entre elementos (fases) do processo de reprodução, para exibição em tela e para sincronização de tempo. Essas ferramentas oneraram, entretanto, maior carga de processamento com respeito à implementação pura do RV, que é mais simples e especializada para a arquitetura proposta. Outro problema é que o GStreamer e o Microsoft Direct Show não são preparados para FPS dinâmico, ou seja, mudança de escalabilidade temporal durante a reprodução do vídeo.

6.3 - MÓDULO ADAPTADOR SVC

O módulo de adaptação dos blocos de vídeo SVC foi implementado tendo por base o *software* JSVM Extractor. Apesar de o JSVM Decoder ser extremamente lento e inviável para execução em tempo real, o que exigiu a utilização de uma solução proprietária no RV, JSVM Extractor, por sua vez, mostrou-se razoavelmente rápido.

A integração do JSVM Extractor com Java deu-se através de arquivos temporários. Uma vez determinada, em função da taxa sugerida pelo TFMCC, a tupla (D, T, Q, *bitrate*) para a qual o bloco deve ser escalonado, executa-se o JSVM Extractor com os parâmetros citados na Tabela 17.

Tabela 17 – Parâmetros do JSVM Extractor para adaptação de bloco

```
BitStreamExtractorStatic –l D –t T –f Q –keepf
```

A opção "–keepf" faz o JSVM Extractor manter todas as camadas Q inferiores, mesmo aquelas não obrigatórias para a decodificação do bloco. O objetivo é que a degradação da qualidade do bloco ocorra mais lentamente ao longo dos saltos na árvore P2P. O algoritmo de compressão CBR proposto na seção 5.8 prever a manutenção de todas as camadas inferiores em cada escalonamento.

6.4 - MÓDULO SUBSTRATO P2P

O alicerce de comunicação entre os nós da rede é feito pelo Pastry, com adição de canal de retorno (*reserve path*) para *feedback* do TFMCC.

A sobreposição para gerência de árvores é dada pelo Scribe, com alteração da API (*Application Programing Interface*) para que seja possível controlar o mecanismo de *push*; Esse é um requisito para integração entre a rede P2P e o módulo Adaptador SVC: os fragmentos recebidos por um determinado nó não podem ser comutados imediatamente para os nós filhos, como é feito no Scribe. É necessário que o nó acumule os fragmentos até que o bloco seja reconstruído pelo módulo Remontador. Uma vez reconstruído, o bloco será processado pelo módulo Adaptador SVC e depois fragmentado novamente. São esses novos fragmentos que serão encaminhados para os nós filhos via *push*. Para isso, a classe principal do Scribe, *ScribeImpl*, foi modificada para permitir a configuração do *push* em modo manual. Pacotes outros que não os fragmentos de bloco SVC, como, por exemplo, sinalizações, não são afetados por essa configuração.

6.5 - PROBLEMAS E SOLUÇÕES

A seguir estão elencados alguns problemas encontrados, conceituais e/ou práticos, e soluções adotadas.

1) PROBLEMA DE BURACOS NO *SEQUENCE NUMBER* RTP DEVIDO AO ESCALONAMENTO SVC

Solução: Cada nó deverá fazer a renumeração dos pacotes RTP.

Exemplo: Suponha que o bloco 1 chegou, em determinado nó, com fragmentos numerados de 1 a 7. Depois de escalonado, o número de fragmentos caiu para 5. A partir dos próximos blocos recebidos, todos os fragmentos terão o *sequence number* subtraído de 2 e esse *offset* é cumulativo. Por exemplo: se o bloco 2 for também escalonado e tiver uma redução de 9 para 3 fragmentos, então o *offset* deverá ser aumentado adequadamente, de 2 para 8.

Característica: A mensuração do número de fragmentos perdidos, que é feita pela contagem de buracos no *sequence number*, não seria corretamente calculada sem o *offset* acima aplicado. O *offset* mantém ainda essa mensuração correta mesmo se um ou mais blocos inteiros forem perdidos (fácil provar).

2) PROBLEMA DA DISTÂNCIA ENTRE "NÓ FONTE DO VÍDEO" E "NÓ RAIZ DA ÁRVORE"

Detalhamento do problema: O Scribe conforme é concebido não garante que a distância entre a fonte de vídeo e a raiz seja pequena. É possível que a fonte seja um nó folha da árvore, o que seria péssimo, pois o vídeo já iniciaria na rede com a pior qualidade e assim todas as instâncias de TFMCC em cada nível da árvore não faria sentido.

Solução: Considerando que em geral uma empresa que deseja transmitir o vídeo detém controle administrativo do nó fonte de vídeo, pode-se admitir que ela também desejará manter controle do nó raiz, para que seja garantido pelo menos o início da transmissão do vídeo. Nesse sentido, pode-se considerar a existência de uma rede confiável dentro da empresa e que o vídeo iniciará a partir dela.

Implementação: Fixar a faixa (*range*) de possíveis identificações (*node Ids*) para tópicos Scribe através de uma implementação própria de "ID Generator" de forma que somente os nós internos da empresa tenham ID numericamente próximos do ID do tópico.

Característica: A rede continua tolerante a falha no nó raiz. Se todos os nós dentro da empresa falharem, o próximo nó (numericamente mais próximo) da rede externa será eleito automaticamente pelo Scribe e permanecerá como raiz somente até que algum nó interno seja reestabelecido.

3) PROBLEMA DA PERDA DE BLOCO DE VÍDEO QUANDO ALGUM NÓ ENTRA OU SAI DA REDE, AFETANDO TODOS OS NÓS FILHOS SUBSEQUENTES

Solução: Definir critérios de entrada e de saída de nó.

Critério de saída: O nó que deseja sair primeiramente habilita a configuração de *push* imediato de qualquer fragmento recebido, que é o modo de operação normal do Scribe. Faz-se, então, o *push* de todos os fragmentos retidos em memória no Remontador. Após isso, o nó sai da rede.

Critério de entrada: O nó que entrar na árvore Scribe deve ficar inicialmente no modo apenas escuta, ou seja, fazendo *push* imediato de qualquer fragmento recebido e não os

mantendo em memória no Remontador. Deve-se registrar o número do primeiro bloco percebido. O nó deve permanecer no modo escuta até que seja percebido o primeiro bloco cujo número seja o "número do primeiro bloco percebido" somado ao "número de compartimentos de remontagem", ou seja, *blockNum' = blockNum + C*. A partir de desse bloco o nó deve entrar no modo de *push* manual, em que haverá retenção dos fragmentos para remontagem, adaptação e posterior fragmentação.

4) AUMENTO DO ATRASO NO RECEBIMENTO DE FRAGMENTO QUANDO HÁ INGRESSO DE NÓ EM NÍVEIS SUPERIORES DA ÁRVORE OU REDUÇÃO DO ATRASO QUANDO HÁ SAÍDA

Solução: Esse é um problema inerente à transmissão em árvore. Após uma determinada quantidade de entradas (igual ao número de blocos na Fila do RV) ocorrerá um *underflow* e o *player* entrará em estado de *buffering* novamente. Algo semelhante ocorre sempre que um nó em nível hierárquico da rede sai: em um determinado momento ocorrerá *overflow*, pois os fragmentos chegarão antecipadamente nos nós filhos.

6.6 - IMAGENS DO SOFTWARE IMPLEMENTADO

A Figura 36 mostra um retrato do *software* implementado executado em um PC Intel Core2 Duo T7500 2,2 GHz, 2 GB RAM, Windows Vista 32 bits.

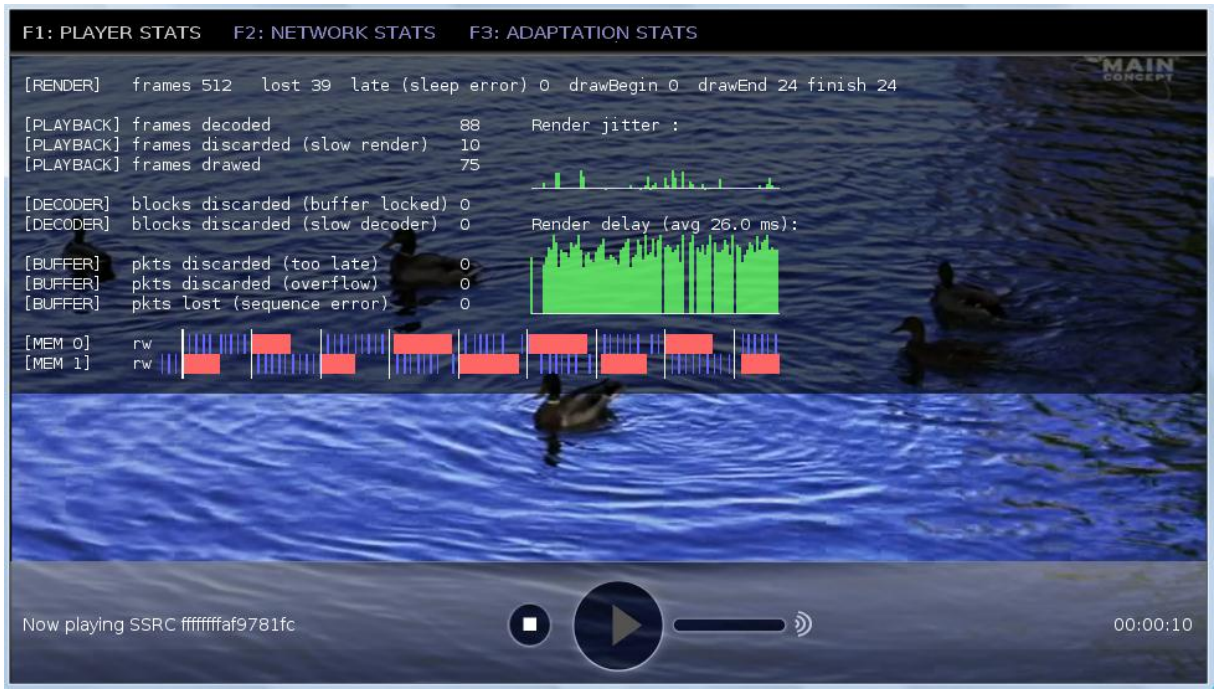


Figura 36 – Retrato do *software* no Windows Vista (cenário normal de execução)

O painel de estatísticas do RV está habilitado. Ele mostra informações sobre o andamento da reprodução:

- *Render jitter:* atraso com relação ao momento esperado da exibição de um quadro pelo relógio do OpenGL;
- *Render delay:* tempo gasto, dentro do OpenGL, para exibição de um quadro;
- *MEM0 e MEM1:* são as memórias compartilhadas explanadas na seção 6.2. O gráfico é inspirado na Figura 34. As barras verticais brancas delimitam os períodos do relógio T1 (frequência de blocos por segundo). Os retângulos vermelhos (largos) indicam o intervalo de tempo que a memória esteve alocada para escrita pelo decodificador SVC. Os retângulos azuis (estreitos) indicam o intervalo de tempo que a memória esteve alocada para leitura pelo exibidor de quadros (OpenGL);
- *Playback:* estatísticas do exibidor de quadros:
    - Número de quadros decodificados;
    - Número de quadros descartados devido a lentidão no sistema. O critério de descarte é: a exibição é cancelada caso iniciar após 1 período completo de quadro (1/FPS);
    - Número de quadros desenhados.
- *Decoder:* estatísticas do decodificador SVC:
    - Número de blocos descartados devido à não liberação da memória compartilhada (M0 ou M1) para escrita;
    - Número de blocos descartados devido a lentidão no sistema. O critério de descarte é: a decodificação é cancelada caso iniciar após ¼ do período do bloco.
- *Buffer:* estatísticas da Fila:
    - Número de blocos descartados devido a atraso de chegada;
    - Número de blocos descartados devido a sobrecarga (*overflow*);
    - Número de blocos perdidos (buracos no *sequence number* RTP).

A Figura 37 mostra um retrato sob condições em que não há recursos computacionais suficientes para decodificar o vídeo.

A Figura 38 mostra um cenário em que o sistema está parado e apenas a *thread* do OpenGL está em execução. Neste exemplo o sistema está sendo executado no Windows XP. Observa-se uma instabilidade no *jitter* do exibidor de quadros, que decorre da imprecisão do sistema operacional em gerenciar interrupções (*sleep*) e levantamentos (*wake up*) de *threads*. Os relógios T1 e T2 também são afetados por esse problema. No gráfico *render jitter*, o eixo das ordenadas tem faixa de 33ms para cima do eixo e 15ms para baixo.

Figura 37 – Retrato do *software* no Windows Vista (lentidão no *decoder* SVC)

Figura 38 – Retrato do *software* no Windows XP (sem *hint ForceTimeHighResolution*)

Na Figura 39 tem-se a execução com o parâmetro de otimização (*hint*) da JVM (*Java Virtual Machine*) chamado *ForceTimeHighResolution*. O sistema operacional adquire maior previsibilidade no comportamento, porém ainda existe o *jitter*. Windows XP é capaz de acordar *threads* somente em momentos múltiplos de 10ms com respeito ao relógio do sistema operacional. Em [51] tem-se uma análise detalhada dessa questão.

Figura 39 – Retrato do *software* no Windows XP (com *hint ForceTimeHighResolution*)

7 - APRESENTAÇÃO E DISCUSSÃO DE RESULTADOS

7.1 - INTRODUÇÃO

No presente capítulo são apresentados os resultados relativos a testes realizados considerando, em síntese: 4 cenários de teste; 4 sequências de vídeo codificadas em H.264 SVC nas resoluções QFIC e CIF; métricas de avaliação de desempenho, tais como PSNR e perda de pacotes. Adicionalmente é apresentado o algoritmo para medição de taxa que se fez necessário a fim de avaliar a efetividade da proposta em uma rede simulada.

Cenários

A prova de conceito implementada foi submetida a testes em 4 cenários distintos, conforme sumariza a Tabela 18:

Tabela 18 – Cenários de teste

Cenário	Breve descrição	Objetivo
1	Considera-se uma rede Scribe na qual, periodicamente, um novo nó ingressa conectado como filho do nó raiz. Os módulos Adaptador SVC e TFMCC estão desativados.	Verificar a qualidade do vídeo nos nós receptores ao longo do tempo em função do crescimento da rede e consequente sobrecarga do nó raiz.
2	Baseado no cenário 1 porém com os módulos Adaptador SVC e TFMCC habilitados.	Verificar o ganho de qualidade de vídeo obtido com ativação do módulo de Adaptação SVC em conjunto com o protocolo TFMCC.
3	Baseado no cenário 2 porém com adição do esquema de retransmissão em camada base.	Verificar qual a melhoria na qualidade de vídeo trazida pela proposta de retransmissão de camada base.
4	Considera-se um nó Scribe que possui apenas 1 filho. Deseja-se analisar o enlace.	Verificar o comportamento da responsividade e da agressividade do TFMCC em função de diferentes padrões de perda de pacotes aplicados em um enlace da rede Scribe.

Em todos os cenários se considerou:

- Fragmentos de 3 KB (3072 bits), que é, aproximadamente, o tamanho do bloco de vídeo na menor qualidade SVC (apenas camada base) conforme as configurações de codificação SVC consideradas;
- 2 compartimentos no Remontador de Blocos;
- Blocos contendo 8 quadros com taxa de 3 blocos/s, isto é, 24 FPS;
- Parâmetros do protocolo TFMCC:
    - k = 4 (RTTs for rate measurement);
    - N = 50 (estimated upper bound on the number of receivers);
    - *n* = 6 (número de pesos usado no cálculo de *average loss interval*);
    - *min(RTT_max)* = 333 ms (valor mínimo de *RTT_max* considerando que a taxa mínima é 3 blocos/s). *RTT_max* é o maior valor RTT entre o nó raiz e seus os receptores medido pelo TFMCC em cada ciclo de *feedback*;
    - *g* = 0,1 (utilizado no cálculo de *suppress rate*);
    - ts_gran = 10 (granularity of the sender's system clock).
- Outros ajustes no protocolo TFMCC:
    - Utilização de *RTT_max* no lugar de *R* na equação da taxa, seção 2.1 de [48]. Justificativa: *R* tende à zero em várias situações causando a taxa *X_r* alcançar valores excessivamente altos. Exemplos são: redes virtuais emuladas, nós dentro da mesma máquina (PC) e enlaces curtos;
    - Utilização de *RTT_max* em vez de *R* no temporizador de *feedback*. Mesma justificativa do caso acima: valores pequenos de *R* (da ordem de alguns milissegundos) causaria explosão de pacotes de *feedback*. O temporização do *feedback_round (fb_nr)* feita no transmissor já se baseia no *RTT_max*, conforme definido na RFC;

Os testes ocorreram em um PC Intel Core i7-2630QM, 4GB RAM, Windows 7 64 bits, de forma que os nós da rede Pastry foram processos executados na mesma instância do sistema operacional Windows.

Sequências de teste

As sequências de vídeo consideradas foram: (a) *Foreman*; (b) *Waterfall*; (c) *Container*; e (d) *News* (fonte: http://trace.eas.asu.edu). Todos os vídeos foram codificados em H.264 SVC conforme o paradigma de compressão em blocos independentes.

Cada bloco contém 2 GOPs inteiros de 4 quadros, hierárquico, dotado das escalabilidades SVC elencadas na Tabela 19. Os vídeos possuem 32 blocos (10,6s) e são reproduzidos

continuamente durante os testes. São consideradas duas camadas de qualidade espacial (QCIF e CIF) com 3 melhoramentos MGS cada.

Tabela 19 – Camadas SVC do cenário 3

Camada	**Resolução (pixels)**	**FPS**	**Taxa (Kbit/s)**
0 (base layer)	QCIF 176x144	24	60
1	QCIF 176x144	24	85
2	QCIF 176x144	24	120
3	QCIF 176x144	24	170
4	CIF 352x288	24	290
5	CIF 352x288	24	400
6	CIF 352x288	24	560
7	CIF 352x288	24	780

A taxa constante (CBR) foi obtida com base no algoritmo proposto na seção 5. Esse algoritmo itera o processo de codificação SVC habilitando uma quantidade N de camadas em cada iteração. No caso em questão, N varia de 1 a 8. Para cada valor de N existe outra iteração para calcular o valor do parâmetro QP (quantização) até que a taxa (*bitrate*) especificada seja encontrada. Considerou-se margem de erro de 5% na taxa. Para validar a taxa resultante nessa segunda iteração, fez-se um escalonamento do *bytestream* utilizando os parâmetros "–l –t –f –keepf" do JSVM Extractor conforme a maior camada SVC presente na iteração. Com isso, é garantido que o vídeo terá uma das taxas especificadas na Tabela 19 na saída do módulo Adaptador SVC implementado na prova de conceito.

Os resultados da compressão em modo CBR das quatro sequências de vídeo estão mostrados nas figuras seguintes.

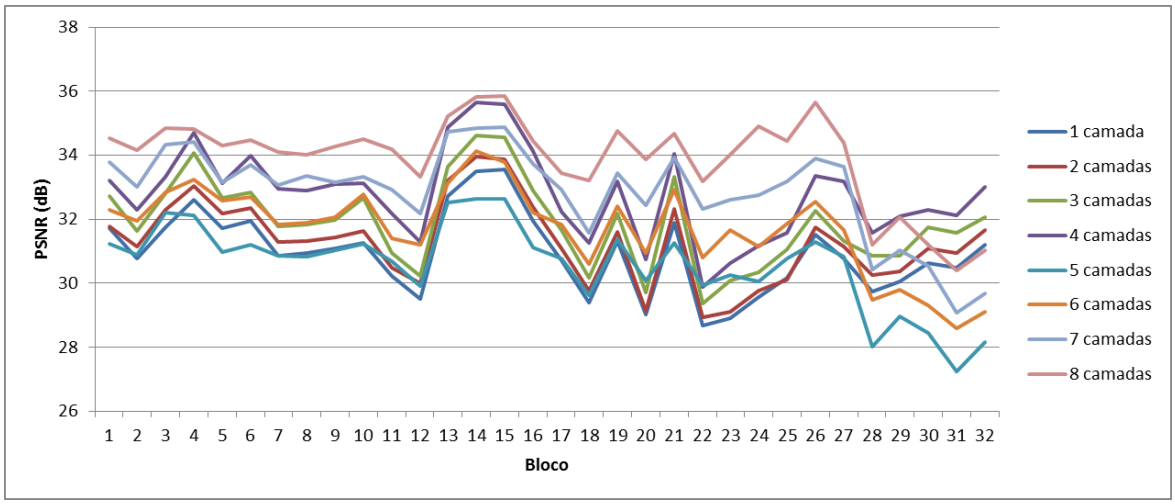


Figura 40 – *Foreman* codificado conforme Tabela 19. PSNR dos blocos.

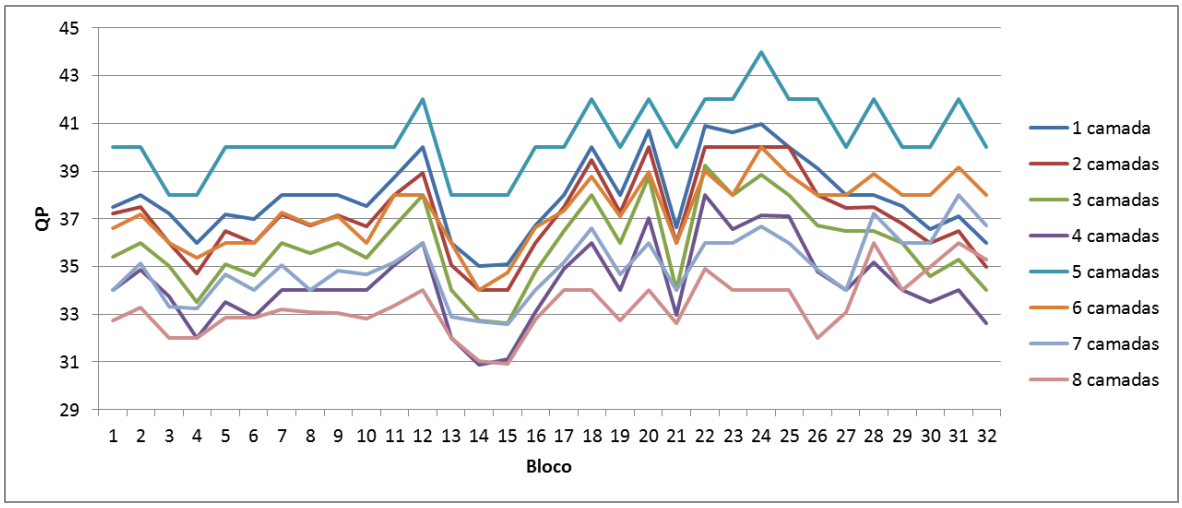


Figura 41 – *Foreman* codificado conforme Tabela 19. QP dos blocos.

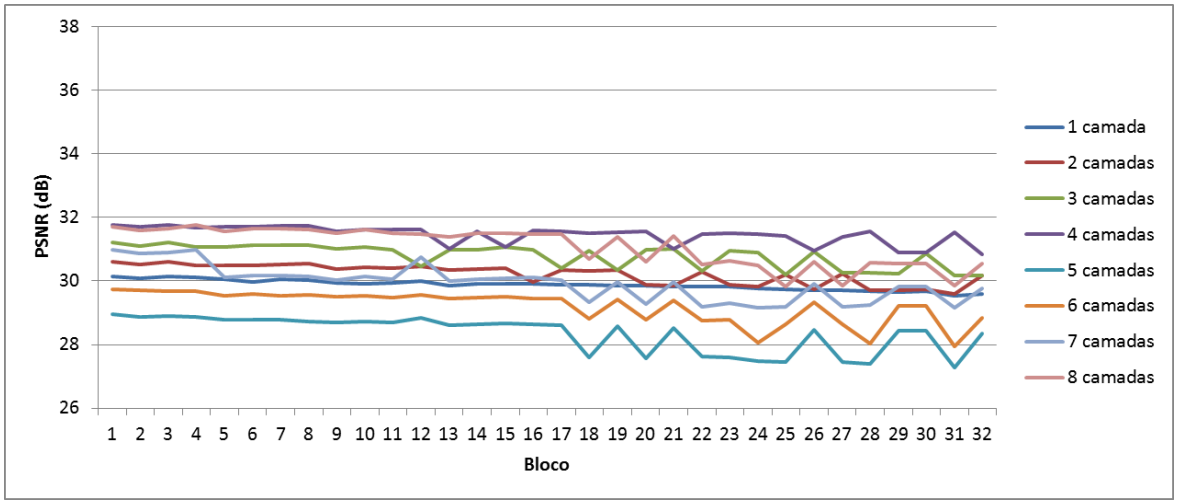


Figura 42 – *Waterfall* codificado conforme Tabela 19. PSNR dos blocos.

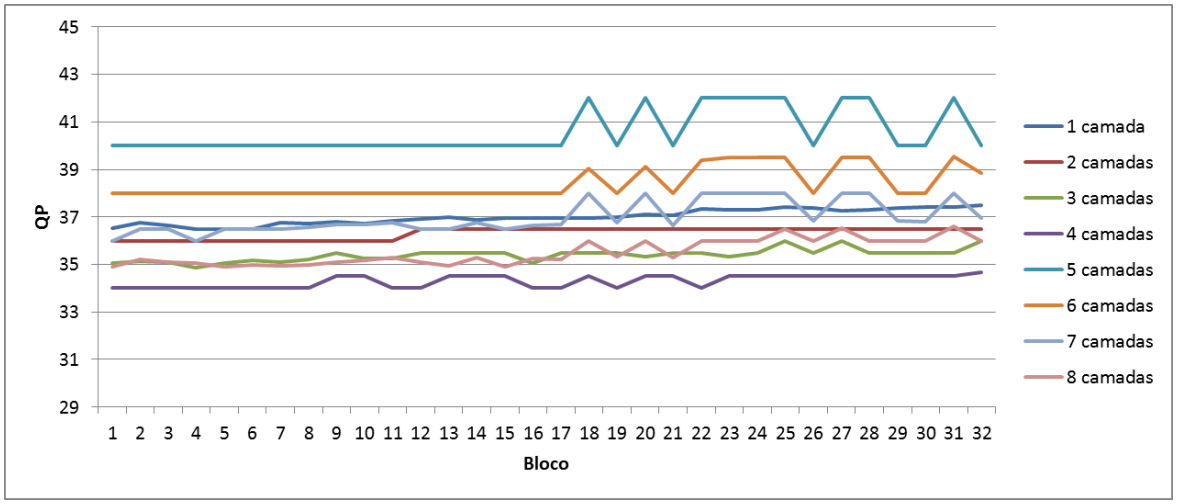


Figura 43 – *Waterfall* codificado conforme Tabela 19. QP dos blocos.

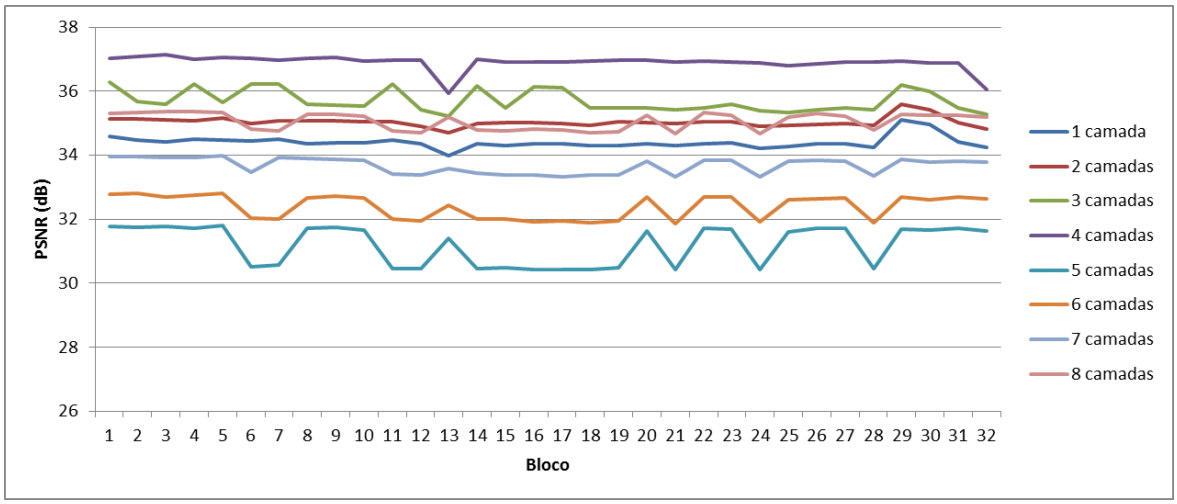


Figura 44 – *Container* codificado conforme Tabela 19. PSNR dos blocos.

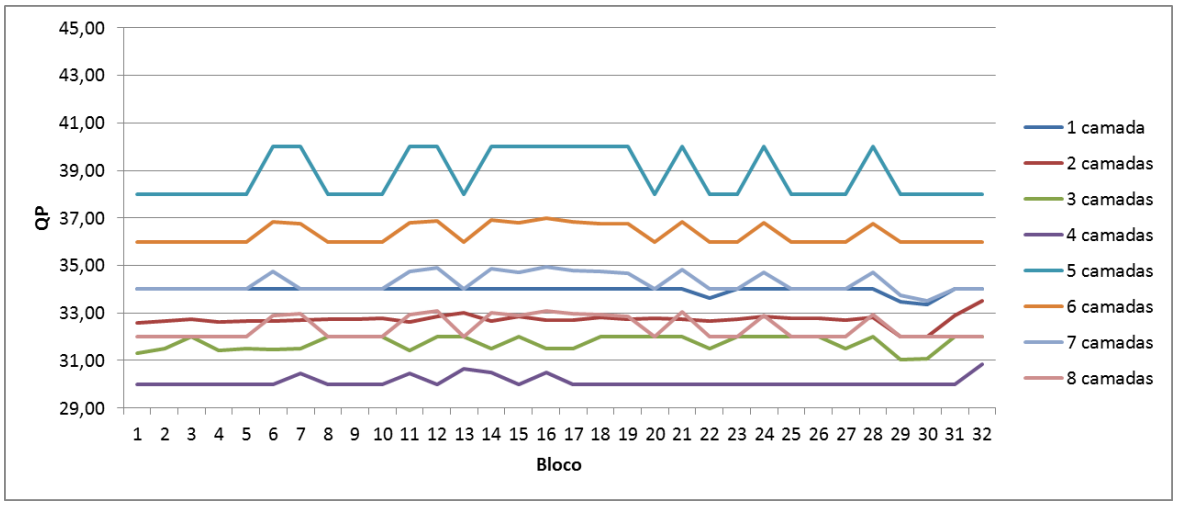


Figura 45 – *Container* codificado conforme Tabela 19. QP dos blocos.

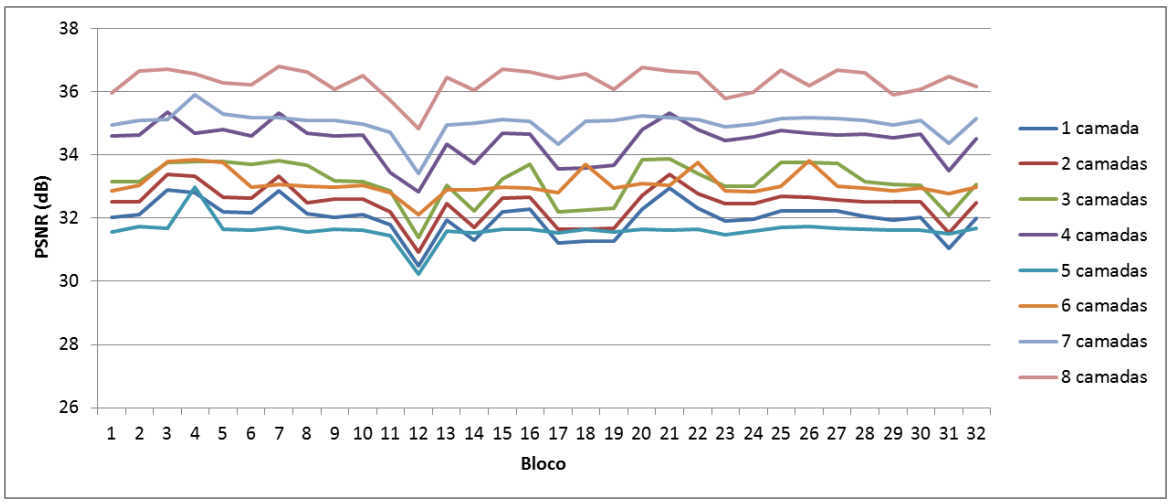


Figura 46 – *News* codificado conforme Tabela 19. PSNR dos blocos.

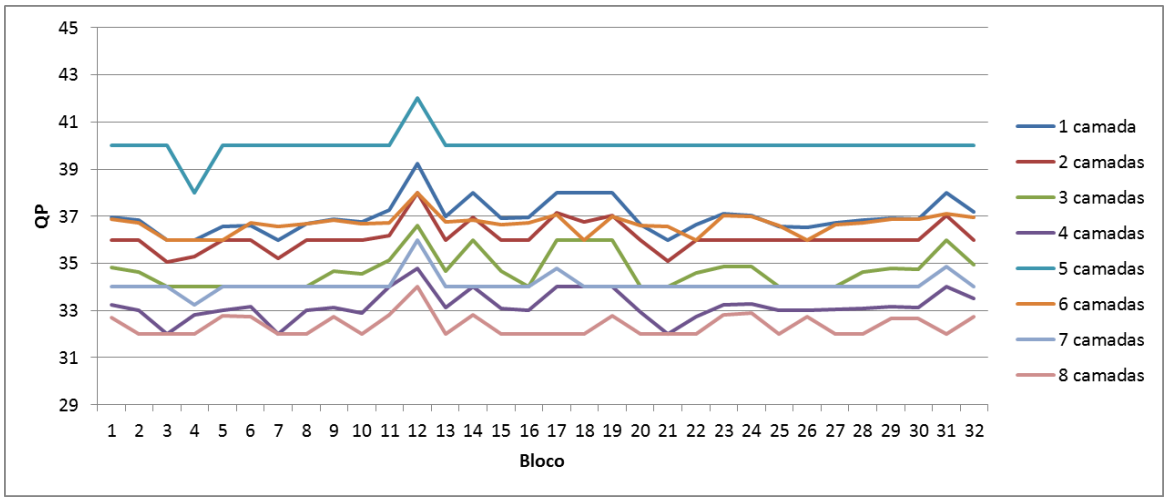


Figura 47 – *News* codificado conforme Tabela 19. QP dos blocos.

A Figura 48 mostra a PSNR média das 4 sequências em função do número de camadas. O aumento da PSNR não é estritamente crescente porque ela não é sensível à resolução espacial do vídeo, que muda justamente da camada 4 para 5 (de QCIF para CIF). As margens de erro dizem respeito ao intervalo de confiança de 95% relativo aos dados.

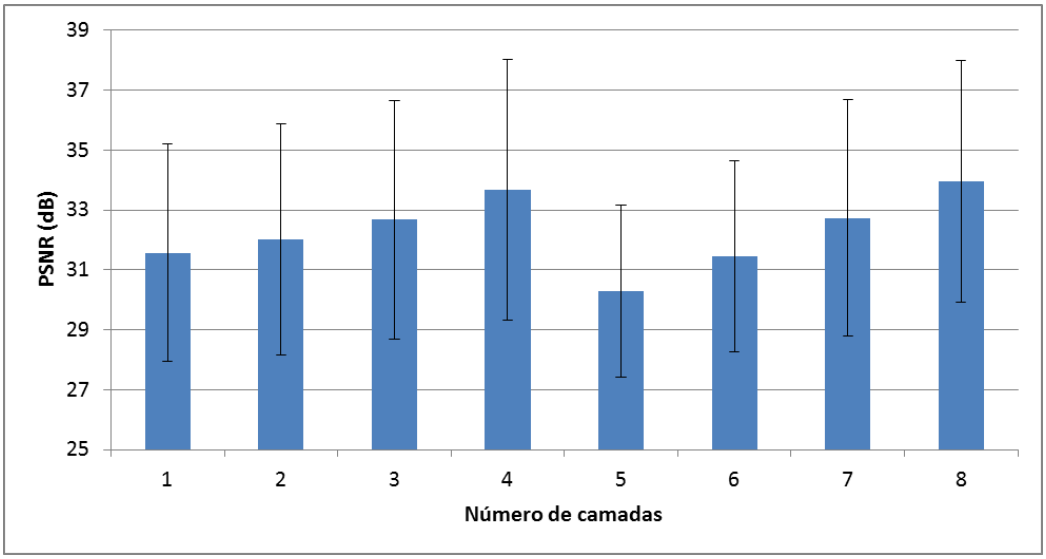


Figura 48 – PSNR média das sequências.

De maneira semelhante à figura acima, a Figura 49 sumariza a taxa média obtida em função do número de camadas. A pequena margem de erro deve-se à tolerância de 5% configurada no processo de compressão CBR.

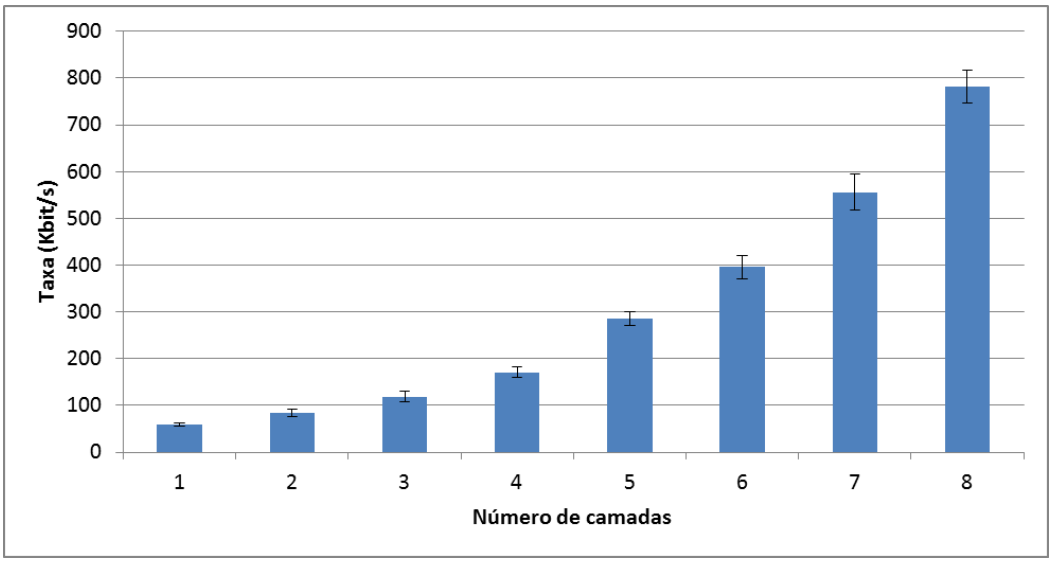


Figura 49 – Taxa média das sequências.

A Tabela 20 detalha as configurações utilizadas para gerar o *bytestream* SVC das sequências via JSVM Encoder.

Tabela 20 – Configurações do JSVM Encoder

VideoBase.cfg

FrameRate	24
MaxDelay	333.0
FramesToBeEncoded	8
NonRequiredEnable	0
CgsSnrRefinement	1
EncodeKeyPictures	1
MGSControl	2
GOPSize	4
IntraPeriod	8
NumberReferenceFrames	1
BaseLayerMode	1
ConstrainedIntraUps	0
SearchMode	4
SearchFuncFullPel	0
SearchFuncSubPel	0
SearchRange	32
ELSearchRange	0
FastBiSearch	0
BiPredIter	4
IterSearchRange	8
LoopFilterDisable	0
LoopFilterAlphaC0Offset	0
LoopFilterBetaOffset	0
InterLayerLoopFilterDisable	0
InterLayerLoopFilterAlphaC0Offset	0
InterLayerLoopFilterBetaOffset	0
NumLayers	8
LayerCfg	Layer0.cfg
LayerCfg	Layer1.cfg
LayerCfg	Layer2.cfg
LayerCfg	Layer3.cfg
LayerCfg	Layer4.cfg
LayerCfg	Layer5.cfg
LayerCfg	Layer6.cfg
LayerCfg	Layer7.cfg
WeightedPrediction	0
WeightedBiprediction	0
LARDO	0
MultiLayerLambdaSel	0

```
PreAndSuffixUnitEnable            1
NestingSEI                        0
SceneInfo                         0
TLNestingFlag                     0
IntegrityCheckSEI                 0
TL0DepRepIdxSeiEnable             0
RPEncCheck                        0
MVDiffThreshold                   20
EnableVclHRD                      0
EnableNalHRD                      0
```

Layer0.cfg

```
SourceWidth             176
SourceHeight            144
FrameRateIn             24
FrameRateOut            24
SymbolMode              0
IDRPeriod               8
IntraPeriod             8
MbAff                   0
PAff                    0
BottomFieldFirst        0
LowComplexityMbMode     0
ProfileIdc              0
MinLevelIdc             10
UseLongTerm             0
MMCOEnable              1
MMCOBaseEnable          1
Enable8x8Transform      1
ConstrainedIntraPred    0
CbQPIndexOffset         0
CrQPIndexOffset         0
ScalingMatricesPresent  0
IPCMRate                0
BiPredLT8x8Disable      0
MCBlocksLT8x8Disable    0
DisableBSlices          0
MaxDeltaQP              6
QP                      32
ForceReOrdering         0
MeQPLP                  32.00
MeQP0                   32.00
MeQP1                   32.00
InterLayerPred          0
ILModePred              0
ILMotionPred            0
ILResidualPred          0
SliceSkip               0
MGSVectorMode           0
MGSVector0..15          0
ExplicitQPCascading     1
DQP4TLevel0             -2
DQP4TLevel1             1
DQP4TLevel2             2
DQP4TLevel3             3
DQP4TLevel4             4
DQP4TLevel5             5
DQP4TLevel6             6
NumSlicGrpMns1          0
SlcGrpMapType           1
SlcGrpChgDrFlag         0
SlcGrpChgRtMns1         85
UseRedundantSlc         0
PLR                     0
SliceMode               0
SliceArgument           0
AvcRewriteFlag          0
AvcAdaptiveRewriteFlag  0
```

Layer1.cfg (apenas linhas diferentes com respeito a Layer0.cfg)

```
InterLayerPred          1
ILModePred              1
ILMotionPred            1
```

Layer2.cfg (idem a Layer1.cfg)

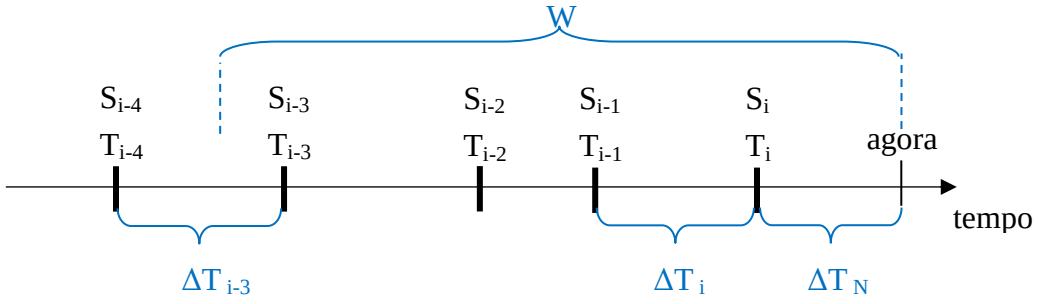
```
Layer3.cfg (idem a Layer1.cfg)

Layer4.cfg (apenas linhas diferentes com respeito a Layer0.cfg)

SourceWidth             352
SourceHeight            288

Layer5.cfg (apenas linhas diferentes com respeito a Layer0.cfg)

SourceWidth             352
SourceHeight            288
InterLayerPred          1
ILModePred              1
ILMotionPred            1

Layer6.cfg (idem a Layer5.cfg)

Layer7.cfg (idem a Layer5.cfg)
```

Medição de taxa

Descreve-se aqui a metodologia utilizada para medição da taxa recebida ou enviada em determinado nó Scribe nos testes supracitados.

A taxa é medida com base nos pacotes ocorridos (recebidos ou enviados) na última janela de tempo *W*. Para cada pacote registra-se o evento E_i representado pela tupla de informações (T_i, S_i), onde T_i é o momento, em milissegundos, de ocorrência do pacote e S_i é seu tamanho, em *bytes*. O evento anterior a E_i é o E_{i-1}. O valor considerado para *W* foi de 1s com base em análise empírica mostrada à diante. A Figura 50 exemplifica uma sequência de eventos *E*.

Figura 50 – Registro de eventos de chegada de pacotes para medição de taxa



Sejam: $\Delta T_i = T_i - T_{i-1}$; ΔT_N o tempo decorrido desde o último evento (E_i); e E_j o evento mais antigo dentro da janela *W* para o qual existe E_{j-1}. A taxa recebida T_R é aproximada pela função que segue:

$$T_R = \frac{\sum_{k=i}^{j} S_k}{\sum_{k=i}^{j} T_k + \Delta T_N} \qquad (1)$$

A taxa é calculada sempre que se deseja saber seu valor, de forma a valer o efeito do termo ΔT_N. Sua função é decrescer, suavemente, o valor da taxa quando não há novos eventos de chegada de pacote. Considera-se o valor mínimo para ΔT_i como sendo metade da janela W. Isso evita picos na medição quando da ocorrência de novo evento e não há eventos E dentro da janela.

A Tabela 21 sugere um algoritmo que implementa a lógica supra descrita. A operação "RegistrarPacote" é executada sempre que um pacote chega no receptor, sendo responsável por registrar o evento E_i. A operação "ObterTaxa" faz o cálculo a taxa recebida com base no histórico de eventos.

Tabela 21 – Algoritmo usado para medir a taxa

```
Define NUM_EVENTOS = 500 // valor inicial da memória de eventos

Define W = tamanho da janela

Cria lista_T de tamanho NUM_EVENTOS para armazenar os valores de Ti-k.
O primeiro elemento da lista é referenciado por lista_T[0]

Cria lista_S de tamanho NUM_EVENTOS para armazenar os valores de Si-k.
O primeiro elemento da lista é referenciado por lista_S[0]

Define indice = 0

RegistrarPacote (pacote)
{
   Se timestamp do pacote = lista_T[indice]
   {
      Atribui lista_S[indice] = lista_S[indice] + tamanho do pacote
   }
   Caso contrário
   {
      Atribui indice = (indice + 1) módulo NUM_EVENTOS
      lista_T[indice] = timestamp do pacote
      lista_S[indice] = tamanho do pacote
   }
}

ObterTaxa ()
{
   Seja agora o timestamp corrente do receptor

   Atribui total_T = 0
   Atribui total_S = 0
   Atribui i = 0
   Atribui i_menos1 = 0

   Repete fazendo n variar de 0 a (NUM_EVENTOS - 1) em incrementos de 1
   {
      // ajuste para fila circular

      Atribui i = (indice_i - n + NUM_EVENTOS) módulo NUM_EVENTOS
      Atribui i_menos1 = (indice_i - n - 1 + NUM_EVENTOS) módulo NUM_EVENTOS

      Se lista_T[i] = 0 ou lista_T[i_menos1] = 0
      {
         Para a repetição, pois não há mais dados em memória
      }

      Se lista_T[i] < (agora - W)
      {
         Para a repetição, pois o evento é anterior à janela
      }

      Atribui deltaT = lista_T[i] - lista_T[i_menos1]
```

```
      Se deltaT < 0
      {
         Para a repetição, pois está havendo sobreposição de dados

         Opcionalmente gerar erro ou aumentar o tamanho da memória (NUM_EVENTOS)
      }

      Se n = 0
      {
         // reduz gradativamente a taxa caso não há eventos faz um tempo
         deltaT = agora - lista_T[i_menos1]
      }

      Atribui total_T = total_T + deltaT
      Atribui total_S = total_S + lista_S[i]

   }

   Se total_T < W/2
   {
      // evita picos quando há novo evento e a memória está vazia
      total_T = W/2
   }

   Se total_T > 0
   {
      Retorna total_S / total_T * 8 * 1000;  // conversao para bit/s
   }
   Caso contrário
   {
      Retorna 0
   }
}
```

Para dimensionamento da largura da janela foram realizados testes com diferentes valores de *W* e tamanho de pacote valendo 3 KB. A Figura 51 mostra a taxa de transmissão da sequência de vídeo e a Figura 52, Figura 53 e Figura 54 mostram, respectivamente, as taxas medidas com janela de 0,5s, 1s e 2s. Para os 3 casos mostra-se a taxa medida com período de amostragem de 0,25s.

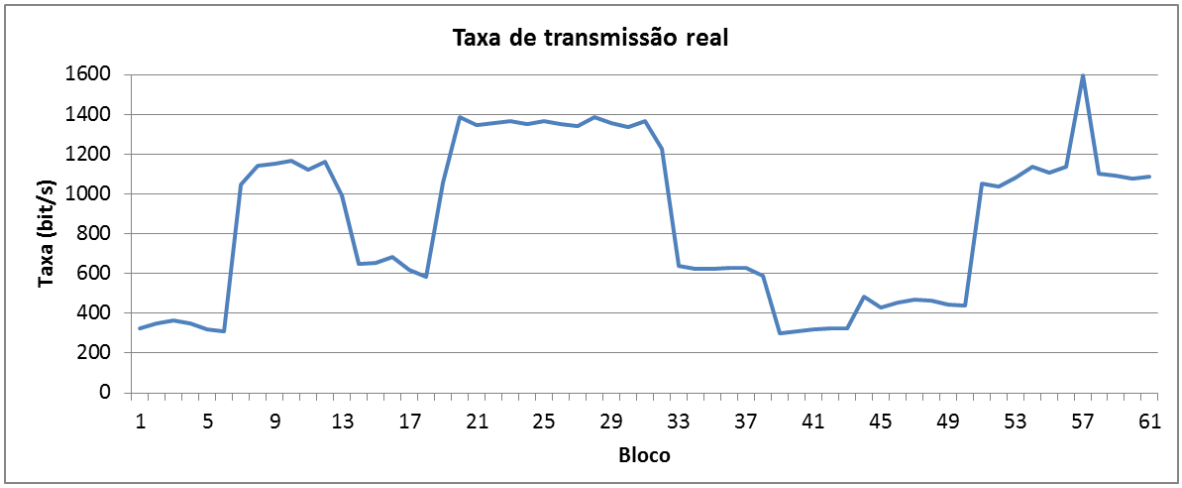


Figura 51 – Taxa de transmissão real

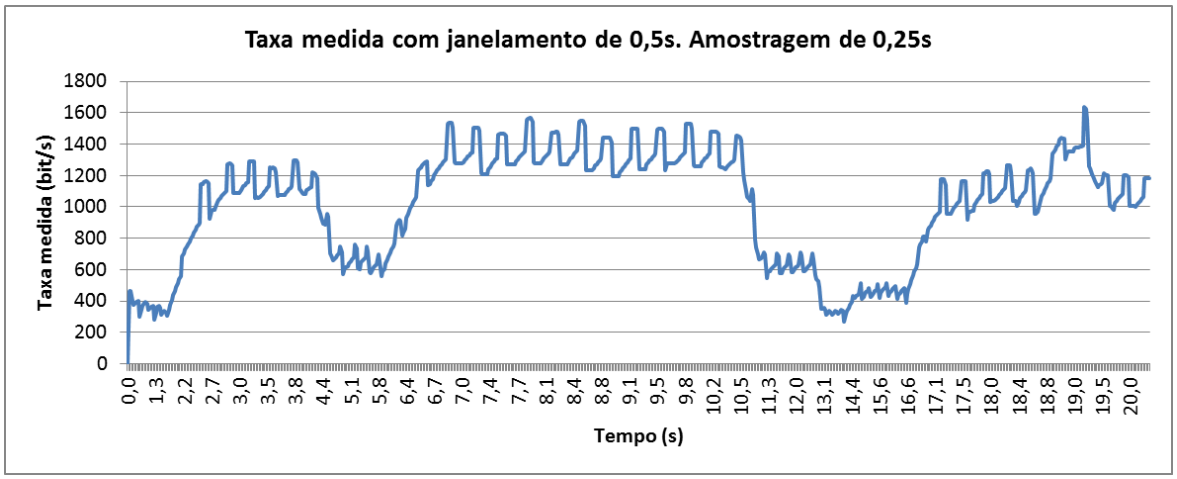


Figura 52 – Taxa medida com janelamento de 0,5s

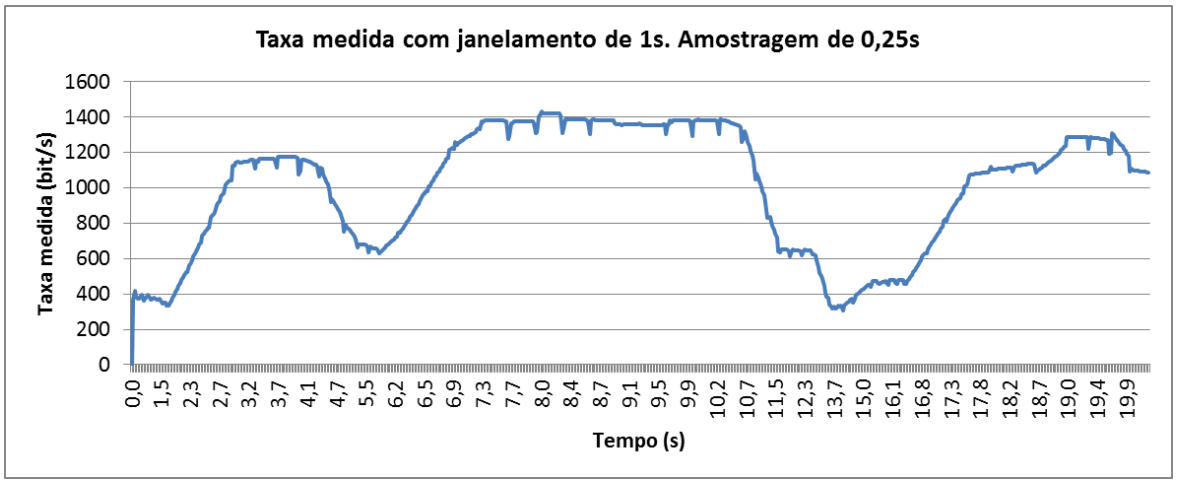


Figura 53 – Taxa medida com janelamento de 1s

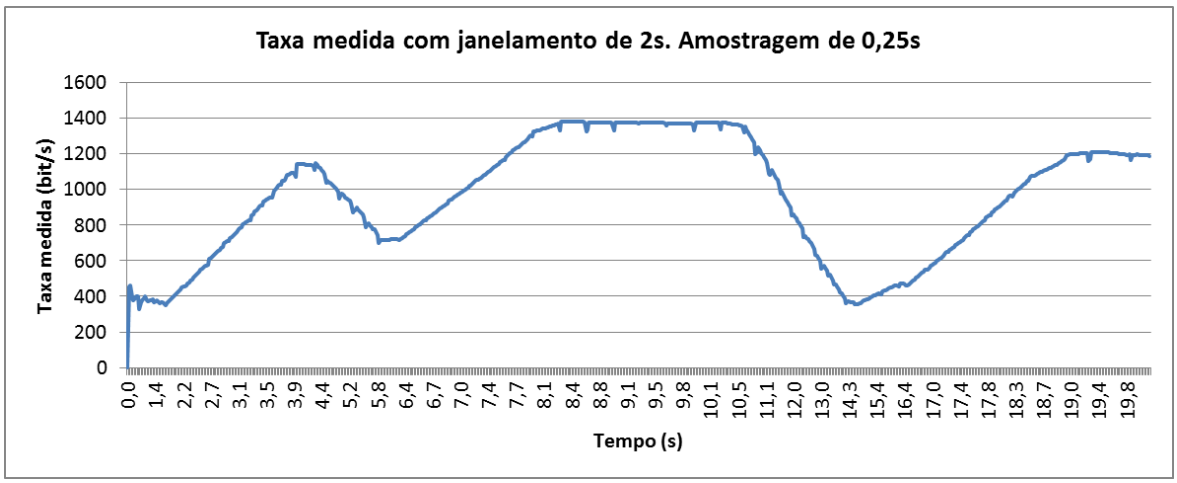


Figura 54 – Taxa medida com janelamento de 2s

7.2 - CENÁRIO 1

Descrição

Inicialmente cria-se uma rede Pastry formada por dois nós: um sendo a fonte de vídeo e o outro a raiz de uma rede Scribe.

A cada 10 segundos (tempo de 30 blocos) um novo nó ingressa na árvore Scribe como filho do nó raiz. O primeiro filho entra 10 segundos após inicio da transmissão do vídeo. O teste tem duração de 120 segundos.

O teste é composto de 12 execuções, sendo 3 para cada uma das quatro sequências de vídeo.

A largura de banda do nó raiz é limitada a 2 Mbit/s através de um esquema de limitação de banda, descrito a seguir.

Limitação de banda

A limitação de banda aqui considerada constitui um recurso utilizado para gerar perda de pacotes no nó raiz.

A limitação de banda é realizada através de um esquema de contenção de canal: para cada pacote enviado, calcula-se o tempo de transmissão conforme a largura de banda definida. O canal é considerado ocupado durante esse tempo. Novas solicitações de envio de pacote são colocadas em uma fila FIFO (*First-In First-Out*). Utilizou-se uma fila de 25 posições para os testes deste trabalho. Quando a fila está cheia, as novas solicitações são imediatamente descartadas.

Implementou-se uma mudança na política de *push* do Scribe para adequação ao esquema de contenção de canal: O Scribe, nativamente, faz o *push* para os nós filhos seguindo sempre uma mesma ordem (filho 1, filho 2, etc). Com isso, devido ao esquema de contenção, se a fila FIFO tem tamanho S_Q e o nó possui uma quantidade S_N de filhos, e $S_Q < S_N$, então, aproximadamente, somente os primeiros S_Q filhos recebem pacotes. Para solução deste problema, implementou-se aleatoriedade na ordem do *push* do Scribe.

Resultado

A Figura 55 mostra a qualidade média do vídeo nos receptores (nós filhos). Observa-se que a partir do tempo de 30s, quando é ingressado o terceiro nó filho, ocorre redução substancial da qualidade do vídeo, vez que o banda necessária para suportar os 3 nós seria de pelo menos 780 * 3 ≈ 2,5 Mbit/s. Sem a capacidade de adaptação a rede degrada rapidamente, de tal forma que o vídeo não é mais reproduzido a partir do bloco 140 (47 segundos).

Para mensuração da qualidade do vídeo utilizou-se a métrica objetiva de referência completa PSNR (*Peak Signal-to-Noise Ratio*).

A região hachurada em cinza compreende a variabilidade dos dados em torno da média para cada bloco. Considerou-se intervalo de confiança de 95% valendo $\pm\, 2{,}201 \cdot dp$ sobre a média, onde dp é o desvio padrão amostral e o fator 2,201 deve-se ao ajuste dado pela distribuição *t* de Student.

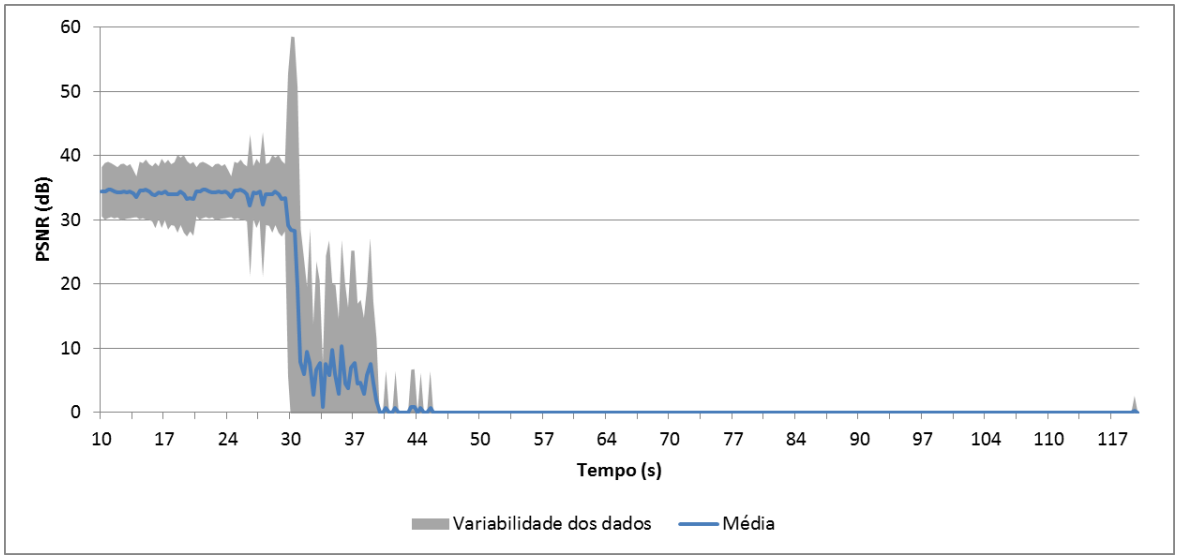


Figura 55 – PSNR média dos receptores (cenário 1)

7.3 - CENÁRIO 2

Descrição

Neste cenário, que é baseado no cenário 1, tem-se habilitados os módulos Adaptador SVC e TFMCC. Com isso, pretende-se analisar o ganho que desempenho da rede no que tange sua capacidade de adaptação quando há sobrecarga de *upload* em determinado nó.

Resultado

A adaptação provida com auxílio do TFMCC resultou em melhoramento substancial do sistema, conforme se ver na Figura 56. Apesar da grande variabilidade dos dados, a rede mantém-se funcionando mesmo com o crescimento da quantidade de nós filhos e com a sobrecarga na raiz, cuja banda de *upload* é limitada a 2 Mbit/s. A variabilidade é decorrente do fato de que, conforme como foi implementado, uma perda de pacote (fragmento) provoca a perda total do bloco, o que é mensurado como PSNR zero.
A região hachurada em cinza compreende o intervalo de confiança de 95%, já considerando o ajuste dado pela distribuição *t* de Student.

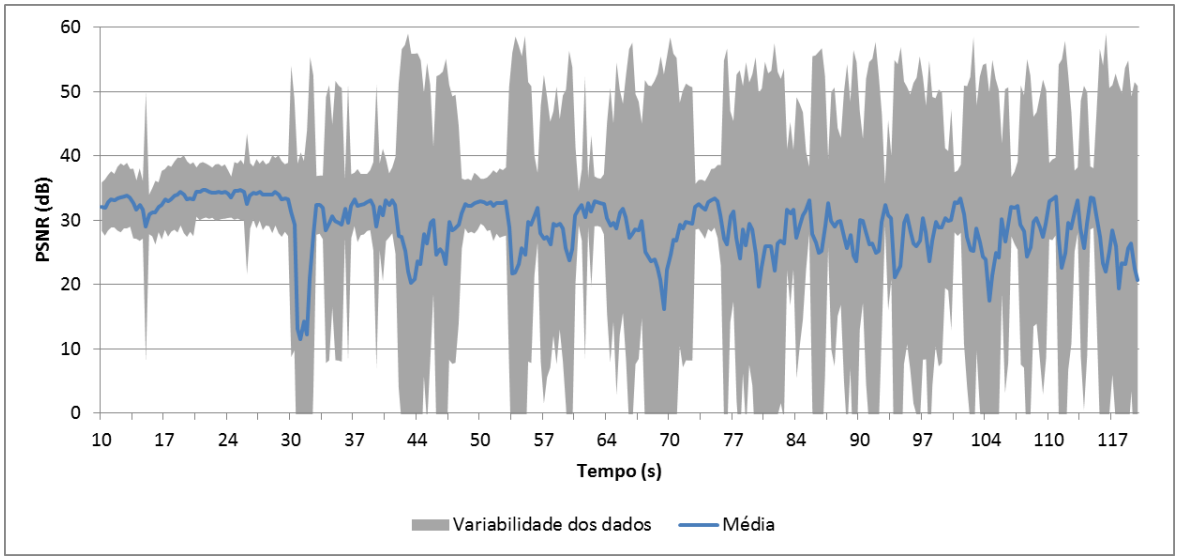


Figura 56 – PSNR média dos receptores (cenário 2)

A Figura 57 mostra a taxa média recebida pelos nós filhos ao longo da execução do vídeo. Foi realizada amostragem da taxa recebida em uma frequência de 2 Hz. O gráfico apresenta a média dos valores coletados em cada período.

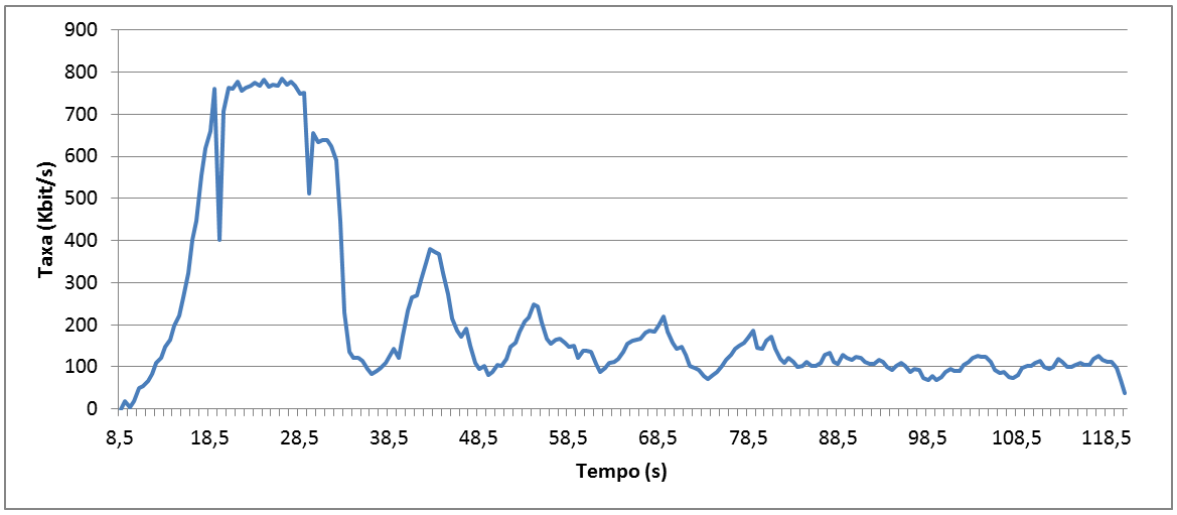


Figura 57 – Taxa média recebida, amostragem a cada 500ms (cenário 2)

De acordo com o mostrado na Figura 48, não se pode inferir uma relação direta entre qualidade do vídeo no sentido amplo – das 3 dimensões do SVC – e PSNR. Um incremento na dimensão espacial pode, por exemplo, reduzir a PSNR sem que esteja ocorrendo, de fato, uma redução da Qualidade de Experiência. Uma maneira de mensurar a qualidade geral do vídeo é calculando-se o número médio de camadas recebidas nos nós filhos, que é a Figura 58.

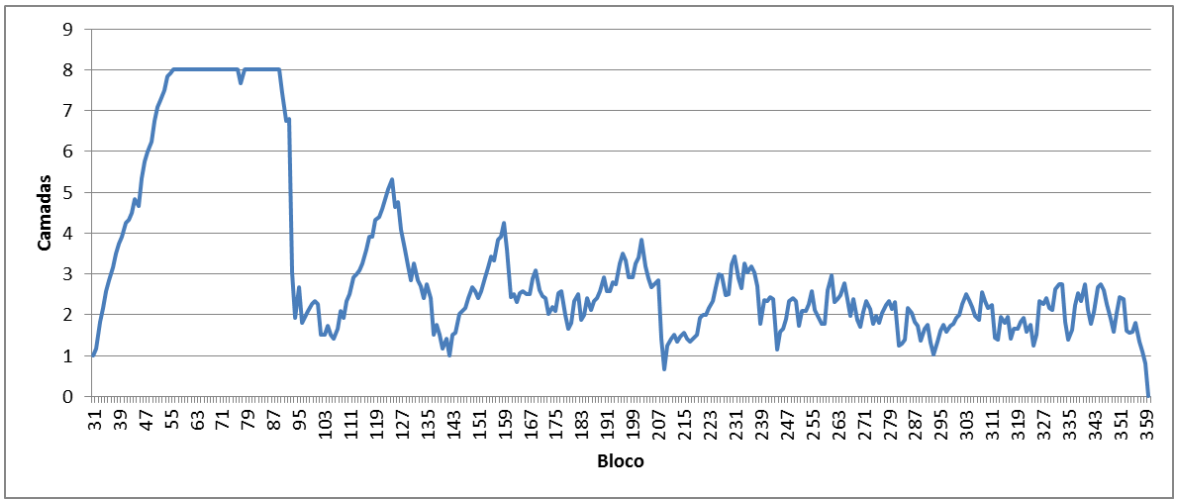


Figura 58 – Número médio de camadas nos receptores (cenário 2)

7.4 - CENÁRIO 3

Descrição

Este cenário é baseado no cenário 2 porém com a ativação do esquema de retransmissão de bloco, proposto na seção 5.10, que objetiva reduzir os efeitos causados pela perda de pacotes.

Resultado

A Figura 59 mostra o resultado da qualidade média dos receptores, em termos de PSNR, com uso do esquema de retransmissão em camada base SVC. A região hachurada em cinza compreende o intervalo de confiança dos dados de 95%, já considerando o ajuste dado pela distribuição *t* de Student.

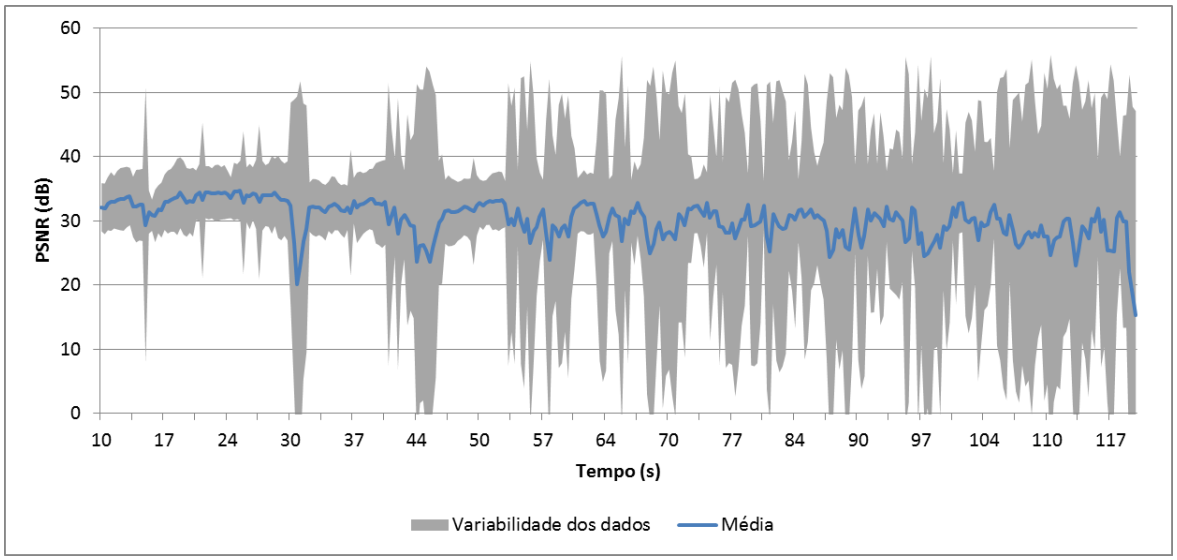


Figura 59 – PSNR média dos receptores (cenário 3)

Mostra-se, na Figura 60, a média geral da PSNR para ambos os cenários 3 e 2 – com e sem retransmissão. Entende-se por "média geral" a média de todos os dados amostrados durante o intervalo de tempo da execução. As margens de erro dizem respeito à variabilidade da média das amostras, ou seja, a incerteza da média amostral com respeito à média populacional. Considerou-se intervalo de confiança de 95% valendo $\pm 2{,}201 \cdot dp/\sqrt{12}$ sobre a média, onde $dp/\sqrt{12}$ é o erro padrão (EP) para o espaço amostral de tamanho 12 considerado no teste. O fator 2,201 deve-se ao ajuste dado pela distribuição *t* de Student.

Com base na métrica PSNR, verifica-se que a retransmissão resultou em melhoria da média de 29,1 dB para 30,2 dB. Dado que os valores de PSNR das sequências consideradas variam de 27,4 dB a 37,1 dB, então a melhoria obtida representa 11,5% desse intervalo.

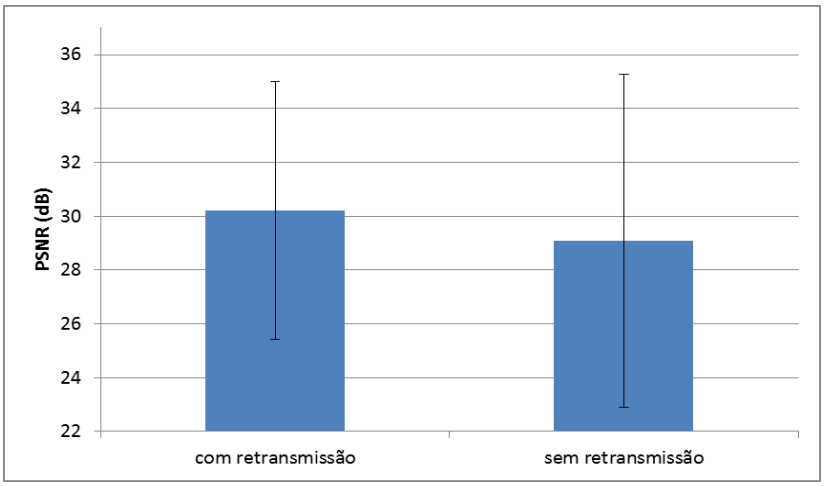


Figura 60 – PSNR média (com e sem retransmissão)

A taxa média recebida pelos nós filhos bem como o número médio de camadas SVC estão mostrados na Figura 61 e Figura 62, respectivamente.

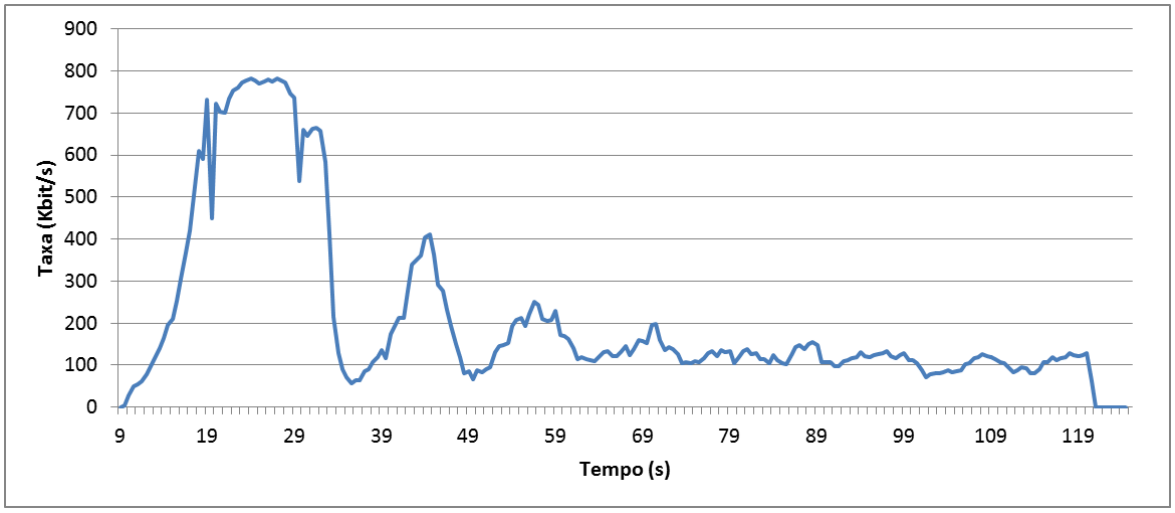


Figura 61 – Taxa média recebida, amostragem a cada 500ms (cenário 3)

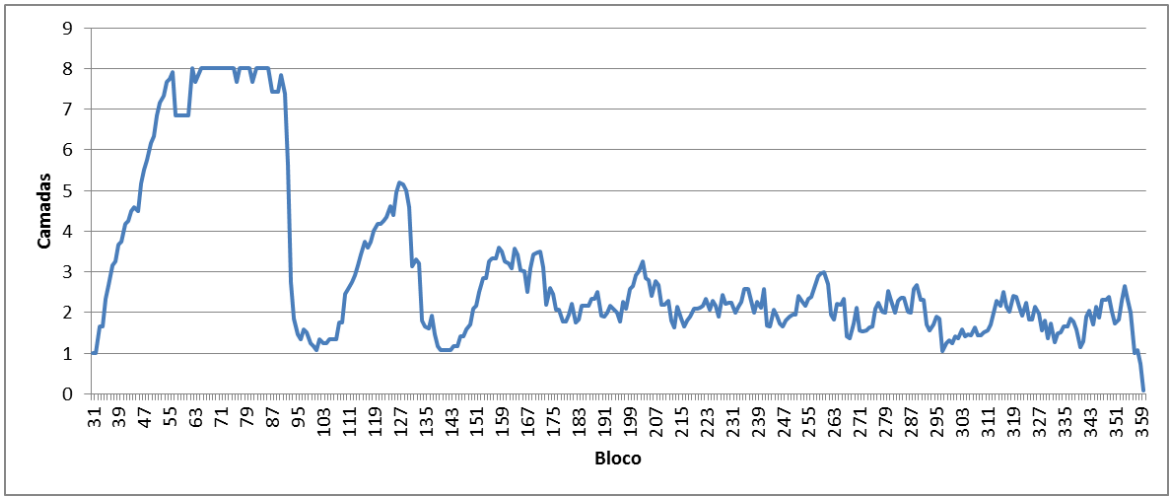


Figura 62 – Número médio de camadas nos receptores (cenário 3)

Para melhor comparar o presente resultado com o cenário 2, seja agora uma métrica fortemente relacionada com a retransmissão de pacote: percentual de perda de bloco. Sabe-se, na configuração utilizada para os testes, que cada bloco perdido causa ausência de vídeo por 333ms para o usuário espectador, fato esse que compromete a Qualidade de Experiência. Parte dos blocos perdidos, devido à perda de um ou mais fragmento, é agora recebida em camada base graças à retransmissão. Para medir a eficácia desse procedimento, calcula-se o percentual de perda de bloco em ambos os cenários 2 e 3. O resultado, amostrado a cada segundo, é apresentado na Figura 63 e Figura 64, respectivamente. O resultado consolidado está na Figura 65. As margens de erro dizem respeito à variabilidade da média das amostras com intervalo de confiança de 95% e considerando ajuste da distribuição *t* de Student.

No cenário 3 tem-se ainda que: dos blocos recebidos nos nós filhos, 8,6 % foram por retransmissão.

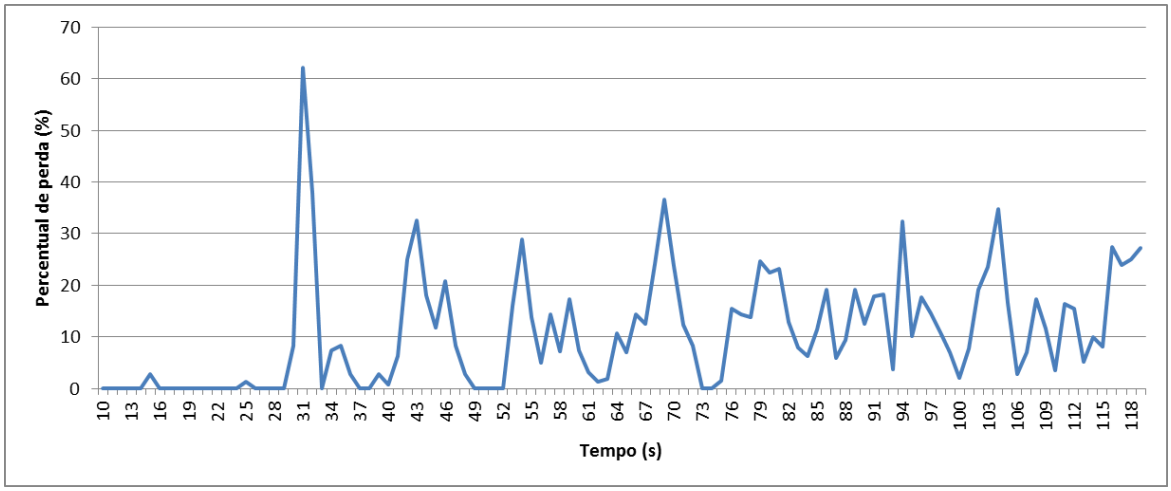


Figura 63 – Percentual de perda de bloco (cenário 2)

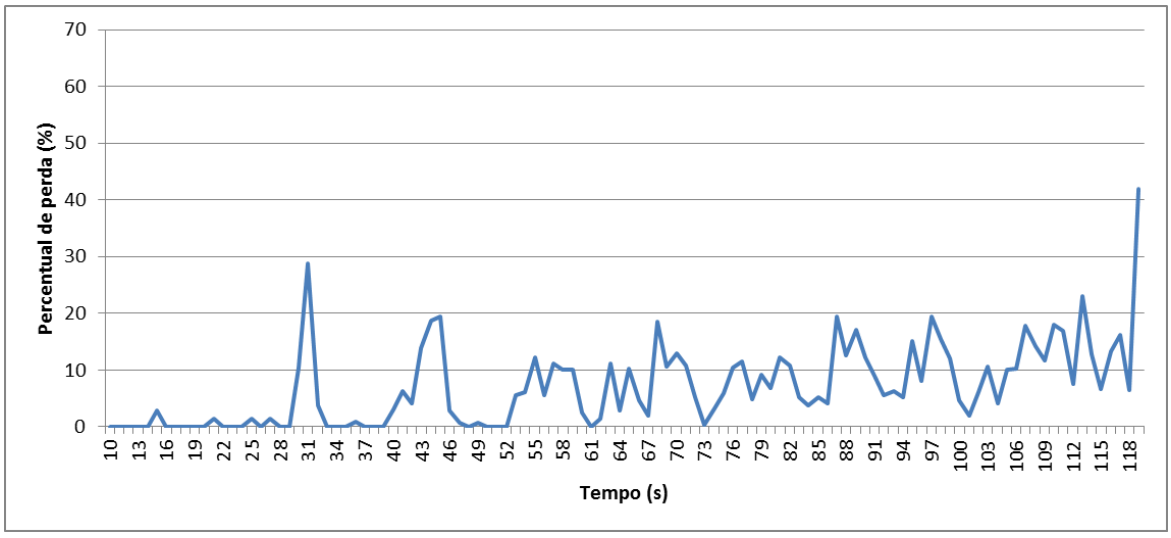


Figura 64 – Percentual de perda de bloco (cenário 3)

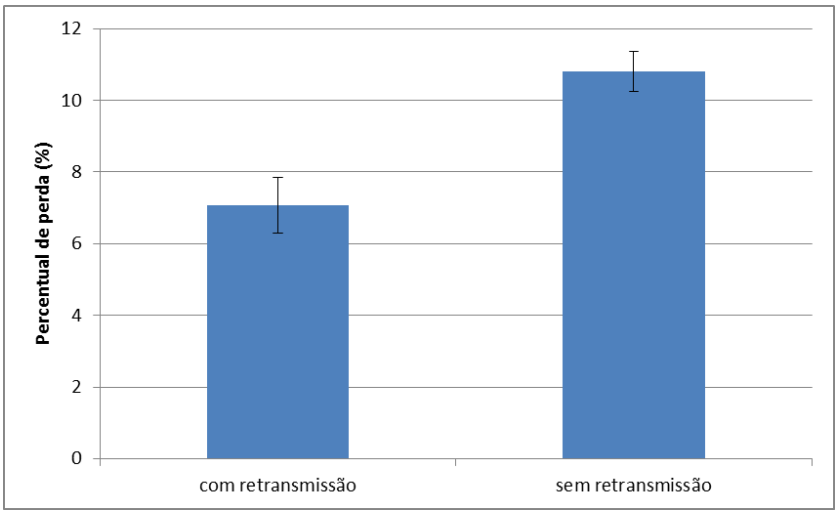


Figura 65 – Percentual total de perda no cenário 3 e no 2

7.5 - CENÁRIO 4

Descrição

Os cenários 1, 2 e 3 verificaram o gargalo na raiz da árvore devido à sobrecarga de receptores bem como o efeito da ativação do módulo adaptativo e do esquema de retransmissão. Neste cenário 4 testa-se a sobrecarga de enlace entre o nó raiz e um receptor com o intuito de verificar o comportamento da adaptação dos blocos SVC em função do TFMCC.

A rede Pastry é criada com 3 nós e assim mantida durante a execução do teste. O primeiro nó é a fonte de vídeo e os outros 2 foram uma rede Scribe de uma raiz e 1 receptor.

A execução tem duração total de 260 segundos.

O enlace na rede Scribe é caracterizado pelas condições descritas na Tabela 22, onde *t* é o tempo decorrido desde o início da execução do teste.

Tabela 22 – Condições de enlace na rede Scribe no cenário 4

Período		**Condição do enlace**
P0	0 <= t < 10s	Sem perda
P1	10s <= t < 40s	1 perda a cada T = 2s
P2	40s <= t < 50s	Sem perda
P3	50s <= t < 80s	1 perda a cada T = 1s
P4	80s <= t < 90s	Sem perda

P5	90s <= t < 120s	1 perda a cada T = 0,5s
P6	120s <= t < 130s	Sem perda
P7	130s <= t < 200s	Perda variando uniformemente de 0% a 10%. Condição semelhante à seção 6.3 de [52]
P8	200s <= t	Mantém-se perda de 10% acima

Resultado

A Figura 66 – Taxa recebida pelo nó filho (cenário 4) – evidencia o comportamento do TFMCC frente à perda de pacotes. Os períodos P1, P3 e P5 referem-se às perdas de 2 pacotes/s, 1 e 0,5, respectivamente, conforme definido na Tabela 22.

Observa-se que o TFMCC converge rapidamente para uma determinada taxa de transmissão conforme o valor da perda de pacotes. A taxa mantém-se estável enquanto o padrão de perda for o mesmo.

A perda de 2 pacotes/s foi interpretada como alta pelo TFMCC e a taxa foi reduzida para seu valor mínimo, de aproximadamente 60 Kbit/s, que corresponde ao vídeo contendo apenas a camada base (L0). O patamar do período P1 mostra esse resultado.

Para uma perda de 1 pacote/s, período P3, o TFMCC se estabilizou em aproximadamente 300 Kbit/s, mantendo o vídeo com 5 camadas SVC.

A perda de 0,5 pacote/s não provocou redução de taxa pelo TMFCC, conforme se observa no período P5.

Entre os blocos 400 e 600, período P7, tem-se a perda gradativa que vai de 0 a 10%. A partir de então ela é mantida em 10%. O TFMCC tenta periodicamente subir a taxa. Entretanto, vez que a perda do canal é percentual, se a taxa sobe então a perda efetiva é maior, fazendo o TFMCC reduzir novamente a taxa. O resultado é o gráfico dente de serra da Figura 66.

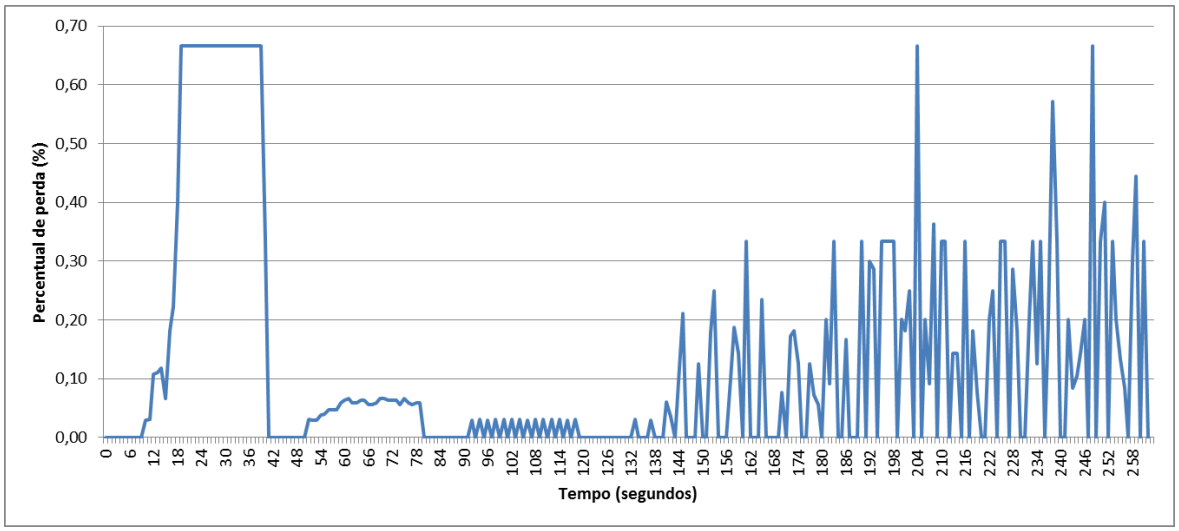


Figura 66 – Taxa recebida pelo nó filho (cenário 4)

A Figura 68 mostra o número de camadas SVC recebidas pelo nó filho Scribe. Nesse gráfico fica evidente o início do período P7 (bloco 400). Ficam visíveis também os intervalos de recuperação, que são P0, P2, P4, e P6. O TFMCC atinge a taxa máxima em aproximadamente 5 segundos (agressividade) e demora aproximadamente 3 segundos para reduzir a taxa quando da detecção da primeira perda após período sem perdas (responsividade).

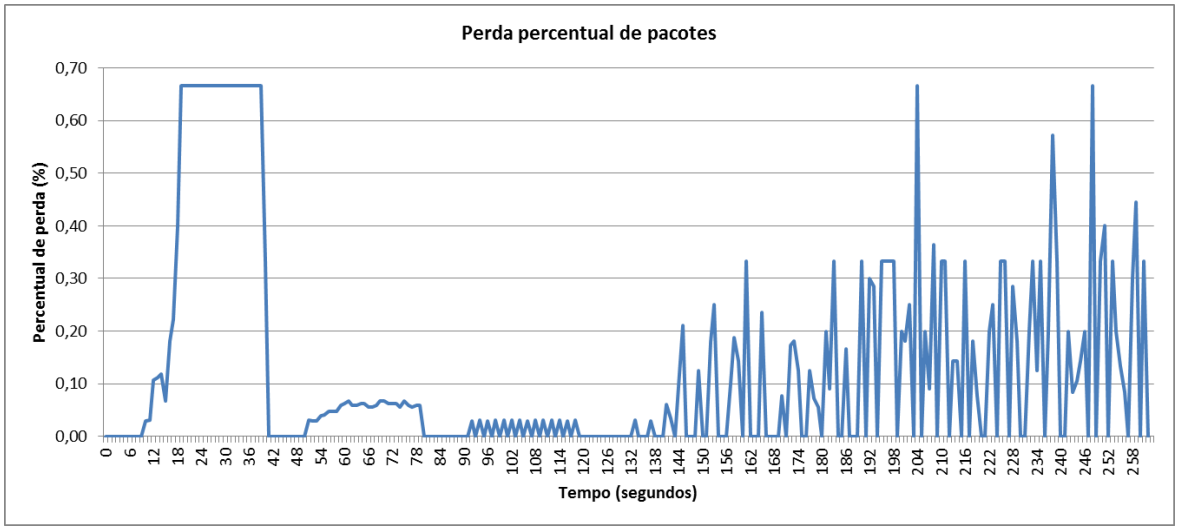


Figura 67 – Perda de pacotes no enlace Scribe (cenário 4)

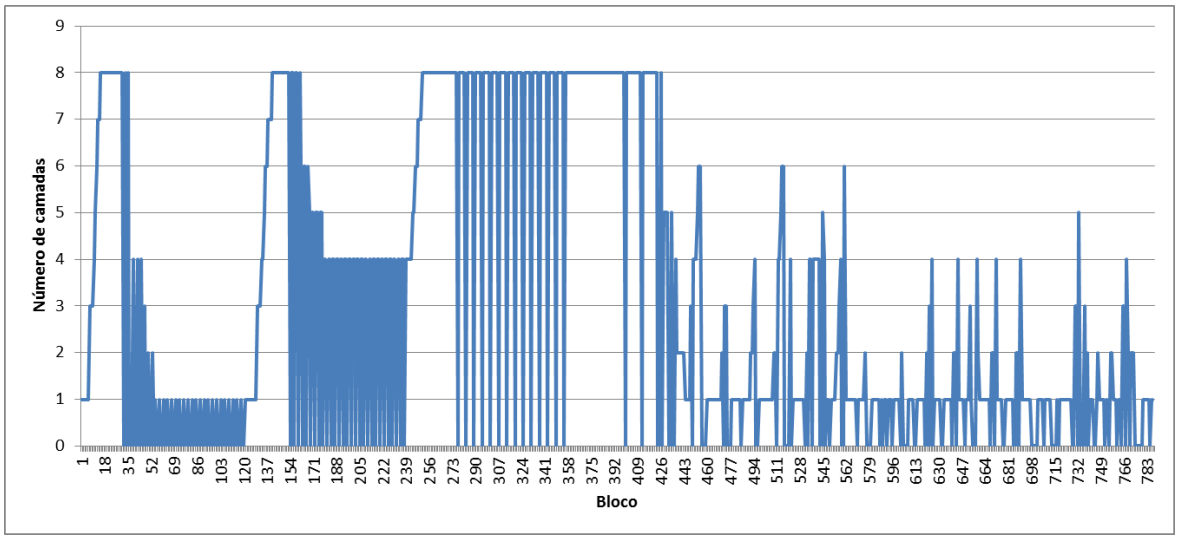


Figura 68 – Número de camadas SVC recebidas pelo nó filho (cenário 4)

A Figura 69 mostra a PSNR medida no receptor. A PSNR é considerada zero quando o bloco é descartado no receptor ou quando não é recebido. Vale lembrar que o descarte ocorre no Remontador de Blocos quando um ou mais fragmentos estão faltando e o compartimento de remontagem é expirado para dar lugar a novos blocos. Cada bloco perdido culmina em um congelamento temporário do vídeo na unidade de reprodução em tela, reduzindo, portanto, a qualidade de experiência do usuário.

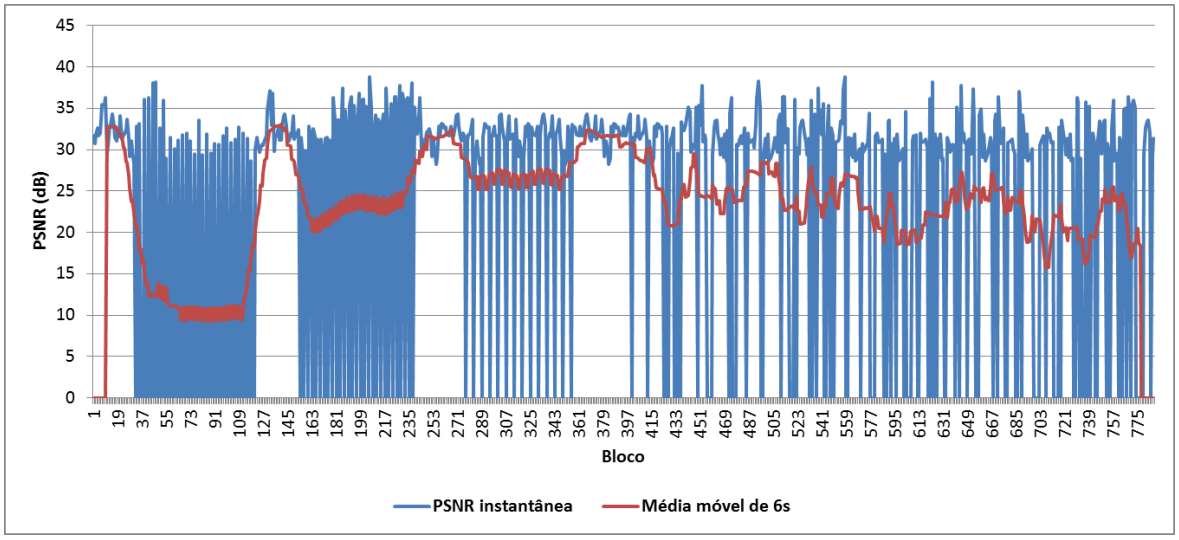


Figura 69 – PSNR medida no nó filho (cenário 4)

7.6 - CONCLUSÕES

Com base nos testes anteriormente descritos, verifica-se que a capacidade de adaptação dos blocos SVC em conjunto com o protocolo TFMCC favorece o tratamento da escalabilidade para a rede de distribuição de vídeo.

Nos cenários 1, 2 e 3 considerou-se, em síntese, um nó Scribe cuja largura de banda de *upload* vale 2 Mbit/s transmitindo vídeo ao vivo para 10 nós filhos, tal que os nós ingressam na rede um a um, em intervalos de 10 segundos. O vídeo foi codificado no formato H.264 SVC, conforme o paradigma em blocos independentes, fragmentado em pacotes de 3 KB, dotado de oito camadas de escalabilidade, nas taxas respectivas de 60, 85, 120, 170, 290, 400, 560 e 800 Kbit/s. Os cenários diferem conforme a Tabela 23.

Tabela 23 – Resumo dos cenários 1, 2 e 3

Cenário	**Adaptação**	**Retransmissão**
1	Não	Não
2	Sim	Não
3	Sim	Sim

Na Tabela 23, entende-se por "Adaptação" a ativação do módulo Adaptador SVC, proposta na seção 5.7, responsável por realizar o escalonamento dos blocos SVC em conjunto com a taxa de transmissão sugerida pelo protocolo TFMCC. O termo "Retransmissão" diz respeito à ativação do esquema de retransmissão em camada base dos últimos blocos SVC sempre que houver redução brusca da taxa, conforme proposto em 5.10.

As quatro figuras que seguem sumarizam os resultados obtidos nos cenários supracitados. São apresentados, respectivamente, o percentual médio de perda de blocos nos receptores, a taxa média recebida, o número médio de camadas recebidas e a PSNR média nos receptores. Os dados são totalizados para cada quantidade de nós filhos conectados na rede Scribe.

Verifica-se, na Figura 70, que a partir de 4 nós filhos, sem adaptação, a rede para de funcionar devido à perda de pacotes (fragmentos), que é de 100%. Com o módulo de adaptação habilitado e quatro nós receptores conectados, a perda de blocos cai para aproximadamente 13% e a rede consegue oferecer, em média, 3 camadas SVC (Figura 72).

Com adição da funcionalidade de retransmissão, que consiste em reenviar apenas a camada base SVC dos últimos blocos sempre que houver redução brusca da taxa sugerida pelo

TMFCC, a perda de blocos diminuiu para, aproximadamente, 7%. Adicionalmente, a rede conseguiu atender 10 nós filhos com uma perda média de blocos de 9%.

Vale lembrar que o percentual de perda de blocos representa o tempo médio percentual em que o vídeo esteve ausente nos receptores, sendo, portanto, uma métrica importante do ponto de vista da Qualidade de Experiência do usuário.

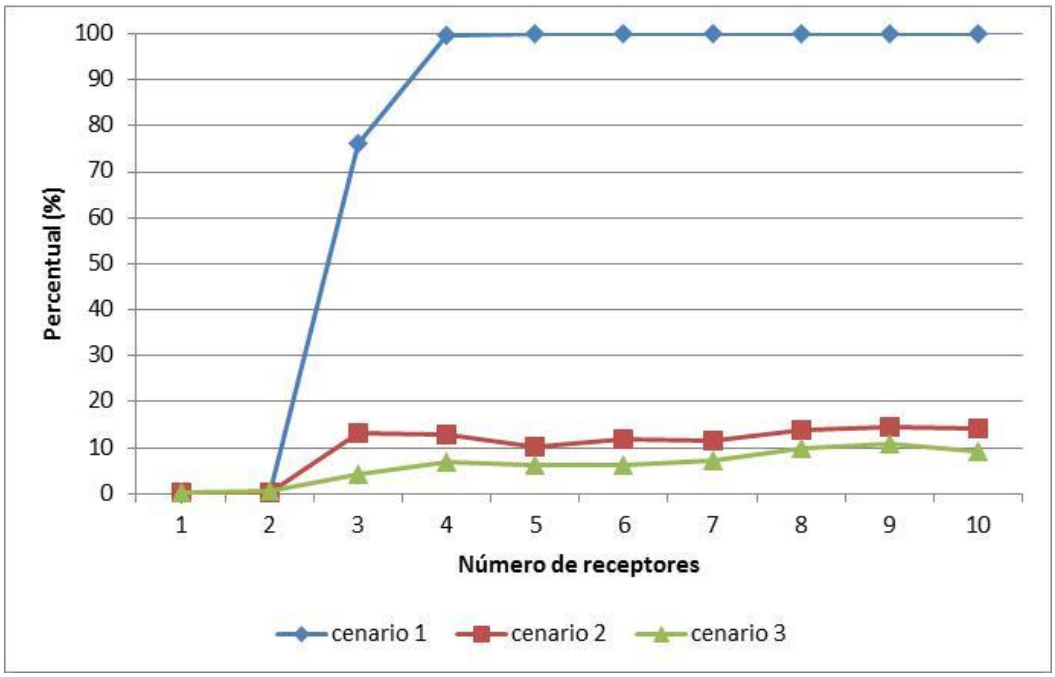


Figura 70 – Perda média de blocos nos receptores (cenários 1, 2 e 3)

Infere-se da sobreposição dos gráficos dos cenários 2 e 3 da Figura 71, que a retransmissão teve pouco efeito no sentido da taxa média recebida. Esse resultado é: (a) esperado, pois a camada base SVC possui *bitrate* baixo; e (b) desejado, pois não é intenção sobrecarregar a rede com retransmissão.

A ausência de um esquema adaptativo, que é abordada no cenário 1, implica em uma elevada taxa de transmissão, acima da capacidade do sistema, o que causa a grande perda de pacotes. Verifica-se, na Figura 71, que essa taxa é aproximadamente o dobro da taxa ajustada pelo TFMCC.

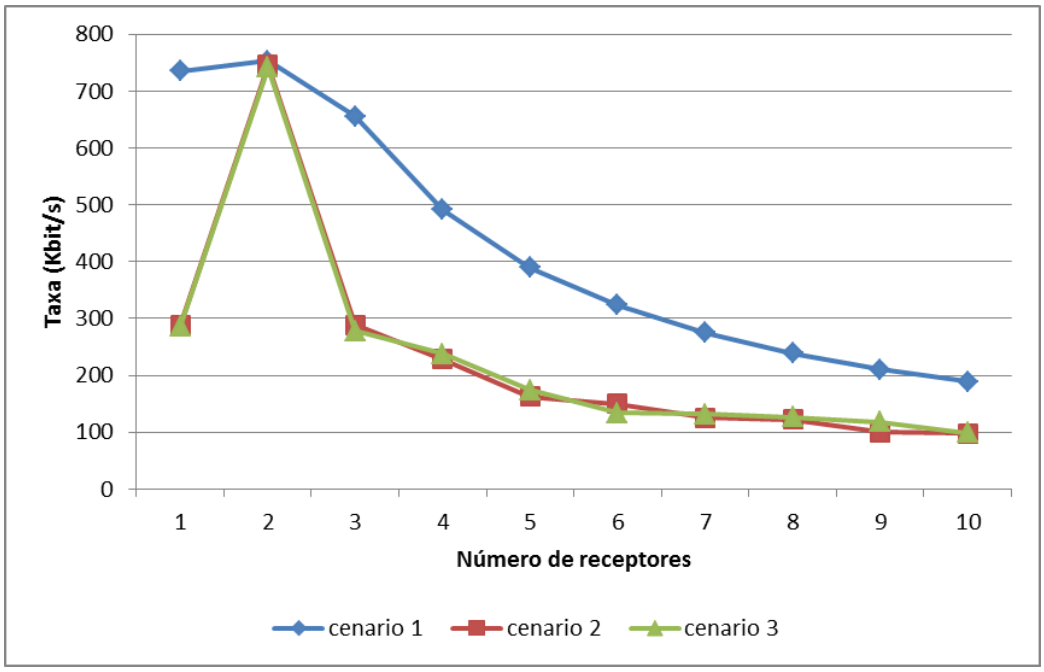


Figura 71 – Taxa média recebida (cenários 1, 2 e 3)

Sabe-se, da Figura 70, que a perda de pacotes se inicia com o ingresso do nó filho 3. O número médio de camadas nos receptores, conforme se observa na Figura 72, vale aproximadamente 3. No cenário 1, em que não há adaptação SVC, ocorre rápida degradação do sistema, de tal forma que, a partir do ingresso do nó 4, o número médio de camadas recebidas é zero, ou seja, não há vídeo nos receptores.

Com o esquema de adaptação ativo, cenários 1 e 2, o número médio de camadas recebidas cai apenas de 3 para 2 ao longo do teste realizado, permitindo, portanto, escalabidade do sistema no que diz respeito à quantidade de usuários sendo favorecidos pela transmissão ao vivo do vídeo.

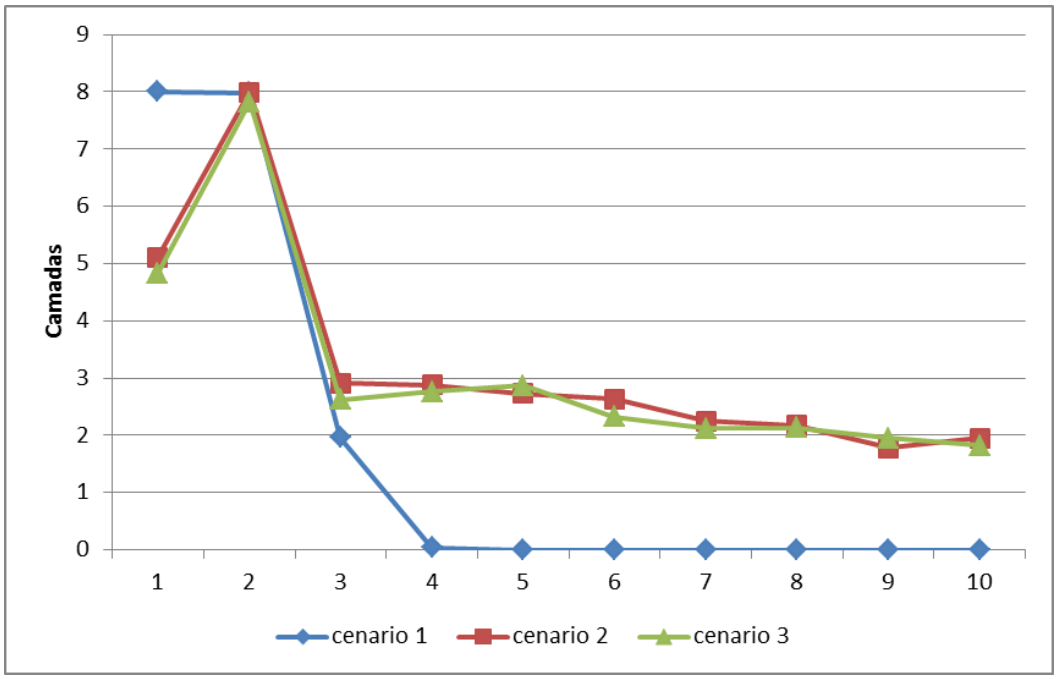


Figura 72 – Número médio de camadas SVC nos receptores (cenários 1, 2 e 3)

Verifica-se, na Figura 73, comparando-se os cenários 1 e 2, que a proposta de retransmissão trouxe ganho de aproximadamente 1 dB na PSNR de maneira quase independente da quantidade de nós receptores conectados. Esse ganho representa aumento de 11,5 % com respeito à faixa de variação da PSNR das sequências de vídeo consideradas nos testes. De maneira análoga aos gráficos anteriores, a PSNR vai para zero a partir do ingresso do nó 4, pois o vídeo está ausente nos receptores devido à alta perda de pacotes, causando perda total dos blocos SVC.

No último cenário testado, padrões distintos de perda de pacotes são aplicados em um enlace Scribe. Esses padrões estão definidos na Tabela 22 anteriormente apresentada. Pode-se concluir, conforme os experimentos realizados no cenário 4, que o TFMCC se estabiliza em determinada taxa de transmissão em função da quantidade de pacotes perdidos por unidade de tempo.

Em síntese, para uma perda de 2 pacotes/s no enlace Scribe, a taxa recebida estabiliza-se em, aproximadamente, 18 Kbit/s. Para uma perda de 1 pacote/s, a taxa estabiliza-se em 330 Kbit/s. Por fim, para uma perda de 0,5 pacote/s, a taxa vai para 740 Kbit/s.

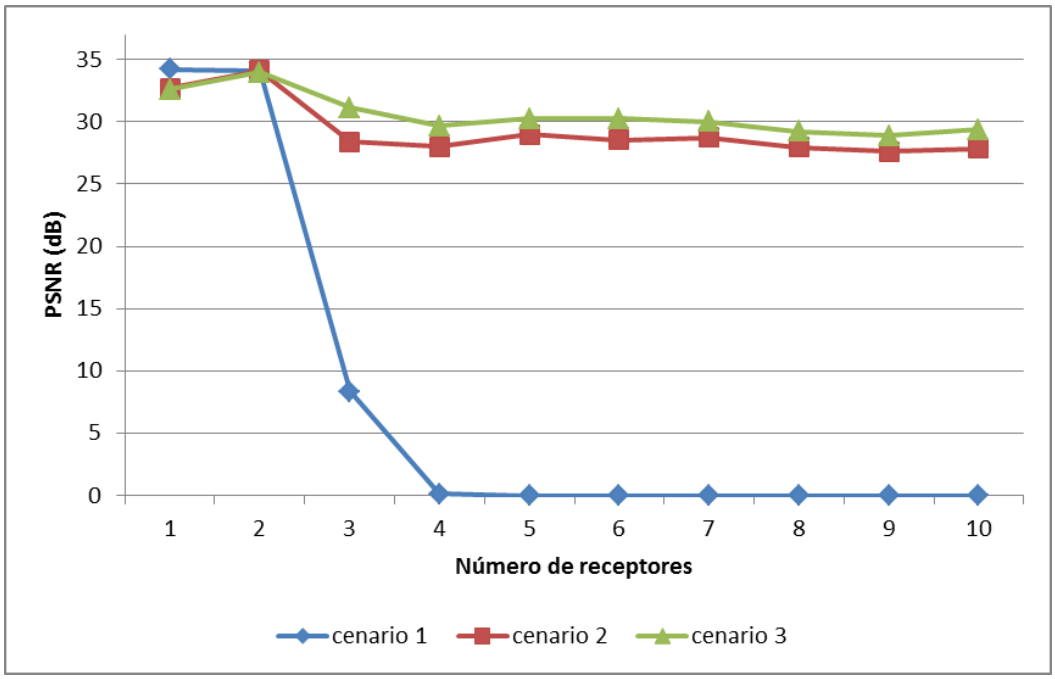


Figura 73 – PSNR medida nos receptores (cenários 1, 2 e 3)

Com respeito ao cenário 4, verifica-se que quando a perda de pacotes no encale Scribe é percentual, como ocorre no período P7, então o TFMCC tem um comportamento menos estável, de forma que a taxa aproxima-se de uma função “dente de serra”. Ocorre que quando o TFMCC sobe a taxa, a perda resultante, em termos absolutos, aumenta. Isso causa a redução da taxa e, por consequência, da perda absoluta. Em função disso, o TFMCC tenta novamente subir a taxa e assim o ciclo se repete.

8 - CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou uma arquitetura para transmissão *multicast* em tempo real de vídeo escalável H.264 SVC sobre rede P2P, que provê adaptação dinâmica da qualidade de vídeo com base no protocolo de controle de congestionamento TFMCC.

A capacidade computacional requerida pelos *softwares* decodificadores SVC atuais é alta, levando à necessidade de soluções que viabilizem o uso da extensão sem custo de processamento adicional expressivo. Nesse sentido a solução proposta busca solucionar o desafio ligado ao transporte de vídeo SVC, de forma a poder aproveitar as características favoráveis providas pela escalabilidade de vídeo dentre as quais a adaptabilidade de conteúdo em estreita relação com a disponibilidade de recursos de rede.

Diversas dificuldades de implementação com uso de vídeo escalável SVC foram superadas, especialmente as relacionadas ao tratamento da camada de abstração de rede NAL (*Network Abstraction Layer*) durante o processo de decodificação. Em especial, a estratégia de compressão de vídeo em blocos independentes e de curto intervalo de tempo trouxe simplificação substancial para a integração de qualquer decodificador SVC. Para esse fim os blocos podem ser dotados das três dimensões de escalabilidade SVC – a temporal, a espacial e a de qualidade, oferecendo assim grande flexibilidade na adequação entre disponibilidade de banda e demanda a ser apresentada por um dado provedor de serviços de vídeo.

Uma estrutura de distribuição em árvore foi considerada para a rede P2P, possibilitando o encaminhamento ao vivo do vídeo com baixo atraso e reduzindo a sobrecarga de controle relativa à manutenção da rede. Para construção dessa rede em árvore utilizou-se o Pastry em conjunto com o Scribe.

O protocolo RTP foi estendido para suportar informações do protocolo de controle de congestionamento TFMCC, auxiliando o processo de ajuste dinâmico da qualidade de vídeo dos blocos SVC. Discussões a respeito do dimensionamento de fragmentos foram apresentadas. Assim, de forma integrada ao RTP, o protocolo TFMCC pôde ser empregado com base em diferentes instâncias, contribuindo para a solução do seu problema de escalabilidade e assegurando suavidade no processo de alocação de banda, sem comprometer significativamente outras conexões TCP. Outro aspecto que favoreceu o emprego do TFMCC envolve a utilização de fragmentos de tamanho fixo dos blocos de vídeo, com reflexos em aspectos referentes a *buffering*.

Uma possibilidade de melhoramento da arquitetura é utilizar fragmentação ciente de conteúdo, ou seja, capaz de tratar as delimitações de início e de fim de NALU. Nesse caso,

o fragmentador precisa implementar mecanismos para evitar grandes variações no tamanho do fragmento, conforme requisito do TFMCC. Pode-se ainda realizar a transmissão das unidades NAL do GOP de maneira priorizada, conforme ordem das camadas de dependência do SVC.

Outra evolução da arquitetura diz respeito ao Remontador de blocos. Na proposta atual, a decodificação de blocos é atômica, de forma que fragmentos perdidos implicam perda total do bloco. Porém, um receptor mais robusto poderia tratar blocos corrompidos e buscar reaproveitar ao máximo as AUs recebidas. Essa solução talvez exija cabeçalhos adicionais no encapsulamento do bloco, além de recursos específicos no software decodificador SVC. É, entretanto, uma alternativa interessante para dispositivos dotados de maior poder computacional.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] G. Camarillo, "Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples, and Applicability," *RFC 5694,* 2009.

[2] J. Risson e T. Moors, "Survey of Research towards Robust Peer-to-Peer Networks: Search Methods," *Computer Networks,* vol. 50, n. 17, pp. 3485-3521, 2006.

[3] Y. Liu, Y. Guo e C. Liang, "A survey on peer-to-peer video streaming systems," *Peer-to-Peer Networking and Applications,* vol. 1, n. 1, pp. 18-28, 2008.

[4] A. Sentinelli, L. Celetto, D. Lefol, C. Palazzi, G. Pau, T. Zahariadis e A. Jari, "Survey on P2P Overlay Streaming Clients," *In Future of the Internet Conference Prague,* 2009.

[5] C. Yeo, B. Lee e M. Er, "A survey of application level multicast techniques," *Computer Communications,* vol. 27, n. 15, pp. 1547-1568, 2004.

[6] J. Peltotalo, J. Harju, A. Jantunen, M. Saukko e L. Vaatamoinen, "Peer-to-Peer Streaming Technology Survey," *ICN '08: Proceedings of the Seventh International Conference on Networking,* pp. 342-350, 2008.

[7] E. Keong Lua, J. Crowcroft, M. Pias, R. Sharma e S. Lim, "A Survey and Comparison of Peer-to-peer Overlay Network Schemes," *IEEE Communications Surveys and Tutorials,* vol. 7, n. 5, pp. 72-93, 2005.

[8] I. M. Moraes, M. E. M. Campista, M. D. D. Moreira, M. G. Rubinstein, L. H. M. K. Costa e O. C. M. B. Duarte, "Distribuição de Vídeo sobre Redes Par-a-Par: Arquiteturas, Mecanismos e Desafios," *SBRC '08: Em Minicursos do Simpósio Brasileiro de Redes de Computadores,* pp. 115-171, 2008.

[9] S. Deering e D. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs," *ACM Transactions on Computer Systems,* vol. 8, n. 2, pp. 85-110, 1990.

[10] J. Jannotti, D. K. Gifford, K. L. Johnson, F. Kaashoek e J. W. O'Toole, "Overcast: Reliable Multicasting with an Overlay Network," *OSDI '00: Proceedings of the 4th Conference on Symposium on Operating Systems Design and Implementation,* vol. 4, pp. 14-14, 2000.

[11] Y.-H. Chu, S. G. Rao e H. Zhang, "A case for end system multicast," *IEEE Journal on Selected Areas in Communications In Selected Areas in Communications,* vol. 20, n. 8, pp. 1456-1471, 2002.

[12] S. Zhuang, B. Zhao, A. Joseph, R. Katz e J. Kubiatowicz, "Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination," *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video,* pp. 11-20, 2001.

[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp e S. Shenker, “A scalable content-addressable network,” *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications,* pp. 161-172, 2001.

[14] M. Castro, P. Druschel, A.-M. Kermarrec e A. Rowstron, “SCRIBE: a large-scale and decentralized application-level multicast infrastructure,” *IEEE Journal on Selected Areas in Communications In Selected Areas in Communications (JSAC),* vol. 20, n. 8, pp. 1489-1499, 2002.

[15] A. Rowstron e P. Druschel, “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems,” *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware),* pp. 329-350, 2001.

[16] ITU-T, “Rec. H.264 - Advanced video coding for generic audiovisual,” 2009.

[17] R. Iain E. , The H.264 Advanced Video Compression Standard, 2 ed., John Wiley and Sons, 2010, p. 346.

[18] P. Seeling e M. Reisslein, “Video Transport Evaluation With H.264 Video Trace,” *Aceito para publicação em IEEE Communications Surveys and Tutorials,* 2011.

[19] T. I. “Overview of Temporal Scalability With Scalable Video Coding (SVC),” *Application Report SPRABG3,* 2010.

[20] T. Klingberg e R. Manfredi, “Gnutella 0.6,” *RFC draft. http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html,* 2002.

[21] I. Clarke, S. Oskar, O. Wiley e T. W. Hong, “Freenet: A distributed anonymous information storage and retrieval system,” *Workshop on Design Issues in Anonymity and Unobservability,* pp. 46-66, 2000.

[22] B. Zhao, J. Kubiatowicz e A. Joseph, “Tapestry: an infrastructure for fault-tolerant wide-area location and routing,” *Technical Report No. UCB/CSD-01-1141. University of California, Berkeley,* 2001.

[23] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek e H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” *SIGCOMM '01: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications,* pp. 149-160, 2001.

[24] C. Tang, Z. Xu e M. Mahalingam, “pSearch: information retrieval in structured overlays,” *ACM SIGCOMM Computer Communication Review,* vol. 33, n. 1, pp. 89-94, 2003.

[25] W. Nejdl, S. Decker e W. Siberski, “EDUTELLA: a P2P networking infrastructure based on RDF,” *WWW '02: Proceedings of the 11th International Conference on World Wide Web,* pp. 604-615, 2002.

[26] X. Liao, H. Jin, Y. Liu, L. Ni e D. Deng, “Anysee: Peer-to-Peer Live Streaming,”

INFOCOM '06: Proceedings of the 25th 25th IEEE International Conference on Computer Communications, pp. 1-10, 2006.

[27] X. Hei, C. Liang, J. Liang, Y. Liu e K. W. Ross, "A Measurement Study of a Large-scale P2P IPTV System," *IEEE Transactions on Multimedia*, vol. 9, n. 8, pp. 1672-1687, 2007.

[28] X. Zhang, J. Liu, B. Li e T. Yum, "Coolstreaming/DONet: A Data-driven Overlay Network for Efficient Live Media Streaming," *INFOCOM '05: Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, pp. 2102-2111, 2005.

[29] A. Sentinelli, G. Marfia, S. Tewari, M. Gerla e L. Kleinrock, "Will IPTV Ride the P2P Stream?," *IEEE Communication Magazine*, vol. 45, n. 6, 2007.

[30] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron e A. Singh, "Splitstream: High-bandwidth Multicast in Cooperative Environments," *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pp. 298-313, 2003.

[31] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz e I. Stoica, "Towards a Common API for Structured Peer-to-Peer Overlays," *Peer-to-Peer Systems II*, vol. 2735, pp. 33-44-44, 2003.

[32] V. Venkataraman, K. Yoshida e P. Francis, "Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast," *ICNP '06: Proceedings of the IEEE International Conference on Network Protocols*, pp. 2-11, 2006.

[33] V. Padmanabhan, H. J. Wang e P. A. Chou, "Supporting Heterogeneity and Congestion Control in P2P Multicast Streaming," *IPTPS '04: Proceedings of the Third international conference on Peer-to-Peer Systems*, pp. 54-63, 2004.

[34] S. Banerjee, B. Bhattacharjee e C. Kommareddy, "Scalable Application Layer Multicast," *SIGCOMM '02: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, vol. 32, n. 4, pp. 205-217, Aug. 2002.

[35] F. Pianese, D. Perino, J. Keller e E. W. Biersack, "PULSE: an Adaptive, Incentive-based, Unstructured P2P Live Streaming System," *IEEE Transactions on Multimedia*, vol. 9, n. 8, pp. 1645-1660, 2007.

[36] S.-J. Seok, W.-C. Song e D. Choi, "Implementation of Pastry-based P2P system to share sensor data," *International Journal of Sensor Networks (IJSNET)*, vol. 8, n. 3/4, pp. 125-135, 2010.

[37] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang e A. Wolman, "An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer Overlays," *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 2, pp. 1510-1520, 2003.

[38] S. Asioli, N. Ramzan e E. Izquierdo, “Efficient Scalable Video Streaming over P2P Network,” *User Centric Media,* vol. 40, pp. 153-160, 2010.

[39] O. Abboud, K. Pussep, A. Kovacevic e R. Steinmetz, “Quality Adaptive Peer-to-Peer Streaming using Scalable Video Coding,” *Wired-Wireless Multimedia Networks and Services Management,* vol. 5842, pp. 41-54, 2010.

[40] T. Schierl, Y. Sánchez, C. Hellge e T. Wiegand, “Improving P2P Live-Content Delivery using SVC,” *VCIP '10: Visual Communications and Image Processing. Proceedings of the SPIE.,* vol. 7744, pp. 77440S-77440S-12, 2010.

[41] M. Fesci-Sayit, E. T. Tunali e A. M. Tekalp, “Bandwidth-aware multiple multicast tree formation for P2P scalable video streaming using hierarchical clusters,” *16th IEEE International Conference on Image Processing (ICIP),* pp. 945-948, 2009.

[42] C. G. Plaxton, R. Rajaraman e A. W. Richa, “Accessing nearby copies of replicated objects in a distributed environment,” *Theory of Computing Systems,* vol. 32, n. 3, pp. 241-280, 1999.

[43] X. Jiang, F. Safaei e P. Boustead , “Enhancing the multicast performance of structured P2P overlay in supporting Massively Multiplayer Online Games,” *ICON '07: Proceedings of the 15th IEEE International Conference on Networks,* pp. 124-129, 2007.

[44] J. F. Kurose e K. W. Ross, Computer Networking: A Top-Down Approach Featuring the Internet, 5 ed., Addison Wesley, 2009.

[45] S.-C. Tsao, Y.-C. Lai e Y.-D. Lin, “Taxonomy and Evaluation of TCP-Friendly Congestion-Control Schemes on Fairness, Aggressiveness and Responsiveness,” *IEEE Network,* vol. 21, n. 6, pp. 6-15, 2007.

[46] B. Braden, D. D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski e L. Zhang, “Recommendations on Queue Management and Congestion Avoidance in the Internet,” *RFC 2309,* 1998.

[47] Y. Zhang, N. Duffield, V. Paxson e S. Shenker, “On the Constancy of Internet Path Properties,” *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement,* pp. 197-211, 2001.

[48] J. Widmer e M. Handley, “TCP-Friendly Multicast Congestion Control (TFMCC),” *RFC 4654,* 2006.

[49] P. Zhu, H. Yoshiuchi e S. Yoshizawa, “QoS-aware multicast for Internet video appplications,” *15th IEEE International Packet Video Workshop (PV),* pp. 46-51, 2007.

[50] H. Schulzrinne, S. Casner, R. Frederick e V. Jacobson, “A Transport Protocol for Real-Time Applications,” *RFC 3550,* 2003.

[51] G. Stuer, K. Vanmechelen, J. Broeckhove e T. Dhaene, “Sleeping in Java,”

Proceedings of the International Euromedia 2004 Conference, pp. 74-78, 2004.

[52] G. Kioumourtzis, C. Bouras e A. Gkamas, "TFMCC versus ASMP: lessons learned from performance evaluation," *International Journal of Network Management,* vol. 22, n. 5, pp. 349-372, 2012.

[53] H. Schwarz, D. Marpe e T. Wiegand, "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 17, n. 9, pp. 1103-1120 , 2007.

[54] G.-I. Kown e J. W. Byers, "Smooth multirate multicast congestion control," *INFOCOM 2003. 22th Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies,* pp. 1022 - 1032 vol.2, July 2003.

[55] Q. Wang, K. Long, S. Cheng e R. Zhang, "TCP-Friendly Congestion Control Schemes in the Internet," *International Conference on Info-tech and Info-net Conference,* 2001.

APÊNDICES

A - CÁLCULO DO POC (*PICTURE ORDER CODE*)

O algoritmo de cálculo do POC (*PicOrderCnt*) está descrito em 8.2 de [16].

Para o tipo de contagem 0 (*cnt_type* = 0 no SPS), o algoritmo está definido em 8.2.1 de [16].

pic_order_cnt_lsb	IDR	POCMsb	prevPOCMsb	prevPOCLsb	TopFieldOrderCnt	Esperado
			0	0		
0	1	0	0	0	0	0
4	0	0	0	0	4	4
2	0	0	0	4	2	2
1	0	0	0	2	1	1
3	0	0	0	1	3	3
8	0	0	0	3	8	8
6	0	0	0	8	6	6
5	0	0	0	6	5	5
7	0	0	0	5	7	7
12	0	0	0	7	12	12
10	0	0	0	12	10	10
9	0	0	0	10	9	9
11	0	0	0	9	11	11
0	1	0	0	0	0	0
62	0	-64	0	0	-2	-2
61	0	-64	-64	62	-3	-3
63	0	-64	-64	61	-1	-1
4	0	0	-64	63	4	4
2	0	0	0	4	2	2
1	0	0	0	2	1	1
3	0	0	0	1	3	3
8	0	0	0	3	8	8
6	0	0	0	8	6	6
5	0	0	0	6	5	5
7	0	0	0	5	7	7
12	0	0	0	7	12	12
10	0	0	0	12	10	10
9	0	0	0	10	9	9
11	0	0	0	9	11	11
0	1	0	0	0	0	0
62	0	-64	0	0	-2	-2
61	0	-64	-64	62	-3	-3

Onde:

- *MaxPOCLsb = 2^(log2_max_pic_order_cnt_lsb_minus4 + 4)*
- *log2_max_pic_order_cnt_lsb_minus4* vem do SPS
- *prevPOCMsb* e *prevPOCLsb* valem zero se *currPic* é IDR
- *prevPOCMsb = POCMsb* anterior se *currPic* não é IDR
- prevPOCLsb = *pic_order_cnt_lsb anterior* se *currPic* não é IDR
- *POCMsb* é calculado conforme 8.2.1 de [16]
- *TopFieldOrderCnt* (*BottomFieldOrderCnt*) é calculado conforme 8.2.1 de [16]
- *POC (currPic) = Min (TopFieldOrderCnt, BottomFieldOrderCnt)*

Informações sobre *num_ref_frames* estão em 8.2.5.3 de [16].

Para o exemplo acima, foi considerada a seguinte configuração do JSVM Encoder:

```
SourceWidth     624
SourceHeight    352
FrameRateIn     24
FrameRateOut    24
IDRPeriod       16
SliceMode       0
UseRedundantSlc 0
```

B - INVESTIGAÇÕES DO *JSVM EXTRACTOR*

Sejam as seguintes definições de [53]:

- DEF1: *"Switching between different dependency layers (D) is only envisaged at defined switching point (IDR access units). However, switching between different quality refinement layers (Q) is virtually possible in any access unit. Quality refinements can either be transmitted as new dependency layers (CGS: different D) or as additional quality refinement layers (MGS: same D, different Q)."*
- DEF2: *"For quality refinement layers with a quality identifier Q > 0, always the preceding quality layer with a quality identifier Q - 1 is employed for inter-layer prediction. For quality layers with Q = 0, any present quality layer of a lower dependency layer can be selected as reference layer for inter-layer prediction."*

Por investigação, que quando Q = 0 o *JSVM Decoder* usa a camada inferior (D-1) onde Q = 0 mesmo que as camadas Q > 0 de D - 1 estejam mantidas no *bytestream*. O vídeo reconstruído é o mesmo.

ESTUDO 1 - JSVM NO MODO CGS

Seja o *bytestream* CGS:

```
layer  DTQ   baseLayerId
------------------------
0      000   -
1      010   -
2      100   0
3      110   0
4      200   0
5      210   0
6      300   0
7      310   0
------------------------
```

Observações importantes:

a) Em *baseLayerId*, "layer" significa o arquivo de configuração, que coincide com o D do DTQ (pois é CGS). No caso do MSG o *baseLayerId* é automático conforme DEF2 acima;
b) Para o *encoder* cada "layer" é um arquivo de configuração. Esse *layer*, se alterar a resolução do vídeo (com relação ao *layer* n-1) ou se manter a resolução e for CGS, então esse *layer* é uma dimensão D do DTQ. Se manter a resolução e for MGS, o *layer* é uma dimensão Q no DTQ (e pertence ao mesmo D do *layer* n-1);

c) Para o *extractor*, "layer" é uma possível combinação de DTQ;
d) *baseLayerId* pode assumir no máximo 2 valores distintos no JSVM.

O comando *extractor* com parâmetros "–sl 4" gera a saída abaixo. O parâmetro –sl A significa extrair um *bytestream* contendo a camada A e suas dependências.

```
DTQ
-----------
000
200
-----------
```

O conteúdo do arquivo não coincide com o relatório do *extractor*. Dentro do *bytestream* fica presente a camada DTQ 100.

O comando *extractor* "–l 2 –t 0 –f 0" gera a saída abaixo. Os parâmetros –l A –t B –f C significam extrair as camadas D<=A,T<=B,Q<=C.

```
DTQ
-----------
000
100
200
-----------
```

O conteúdo do arquivo coincide com o relatório do *extractor*.
Calculando a PSNR obtemos resultados iguais, indicando que a camada DTQ 100 é desnecessária. Ou seja, o extractor "–l 2 –t 0 –f 0" retorna um *bytestream* maior que o necessário.

Ocorre que D=1 não é referencia para D=2, pois o *baseLayerId* é zero.

Analisando o *bytestream* gerado pelos dois comandos, verifica-se que a única diferença reside na primeira NALU, que é do tipo 6 (SEI). O relatório gerado pelo *extractor –sl* está correto (sem DTQ 100), porém parece haver uma falha no *software* pois há inconsistência com resultado gerado (que contém 100). Em resumo, o parâmetro –sl do *extractor* corrige o SEI mas não trunca as camadas corretamente, causando um *bitrate* maior que o esperado.

Por sorte o CGS não é escopo do trabalho, pois há limite de apenas 2 valores de *baseLayerId* e a eficiência de RD (*Rate-Distortion*) é baixa.

ESTUDO 2 - JSVM NO MODO MGS

Seja o *bytestream* MGS:

```
layer  DTQ
------ -----
0     000
1      010
2      001
3      011
4      100
5      110
6      101
7      111
-----------
```

O comando *extractor* com parâmetros "–sl 4" gera a saída abaixo. O parâmetro –sl A significa extrair um *bytestream* contendo a camada A e suas dependências.

```
DTQ
-----------
000
001
100
-----------
```

O conteúdo do arquivo coincide com o relatório do *extractor*.

O comando *extractor* com parâmetros "–l 1 –t 0 –f 0" gera a saída abaixo. Os parâmetros –l A –t B –f C significam extrair as camadas D<=A,T<=B,Q<=C.

```
DTQ
-----------
000
100
-----------
```

O conteúdo do arquivo coincide com o relatório do *extractor*.

Calculando a PSNR obteve-se resultados diferentes, indicando que a camada DTQ 001 é opcional. Ou seja, o extractor via –sl resultou no maior *bytestream* possível.

Com a análise dos *logs* do estudo MGS via *extractor* "–l –t –f" tem-se que a dependência entre as camadas ocorre da seguinte forma (para D=0):

```
PSNR
 | (2) > (3)
 |
 |  ^     ^
 |
 | (0) > (1)
-------------FPS
```

Entre as camadas espaciais (D=0 e D=1):

```
QCIF      CIF
(0 ou 2)   >   (4)
(1 ou 3)   >   (5)
(2)        >   (6)    *1
(3)        >   (7)    *2
```

*1: Tecnicamente seria possível 0 ou 2, porém não há comando *extractor* para isso. Extrair (6) via "–l –t –f" ou "–sl" mantém a camada (2).
*2: Tecnicamente seria possível 1 ou 3, porém não há comando *extractor* para isso. Extrair (7) via "–l –t –f" ou "–sl" mantém a camada (3).

C - *ADAPTIVE MULTICAST STREAMING BASED ON SCALABLE VIDEO BLOCKS AND TFMCC PROTOCOL*

Artigo publicado no evento: IEEE International Conference on Communications (ICC 2012)

Simpósio: Communication Software Services and Multimedia Applications Symposium

Data: 06/2012

Adaptive Multicast Streaming Based on Scalable Video Blocks and TFMCC Protocol

Bernardo Vergne Dias[1], Paulo Roberto de Lira Gondim[2]
Universidade de Brasilia (UnB)
Brasilia, DF, Brazil
[1]bernardo.dias@gmail.com, [2]pgondim@unb.br

Abstract—This paper presents an architecture for live streaming of scalable video, which is encoded using the H.264 SVC (Scalable Video Coding) standard and transmitted over multicast groups on a peer-to-peer (P2P) network. The architecture has two characteristics: 1) the utilization of a paradigm of video compression based on independent compressed blocks in order to minimize some of the practical difficulties of the decoding process of the H.264 SVC when it is transmitted in real time over unreliable channel; 2) bitstream adaptivity through dynamic quality adjustment of the video blocks on an Internet best-effort environment, leading to a significative improvement in the quality of received video sequences.

Scalable Video, Multicast, Content Adaptation, P2P

INTRODUCTION

Several studies show the applicability of peer-to-peer architectures for various purposes [54] [55] [56] [57], including video streaming. In order to provide better Quality of Service (QoS) and Quality of Experience (QoE), the distribution system must be endowed with the ability of adaptation, which must take into account, for example, variations in terms of bandwidth availability and the possibility of packet loss and jitter. The use of video coding standard H.264 SVC [58] is a key option to meet this need, as a kind of Content Adaptation Technique (CAT). Moreover, the use of the TFMCC (TCP-Friendly Multicast Congestion Control) protocol [59], has been considered due to its fair and moderately aggressive behavior, as well as the possibility of adequate fitting to the proposed video compression paradigm, to be presented in this paper.

In spite of the native support of H.264 SVC for content adaptation, some difficulties related to the implementation of the standard need to be overcome, especially in the scenario of real-time streaming over unreliable channels. These difficulties are mainly related to the computational cost of processing of headers of NAL (Network Abstraction Layer) units for properly decoding in the scenario of packet loss and picture time stamping to fit dynamic FPS (Frames per Second) of a SVC video.

In order to minimize these difficulties, an architecture for live streaming of scalable video is proposed, encompassing two relevant characteristics: 1) the utilization of a paradigm of video compression based on independent compressed blocks (ICB), each block refering to a short period of time and being encapsulated into RTP (Real-Time Protocol) [7] packets; 2) the adaptivity of the bitstream, through dynamic quality adjustment of the video blocks on an Internet best-effort environment. ICB was proposed in order to minimize some of the practical difficulties of the decoding process of the H.264 SVC when it is transmitted in real time over unreliable channel. We observe that ICB works seamlessly with the TFMCC behavior, while the bitstream adaptivity is provided by the combination of TFMCC and some properties associated to scalable coding. Additionally, technical details regarding the implementation of the proposed architecture are discussed, and some paths of success and error related to the development of the proof of concept are presented.

The rest of the paper is organized as follows. Section II deals with the network abstraction layer of the H.264 SVC. Sections III and IV present a brief overview of P2P networks and TFMCC, respectively. Section V describes the problem. Section VI contains the architecture proposal. Section VII describes the implementation and section VIII reports evaluation results. Main conclusions are outlined in Section IX. Finally, future work is presented in Section X.

H.264 SVC STANDARD

The H.264 standard encodes the video data in the form of a sequence of NAL units according to the hierarchy shown in Figure 1.

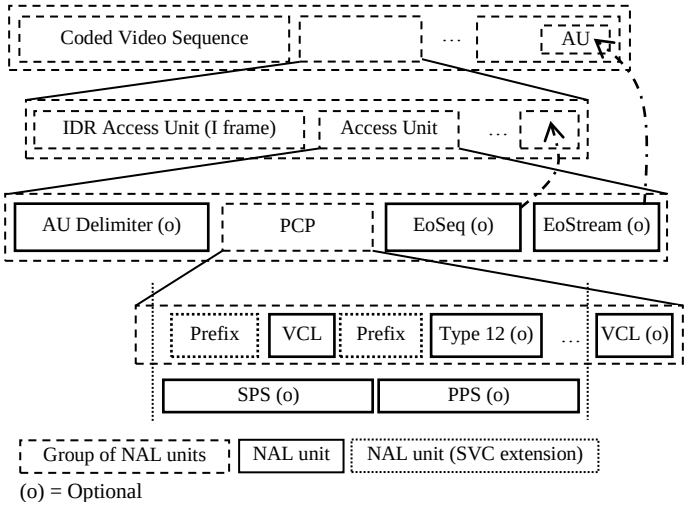


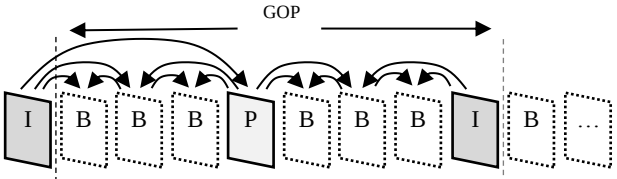
Figure 1. Structure of H.264 SVC, based on [5]

The IDR (Instantaneous Decoding Refresh) is the first AU (Access Unit) of a coded video sequence. Each AU is a frame (or field). SPS (Sequence Parameters Set) and PPS (Picture Parameters Set) carry fundamental information for decoding one or more AU. If not present, they are inherited from the previous AU. PCP (Primary Coded Picture) is a group of VCLs (Video Coding Layers) which are parts (slices) of a frame (or

field). “EoSeq” means End of coded video Sequence and “EoStream” means end of last coded video sequence. Other types of units were omitted. Details about each NAL unit type can be found in [58].

The H.264 standard defines three kinds of picture ordering: (a) Decoding Order, the order in which the pictures are decoded at the receiver, (b) Display Order, the order in which the pictures are displayed on the screen, (c) Reference Order, the order in which the pictures are stored in the decoder’s memory to facilitate inter-frame prediction. The Display Order is calculated based on headers of VCL units. The most important header for that purpose is the POC (Picture Order Count), and its calculation depends on headers of VCL and SPS units.

Figure 2 shows the picture ordering of hierarchical GOP (Group of Pictures) structure of 8 frames. Row (C) means: picture p0 is first processed by the decoder, then p8 is processed, then p4, etc. The definition of the GOP differs from author to author. In [8], for example, it is said that the "GOP starts at frame I." However, the definition used in this paper, which is in agreement with the reference implementation JSVM (Joint Scalable Video Model), is: "The GOP finishes on each image of temporal layer T0".



(A)	T0	T3	T2	T3	T1	T2	T3	T2	T0	T3
(B)	p0	p1	p2	p3	p4	p5	p6	p7	p8	p9
(C)	0	4	3	5	2	7	6	8	1	12
(D)	p0	p8	p4	p2	p1	p3	p6	p5	p7	p16

Figure 2. Picture ordering in H.264 SVC. (A) Temporal layer. (B) Display Order / Sequential numbering of the pictures. (C) Decoding Order. (D) Transmission Order.

P2P NETWORKS FOR STREAMING

Peer-to-peer (P2P) networking represents a new paradigm related to the development of distributed network applications, where each peer acts as both client and server, in opposition to the traditional client-server service model (where a scalability problem can be easily verified).

Several networks have been proposed aiming to broadcast video streaming, such as Anysee [61], PPLive [62], Coolstreaming/DONet [63], SplitStream [64], ESM (End System Multicast) [65], PULSE [66] and Pastry [67] with Scribe [16].

Considering the overlay network structure, P2P streaming systems can be classified into tree-based and mesh-based. Pastry is a P2P mesh network infrastructure that can be used with Scribe as an overlay for multicast communication. The combination involving Pastry and Scribe represents an interesting option, considering valuable characteristics such as decentralization, scalability, large-scale, self-organization and fault tolerance. These characteristics, in addition to the availability of an updated and well-documented Java-based open-source implementation – FreePastry – leaded us to select it as a framework to compose our architecture.

The Scribe is based on the push communication strategy, in which information always spreads from the root to the child nodes until the end of the tree, similarly to the model adopted by the IP Multicast. The advantages of low latency and low overhead of control are interesting to live broadcast. The performance analysis of the Scribe [68] shows that it is, at worst case, twice as slow as IP Multicast, according to the RAD metric (Ratio between the Average Delay using Scribe and the average delay using IP Multicast). The distribution function of the overhead of link is very close to the IP Multicast.

TFMCC PROTOCOL

TFMCC [59] is a multicast version of the TFRC (TCP-Friendly Rate Control) protocol and is proposed for multimedia applications. Results [69] show that the TFRC protocol, in addition to meeting both criteria TCP-equivalence and TCP-equal share, reaches satisfactory equilibrium of aggressiveness, responsiveness and fairness when compared to several other proposals, such as GAIMD (General Additive Increase / Multiplicative Decrease), IIAD (Inverse Increase / Additive Decrease), SQRT (Square Root Increase / Decrease), SIMD (Square Increase / Multiplicative Decrease), AIAD/H (Additive Increase / Additive Decrease with History), TFRCP (TCP-Friendly Rate Control Protocol) and TEAR (TCP-Emulation at Receiver).

Reference [19] considers that rate-based congestion control protocols, such as TFMCC, are suitable for real-time multimedia applications due to low cost bandwidth and fair behavior when coexisting with TCP traffic. To provide scalability, TFMCC dedicates more processing load to the receiver nodes.

PROBLEM DESCRIPTION

In this section, some difficulties related to the live streaming of scalable video in P2P networks are presented. The first of them is related to the scarcity of resources involved in the existent SVC video decoders. Several free SVC video decoders (for example, OpenSVCDecoder and JSVM) as well as proprietary solutions analyzed (for example, MainConcept), only support complete NAL unit as input in each iteration of decoding. Thus, the first practical issue of SVC is the fragmentation of the coded video sequence in NAL units, since this process involves high computational cost.

The second difficulty concerns the proper time stamping of the restored pictures that are delivered to the SVC decoder. To explain this problem, consider a decoding process by iterations of entire AU (Access Unit), which means frame by frame or field by field. The extraction of AU from the received stream requires the implementation of algorithms for detecting start and end of AU. These algorithms require parsing the headers of VCL units, which is computationally costly. Note that this means a high-layer analysis on each node of the P2P network, due to quality adaptation performed on each hop. Likewise, the headers of VCL units often depend on header of the previous units, due to the historical dependence of NAL units. The maintenance of these units in the decoder’s memory, which is

necessary to achieve the calculations that have historical dependence, requires understanding the algorithms of storage and disposal of objects of the DPB (Decoded Picture Buffer), where are stored the pictures used for inter-frame prediction. The H.264 standard describes these algorithms in the section that explains the HDR (Hypothetical Reference Decoder).

A third difficulty is: when each AU is received at the decoder, the output can be null, a single picture or even a set of multiple restored pictures. This behavior is the result of the decoding order previously shown in Figure 2 in the previous section. Decoders tested (MPlayer, VLC) do not respect the Display Order. So, we need to order the output of the decoder according to the Display Order. To do this, it is necessary to calculate the POC ordering, which requires, once more, the analysis of the headers of the NAL units. Note that SVC has dynamic FPS rate, what makes the time stamping process a not trivial task.

The Table I shows the corresponding result for the Figure 2.

TABLE I. SVC DECODER OUTPUT AFTER EACH ITERATION OF AU

Iteration	Input	Output	Comment
0	p0	None	p0 stored in DPB because it's referenced by p1, p2, p4
1	p8	None	p8 stored (referenced by p9, p10, p12)
2	p4	None	p4 stored (referenced by p2, p3, p5, p6)
3	p2	None	p2 stored (referenced by p1, p3)
4	p1	None	p1 decoded, but waits p0 output (decoder output must obey Display Order)
5	p3	None	p3 decoded, but waits output of p0, p1 and p2
6	p6	None	p6 stored (referenced by p5, p7)
7	p5	None	p5 decoded, but waits output from p0 to p4
8	p7	None	p7 decoded, but waits output from p0 to p6
9	p16	None	p16 stored (referenced by p17, p18, p20)
10	p12	None	p12 stored (referenced by p10, p11, p13, p14)
11	p10	None	p10 stored (referenced by p9, p11)
12	p9	p0, p1, p2, p3, p4, p5, p6, p7, p8, p9	p0 is released, as their dependents have been processed. Others are released according to the Display Order.

Another difficulty is related to the transmission of SVC over RTP, considering the RFC draft presented in [20]. The draft considers the inclusion of several new headers, including redundancy of parameters in NAL units and aiming to be efficient from the point of view of treatment of packet loss. The proposal is still computationally expensive and does not prevent the analysis of the headers (high layer) in the process of decoding and packetizing.

PROPOSAL

To reduce the computational costs described above, it is proposed the Independent Compressed Blocks (ICB) paradigm: in place of transmit small pieces of a long coded video sequence, we propose to split the video in several small videos and encode them separately. Each small video, called block, has a complete GOP structure. The concatenation of the decoded blocks will allow to recover the original video .

The basic structure of each node is:

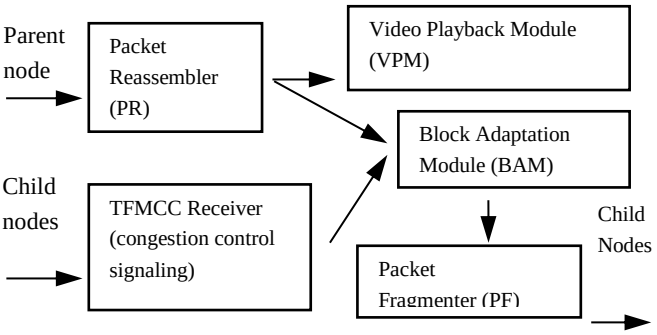


Figure 3. Components of the architecture

The dimensions of the block in terms of GOP size (pictures per block) and in terms of bytes are characteristics to be determined by the system designer as every need. Large GOP size allows more granularity over the scalability of SVC, resulting in better match the bandwidth of the network. Larger GOP, however, increases the delay of video playback, since the block is decoded atomically. The size in bytes of the fragments is related to the precision of TFMCC and the constraints of network layer or link. It is proposed to use the SSM mode (Source-Specific Multicast) and perform forward signaling through piggyback. The adaptation done by BAM (Block Adaptation Module) must meet the suggested bit rate of the TFMCC module as close as possible. An instance of TFMCC is created for each node of the tree (and their first level children).

Advantages: (a) Independence of the compression technique: it is not necessary to treat the interpretation of features such as POC and HRD, among others, (b) Error-free decoding: each block is a valid and full stretch SVC video. So the decoder output will always be all pictures of the GOP, (c) Parallel decoding, since the blocks do not have dependency between themselves, (d) Time stamping of the decoded pictures becomes trivial, since the rate of blocks per second is fixed. The time stamps are distributed over the duration of the block according to the GOP structure.

Disadvantages: (a) The delay at each network node is at least the duration of a block (GOP) for both video playback, adaptation and switching in the Scribe tree; (b) The blocks need to be fragmented to meet the restriction of the fixed packet size of TFMCC.

IMPLEMENTATION

The software needed to implement the proof of concept was written in Java and C.

The VPM (Video Playback Module) has two shared memories for storing reconstructed video blocks. Two mutexes ensure that while one memory is being read by the VPM, the other is being written by the decoder. A low frequency timer (f = block per second) controls the SVC decoder and the write access of shared memory. A high-resolution timer (f = frame per second) reads a region of the shared memory related to the current time stamp. Besides the two clocks, the schema of reproduction based on OpenGL (Open Graphics Library) has a

third clock responsible for updating the image on the screen (the one read by the high-resolution timer). The OpenGL clock should be higher than the highest possible FPS video. The calibration of both high-resolution clocks results in the effect called Pull-Down. Graphical user were developed interfaces using Java Swing with Java 2D, Java Swing with JOGL (Java Binding for the OpenGL) and then, finally, Eclipse SWT (Standard Widget Toolkit) with LWJGL (Lightweight Java Game Library), which resulted in satisfactory performance.

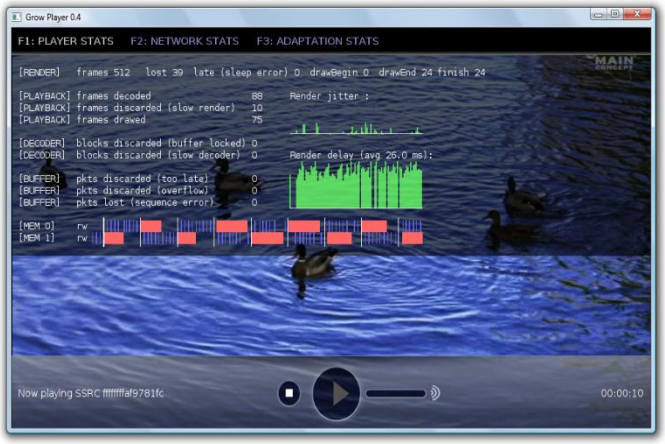


Figure 4. Screenshot of the implemented program

We tried the following techniques for integration of the SVC decoder with the VPM: (a) Native C library via JNA (Java Native API); (b) Pipe files (or FIFO, First in First out) of the operating system (Linux); (c) Standard in/out of the operating system, and (d) Socket. The reference decoder JSVM showed extremely poor performance (about 300 times slower than a proprietary solution). The OpenSVCDecoder generated unexpected results. Some care has been taken in order to acquire good optimization of the JNA integration. Note that a reconstructed video of 24fps at 720x414 requires a memory transfer rate of at least 320 Mbps. One of the critical issues was the memory pointer alignment to meet SSE2 optimization (Streaming SIMD Extensions 2), in addition to shared memory allocating between Java and C.

The PR (Packet Reassembler) was implemented based on the IPv4 scheme, but extended to support multiple reassembly slots. The number of slots is a parameter to be configured aiming to balance the tolerance of delay and memory consumption.

RESULTS

We evaluated the implementation considering a single Scribe tree where a new child node joins the P2P network on each 20 seconds. The first child joined 20 seconds after start of broadcasting. The upload bandwidth of the root node was limited to 2 Mbit/s through a traffic shaping software. The root node transmitted a video sequence of duration of 20 seconds in loop until 220 seconds. This video sequence was compressed in blocks using JSVM. Each block contains one entire GOP of 8 frames and was encoded according to the SVC configuration presented in the Table II.

TABLE II. SVC LAYERS OF EACH VIDEO BLOCK

Layer	Resolution (pixels)	FPS	Average bitrate (Kbit/s)	Average PSNR (dB)
0 (base layer)	176x144	24	58	25
1	352x288	24	210	27
2	352x288	24	862	33

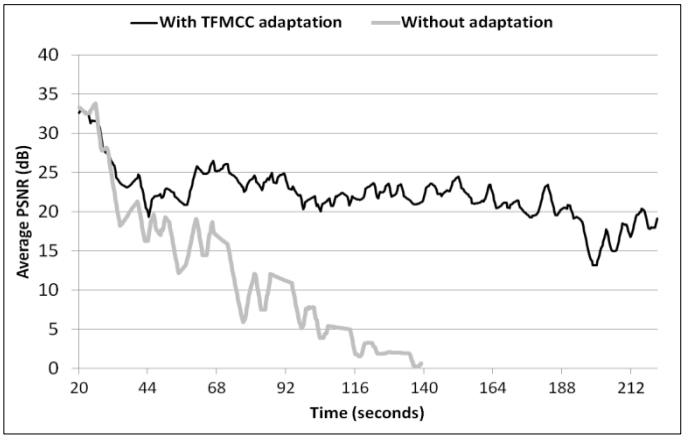


Figure 5. Average quality received by the child nodes

The Figure 5 shows that the adaptation provided by TFMCC, used in conjunction to the properties of scalable coding, leaded to an improvement of PSNR (Peak Signal-to-Noise Ratio), used as a full reference metric for quantifying the objective quality of video. Moreover, we observe that the absence of adaptation leaded to a severe degradation of video quality.

CONCLUSION

This paper presented an architecture for multicast live streaming of scalable video, which provides dynamic adaptation of quality based on the congestion control protocol TFMCC. We proposed the paradigm of independent compressed video block, aiming to reduce de computational cost related to the decoding process the H.264 SVC, since each node of the P2P network must also deal with reassembling, adaptation and fragmentation before commuting the video block. Some of the advantages are: parallel decoding; trivial time stamping of the decoded pictures; no need to process headers of NAL units; error-free decoding, since each block is a full and valid video coded sequence. The mentioned architecture and the proposed paradigm were implemented in a practical framework which has been tested in different scenarios.

FUTURE WORK

A possibility of improving the presented architecture is to perform content aware fragmentation, which means the capacity of treating the boundaries of start and end of NAL units. In this approach, the fragment must implement mechanisms to avoid large variations in the size of the fragment, due to TFMCC requirement. Moreover, implementation may consider prioritized transmission of NAL units according the dependency order of SVC layers. A third idea is related to the reassembling of packet. In the current proposal, lost fragments imply total loss of the block. However,

a more robust receiver could handle corrupted blocks. This solution may require additional headers in the encapsulation of the block, in addition to specific features in SVC decoder software.

REFERENCES

[1] Camarillo, Gonzalo. "Peer-to-peer (P2P) architecture: definition, taxonomies, examples, and applicability". RFC 5694. Nov. 2009.

[2] Liu, Y., Guo, Y. and Liang, C. "A survey on peer-to-peer video streaming systems". Peer-to-Peer Networking and Applications. Jan. Jan, 2008, Vol. 1.

[3] Sentinelli, A., Celetto, L., Lefol, D., Palazzi, C., et al., "Survey on P2P overlay streaming clients". 2009.

[4] Peltotalo, J., Harju, J., Jantunen, A., Saukko, M., Vaatamoinen, L., et al. "Peer-to-peer streaming technology survey". Seventh International Conference on Networking, Cancun, Mexico. Apr. 2008, pp. 342-350.

[5] ITU-T. H.264. "Advanced video coding for generic audiovisual". 2009.

[6] Widmer, Joerg and Handley, Mark. "TCP-Friendly Multicast Congestion Control (TFMCC)". RFC 4654. Aug. 2006.

[7] Casner, S., Frederick, R. and Jacobson, V. "A transport protocol for real-time applications". RTP 3550. Jul. 2003.

[8] E. Richardson, Iain. "The H.264 Advanced Video Compression Standard". 2 ed. s.l. : Wiley, 2010.

[9] Liao, X., Hai, J., Liu, Y., Ni, L., M., Deng, D. "Anysee: peer-to-peer live streaming". Proc. of the 25th IEEE INFOCOM, Barcelona, Spain. Apr. 2006.

[10] Hei, X., Liang, C., Liang, J., Liu, Y., Ross, K.W. "A measurement study of a large-scale P2P IPTV system". IEEE Transactions on Multimedia, no. 8. Dec. 2007, Vol. 9.

[11] Zhang, X., Liu, J., Li, B., Yum, T., "Coolstreaming/DONet: a data-driven overlay network for efficient live media streaming". Proc. of the 24th IEEE INFOCOM, Miami, FL, USA. Mar. 2005.

[12] Castro, M., Druschel, P., Kermarre, A., Nandi, A., Rowstron, A., Singh, A., "Splitstream: high bandwidth multicast in cooperative environments". SOSP 2003, Bolton Landing, NY, USA. Oct. 2003.

[13] Chu, Y.-H., Rao, S. G. and Zhang, H. "A case for end system multicast". Proc. of ACM Sigmetrics. Jun. 2000, pp. 1–12.

[14] Pianese, F., Perino, D., Keller, J., Biersack, E.W. "PULSE: an adaptive, incentive-based, unstructured P2P live streaming system". IEEE TRANSACTIONS ON MULTIMEDIA. Nov. 2006.

[15] Rowstron, A. and Druschel, P. "Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems". IFIP/ACM Middleware. Nov. 2001. Microsoft Research.

[16] Rowstron, A., Kermarrec, A., Castro, M., Druschel, P. "SCRIBE: the design of a large-scale event notification infrastructure". Proc. of 3rd International Workshop on Networked Group Communication (NGC2001), UCL, London, UK. Nov. 2001. Microsoft Research.

[17] Castro M., Druschel Peter, Kermarrec Anne-Marie and Rowstron A. "SCRIBE: a large-scale and decentralized application-level multicast infrastructure". IEEE Journal on Selected Areas in Communications 20 (8). Oct. 2002.

[18] Tsao, Shih-Chiang, Lai, Yuan-Cheng and Lin, Ying-Dar. "Taxonomy and evaluation of TCP-Friendly congestion-control schemes on fairness, aggressiveness and responsiveness". IEEE. Dec. 2007.

[19] Wang, Q., Long, K., Cheng, S., Zhang, R. "TCP-Friendly congestion control schemes in the internet". International Conference on Info-tech and Info-net Conference. 2001.

[20] Wenger, S., Wang, Y.-K. and Schierl, T. "RTP payload format for SVC vídeo". RFC draft. Mar. 2009.