Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Informática

Infraestrutura para Transmissão de Conteúdo Multimídia com Suporte à Adaptação de Fluxos

Ivo Augusto Andrade Rocha Calado

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Angelo Perkusich (Orientador)
Hyggo Oliveira de Almeida (Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

C141i
2010 Calado, Ivo Augusto Andrade Rocha

Infraestrutura para transmissão de conteúdo multimídia com suporte à adaptação de fluxos / Ivo Augusto Andrade Rocha Calado. - Campina Grande, 2010.

99f. : il.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.

Referências.

Orientadores : Profº. Angelo Perkusich, Profº. Hyggo Oliveira de Almeida.

1. Multimídia. 2. Fluxos - Adaptação. 3. Redes de Computadores. 4. Engenharia de Software. 5. Ciência da Computação. I. Título.

CDU 004.032.6(043)

Resumo

A disseminação dos diferentes tipos de aplicações multimídia tem possibilitado uma maior interatividade entre usuários nas mais diversas localizações. Contudo, a falta de mecanismos que garantam níveis de qualidade de serviço, no núcleo da Internet, obriga tais aplicações a realizarem um contínuo processo de adaptação dos fluxos multimídia, de forma a atender aos níveis de transmissão disponíveis, tornando esta uma tarefa adicional no desenvolvimento de tais aplicações.

Por outro lado, diante da grande heterogeneidade existente tanto em termos de recursos de transmissão disponíveis quanto de requisitos dos terminais envolvidos, a implementação de uma única abordagem de adaptação que dê um suporte satisfatório a todos os cenários mostra-se muitas vezes inviável, fazendo-se necessário a utilização de diferentes abordagens, de acordo com o contexto de utilização. Como consequência tem-se um aumento na complexidade do desenvolvimento.

Neste trabalho apresenta-se uma infraestrutura para o desenvolvimento de aplicações multimídia com suporte à adaptação de fluxos. A partir desta infraestrutura, é possível obter uma flexibilização em como o processo de adaptação de fluxos multimídia será realizado, atendendo aos requisitos dos diferentes tipos de aplicações. Neste sentido, de forma a validar a infraestrutura proposta, foi realizado o desenvolvimento de dois estudos de caso visando demonstrar tanto o suporte oferecido ao desenvolvimento de aplicações multimídia quanto à extensibilidade da infraestrutura a partir da inclusão de novos mecanismos de adaptação.

Abstract

The spread of different kinds of multimedia applications has enabled a powerful interaction between users in diverse locations. However, the best effort service nature of the Internet requires a continuous process of stream adaptation, in order to fetch such differents scenarios of transmission. As a consequence, this additional task should be met in the multimedia application development.

On the other hand, according to the great heterogeneity of both transmission resources and terminal requirements, it is impracticable to define a unique adaptation method which be optimal for all scenarios. Therefore, we have an increasing complexity of multimedia application development.

This work presents an infrastructure to multimedia application development to support stream adaptation. It provides a flexible way to implement and to use adaptation methods, meeting the requirements of different applications. Within this context, in order to validate the developed work, it was implemented two case studies: how to develop multimedia applications with support of stream adaptation and present the extensibility provided by proposed infrastructure through the dynamic addition of new adaptation methods.

Agradecimentos

Em primeiro lugar, agradeço a Deus a conclusão desta fase da minha vida, pois se consegui chegar até esse ponto e obter tantos frutos foi unicamente porque Ele esteve ao meu lado.

Agradeço aos meus pais por me mostrar que a verdadeira sabedoria está muito além do ensinado no ambiente acadêmico. A cada dia que passo longe de vocês, percebo o quão valiosa foi a convivência que vocês me proporcionaram. Não tenho palavras para descrever o quanto sou agradecido pelo incodicional amor e apoio de vocês. Tenho o maior orgulho do mundo de ter pais como vocês.

À minha irmã Erika, dedico este trabalho. Que nesta nova fase de sua vida, ela consiga realizar todos seus sonhos, construindo sua família. Estou torcendo de todo meu coração por isso. Dedico também este trabalho ao meu sobrinho Icaro que está para nascer, provavelmente no dia da apresentação deste trabalho. Que ele possa vir ao mundo cheio de saúde e que se ele herdar apenas metade da força e coragem da mãe, já será uma pessoa incrível.

Dedico este trabalho a minha melhor amiga, a minha noiva e futura esposa Lilian Karolline ("mozinho"), por me acompanhar durante esta jornada. Karol, saiba que te admiro de todo meu coração. Que estar construindo um presente e um futuro ao teu lado é a melhor coisa que eu poderia almejar. Obrigado por todo o seu apoio, por sua compreensão pelas minhas ausências nessas idas e vindas entre Maceió-Campina Grande e acima de tudo, por muitas vezes acreditar mais em mim do que eu mesmo.

Agradeço aos meus amigos, tanto os novos que fiz em Campina Grande quanto os antigos: Thiago Bruno, MF, Pablo, Glauber, Fred Bublitz e o restante do pessoal do Laboratório Embedded e da Signove. Graças a eles, esse momento merece ser lembrado.

Agradeço aos alunos de graduação com quem tive oportunidade de trabalhar nos projetos WebEffort (Vinicius Cavalcanti e Romeryto) e Ford (Vicente, Maurílio, Gustavo, Brauner, João Victor e Danilo). Além disso, sou profundamente grato aos alunos do projeto ePhone, Heverton e Erivaldo, por seus apoios ao meu trabalho, sobretudo nos momentos de "filantropia".

Agradeço aos meu orientadores Hyggo e Angelo por me auxiliarem neste trabalho, por me darem valiosas orientações sobre o desenvolvimento do mesmo e por me darem a oportunidade de trabalhar em um ambiente como o Embedded. Juntamente com eles, agradeço

a Leandro Sales, que me ajudou muito, através de conselhos e discussões. Além disso agradeço a Gerrit Renker, por ter contrbuído através de discussões sobre o desenvolvimento do protocolo DCCP.

Por fim, agradeço a Capes pelo apoio financeiro.

Conteúdo

Lista de Símbolos

3G - *3rd Generation (wireless)*

API - *Application Programming Interface*

ACK - *Acknowledgement*

ADSL - *Asymmetric Digital Subscriber Line*

API - *Application Programming Interface*

ATM - *Asynchronous Transfer Mode*

DCCP - *Datagram Congestion Control Protocol*

DNS - *Domain Name System*

ECN - *Explicit Congestion Notification*

FTP - *File Tranfer Protocol*

IPTV - *TV over IP*

IETF - *Internet Engineering Task Force*

MSS - *Maximum Segment Size*

QoS - *Quality of Services*

RFC - *Request For Comments*

RTP - *Real Time Protocol*

RTT - *Round Trip Time*

SS - *Spatial Scalability*

TCP - *Transmission Control Protocol*

TS - *Temporal Scalability*

UDP - *User Datagram Protocol*

VoIP - *Voice over IP*

WiFi - *Rede sem-fio padrão 802.11a/b/g*

WiMax - *Rede sem-fio padrão 802.16*

WWW - *World Wide Web*

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

Desde a primeira comunicação em rede da ARPAnet realizada em 1969 até os dias atuais, o processo de comunicação utilizando redes de computadores sofreu profundas alterações. Muitas foram as alterações sofridas e a evolução ocorrida tanto do ponto de vista de software quanto de hardware e até mesmo de propósito de utilização. Se por um lado a ARPAnet e a MILNET tinham como objetivos a transmissão de informação de pesquisa e militar, respectivamente, hoje a Internet serve aos mais diversos propósitos: comercial, saúde, educação, militar etc [60].

No início, devido às limitações tanto de infraestrutura de rede quanto dos nós finais, a comunicação era realizada basicamente pela troca de mensagens textuais através de e-mails. Nos dias atuais há uma diversidade muito grande de possibilidade de comunicação, como a Web, FTP, aplicações multimídia, entre outras. Neste contexto, as aplicações multimídia, em especial, mostram-se como um importante foco de pesquisa, diante de todos os requisitos necessários para sua disponibilização e de suas possibilidades de aplicação.

Nos últimos anos tem-se observado um grande crescimento na disponibilização de aplicações multimídia utilizando redes de comunicação, como Internet e redes locais. Aplicações de Voz sobre IP (VoIP), TV sobre IP (IPTV), vídeo e áudio sob demanda, videoconferência, jogos *online*, entre outras, têm possibilitado que usuários nas mais diversas localizações tenham uma interação mais rica [61] visto que agora o usuário pode além de ler uma mensagem enviada pelo outro usuário, como no caso do e-mail, ver e conversar com seu parceiro. Além disso, a grande disseminação de dispositivos móveis capazes de prover um alto grau de interatividade através de infraestrutura de acesso sem-fio, como WiFi, WiMax, 3G, tende a

aumentar o número de nós tanto de recepção quanto de geração de dados multimídia [31].

Aplicações multimídia apresentam requisitos diferentes das aplicações mais comuns na Internet. Aplicações como a Web, transferência de arquivos e e-mail são caracterizadas por serem tolerantes a atrasos. Uma página Web pode sofrer atrasos de entrega na ordem de segundos, sem no entanto prejudicar substancialmente a interação com o usuário [41]. Sendo assim, estas aplicações têm como principal requisito a garantia de entrega dos dados, o que torna sua disponibilização mais simples usando apenas mecanismos de retransmissões e controle de congestionamento. Para isto, protocolos da camada de transporte como o *Transport Control Protocol* (TCP) [5; 2] e suas variações [10; 22; 26] satisfazem os requisitos destas aplicações, sendo desnecessária qualquer forma de controle a nível de aplicação sobre a ordem ou a taxa de envio dos dados.

Aplicações multimídia em geral não toleram atrasos, pois estes tendem a dificultar ou até mesmo impedir a comunicação entre os usuários envolvidos na transmissão. Como exemplo, considere aplicações que efetuam comunicação de áudio, como VoIP ou até mesmo uma chamada telefônica comum. O ser humano tolera atrasos de no máximo 150 *ms* sem prejudicar o entendimento do que foi dito [11]. Em vista disto, uma aplicação que pretenda possibilitar uma conversa VoIP, por exemplo, tem que atender ao requisito de atraso máximo para que possa ser bem sucedida.

Um outro aspecto inerente a uma transmissão multimídia que a difere de uma transmissão de dados comum é sua tolerância a perdas de dados ou mesmo alteração do conteúdo transmitido. Uma transmissão multimídia pode ser bem sucedida mesmo diante de perdas de parte da informação transmitida. Dependendo da quantidade de dados perdidos, mesmo que não haja qualquer processo de retransmissão, o usuário pode estar apto a compreender a informação transmitida, percebendo pouca ou nenhuma redução na qualidade da apresentação. Por outro lado, as aplicações multimídia tornam possível a alteração do conteúdo transmitido sem contudo mudar o sentido da informação. Por exemplo, suponha que o usuário deseja assistir a um determinado vídeo com alta qualidade em algum servidor de fluxo de vídeo (como o YouTube [1]). Diante de uma situação de congestionamento na rede, o servidor poderia adotar uma tática de transcodificação e passar a transmitir o fluxo em uma qualidade menor, porém consumindo menos recursos da rede. Além disso, mesmo que ocorra perda

[1] http://www.youtube.com/

de informação durante o tráfego, o fluxo pode ser executado, apresentando ao usuário apenas uma queda na qualidade de apresentação da transmissão, dependendo da quantidade de dados perdidos e também do método utilizado para codificação dos dados.

Diante do exposto, muito se tem pesquisado em como tornar mais eficientes as transmissões multimídia na Internet, ou seja, conseguir efetuar transmissões com a melhor qualidade possível em um ambiente onde não há controle sobre os recursos disponíveis para transmissão. Tais pesquisas tendem a englobar fatores tanto a nível de transporte das informações quanto de codificação nas transmissões multimídia.

No nível de transporte, em geral, busca-se atender aos requisitos de como os dados serão entregues. Para tal, como dito anteriormente, requisitos como: *i)* sensibilidade ao tempo de entrega, *ii)* confiança de entrega e *iii)* controle da quantidade de informação injetada na rede devem ser levadas em consideração.

Dentro desse contexto, protocolos como o *User Datagram Protocol* (UDP) [55] e *Datagram Congestion Control Protocol* (DCCP) [39] têm sido bastante empregados para realização de transmissões multimídia. O protocolo UDP, durante muitos anos, foi utilizado como o protocolo padrão para transmissão de conteúdo multimídia. Este protocolo provê o serviço de multiplexação de portas e verificação de erros em datagramas, sem realizar qualquer processo de retransmissão de dados perdidos. No entanto, ele não implementa mecanismos de controle de fluxo, obrigando o desenvolvedor a implementar tal mecanismo a nível de aplicação, o que é uma tarefa onerosa para o desenvolvedor.

Por outro lado, o protocolo DCCP foi projetado especificamente para a transmissão de aplicações multimídia. Trata-se de um protocolo que não oferece retransmissão de informações perdidas, assim como o UDP, mas que permite o controle de fluxo de transmissão. O protocolo DCCP possibilita o acoplamento de diferentes algoritmos de controle de congestionamento [20; 17; 19], de forma que possa ser utilizado para diferentes classes de fluxos de dados.

No nível de codificação dos fluxos multimídia, diferentes técnicas e padrões têm sido propostos para a criação de fluxos que aliem melhor qualidade da informação gerada à menor quantidade de dados. Técnicas como *Lossless Compression*, *Lossy Compression*, entre outras, fazem parte do universo de codificação, provendo uma gama de possibilidades para codificação do fluxo multimídia [65]. É neste contexto de adaptação de fluxo multimídia que

se insere o presente trabalho.

1.1 Problemática

Uma das características primordiais do projeto da Internet é o serviço de entrega de dados de melhor esforço (do inglês, *Best Effort Service*). O serviço de melhor esforço é caracterizado por ser um serviço de comunicação entre redes onde não há garantia de entrega de dados, nem tão pouco, tempo de entrega das informações. Sendo assim, de maneira mais precisa, pode-se dizer que um serviço de melhor esforço não oferece nenhum tipo de garantia de Qualidade de Serviço (QoS) [75]. Tal propriedade apresenta tanto pontos positivos quanto negativos para qualquer sistema de comunicação. Como ponto positivo, pode-se apontar que o fato da Internet prover apenas um serviço de melhor esforço possibilita uma rápida expansão e disponibilidade de recursos para um grande número de usuários, visto que nenhuma reserva de recursos é realizada, como acontece nas redes ATM ou PTSN [41]. Deste modo, nenhum usuário terá sua solicitação de transmissão rejeitada *a priori* mesmo que a rede esteja com um alto grau de utilização.

Por outro lado, o fato de ser possível haver sobrecarga nos nós de transmissão traz à Internet um revés, que é o congestionamento e, como consequência, atraso e perda de informações. Em vista disso, sistemas de comunicação que efetuam transmissão pela Internet devem ser capazes de lidar com tais restrições [75].

Diante do contexto apresentado, prover adaptação de fluxos tendo como foco os nós finais para aplicações multimídia em tempo real torna-se uma tarefa complexa, visto que deve-se levar em consideração os seguintes fatores [61]:

- Manter informação sobre o estado atual da rede e, em caso de congestionamento, ser capaz de selecionar quais informações são mais importantes e que devem ter prioridade no envio;
- Implementar mecanismos para definir qual o nível de qualidade ideal do fluxo multimídia dados os recursos de rede disponíveis;
- Levar em consideração as características particulares de cada fluxo multimídia, para determinar qual política de adaptação é mais adequada.

A partir das premissas apresentadas acima, diversas propostas têm sido apresentadas para o processo de adaptação multimídia, tendo como foco tanto a camada de transporte quanto a de aplicação.

No primeiro caso, objetiva-se através da camada de transporte obter uma melhor utilização dos recursos da rede. Isto pode ser efetuado de diversas maneiras, como *i)* realizar retransmissão seletiva de pacotes [34] ou *ii)* selecionar qual pacote deve ser enviado dado o tempo de execução [47].

Apesar do DCCP representar um grande avanço para aplicações multimídia [75], se comparado aos protocolos TCP e UDP, ele não efetua nenhum tipo de diferenciação entre pacotes de dados num mesmo fluxo, impossibilitando possíveis descartes de pacotes. Com isso, pacotes de dados multimídia que possuem um tempo de expiração para a sua execução são enviados ainda que seu tempo de apresentação tenha encerrado, visto que o algoritmo de controle de congestionamento pode impor atrasos ao envio dos dados de forma a evitar um congestionamento na rede.

No nível de camada de aplicação, diversas propostas têm sido apresentadas [13; 58; 50] como forma de alterar o fluxo multimídia diante de restrições da rede. Tais métodos de adaptação abrangem a escolha do algoritmo de codificação mais apropriado de acordo com a largura de banda disponível da rede e até mesmo alterando alguns parâmetros dentro do próprio método de codificação. Para obter tais informações e efetuar negociação de formatos de codificação, podem ser utilizados protocolos como RTP [63], XMPP [53], SIP [29], entre outros.

Independente de qual abordagem seja utilizada, cabe ao desenvolvedor da aplicação agregar inteligência à aplicação final com o intuito de garantir que os recursos de rede disponíveis serão utilizados da melhor maneira possível, isto é, entregar o fluxo multimídia com a melhor qualidade possível, sem sobrecarregar a rede.

Outro aspecto que deve ser salientado nas abordagens apresentadas é que não há uma separação clara entre o método de adaptação que será aplicado a um determinado fluxo multimídia e as políticas de adaptação que serão utilizadas. Deste modo, não é possível definir estratégias de adaptação diferentes para uma mesma classe de fluxo. Por exemplo, em uma transmissão de vídeo, dependendo do contexto, para tornar melhor a compreensão, pode ser mais interessante obter um vídeo de melhor qualidade. Em outras situações, o áudio poderia

ser considerado como a informação mais valiosa. Consequentemente, haverá a necessidade da utilização de diferentes políticas no exemplo supracitado.

A partir dos argumentos apresentados, é possível distinguir diferentes atores envolvidos no processo de desenvolvimento de uma aplicação multimídia que efetue adaptação de fluxos como descrito abaixo:

- desenvolvedor dos métodos de adaptação: responsável pela criação de um método de adaptação específico. Um exemplo seria um novo processo de adaptação que poderia ser aplicado a um *codec* de vídeo;
- desenvolvedor de políticas de adaptação: responsável por definir a forma como os métodos de adaptação devem ser utilizados dadas as diversas situações do meio de transmissão e as características particulares de cada classe de fluxo;
- desenvolvedor da aplicação multimídia: é responsável pelo desenvolvimento da aplicação final. Faz uso dos métodos de adaptação e dos perfis de adaptação providos.

Diante da problemática discutida nesta seção, percebe-se a falta de infraestruturas que possibilitem o suporte aos papéis supracitados, tornando, portanto, o processo de desenvolvimento de aplicações multimídia uma atividade complexa.

1.2 Objetivos do Trabalho

Neste trabalho, tem-se como objetivo o desenvolvimento de uma infraestrutura para transmissão de conteúdo multimídia em tempo real com suporte à adaptação de fluxo voltado para comunicação *unicast*. Tal infraestrutura oferece suporte à definição de políticas de adaptação, possibilitando a personalização da forma como a adaptação do fluxo é realizada, com o intuito de torná-la flexível o bastante para a utilização em diferentes classes de aplicações multimídia. Além disso, utiliza-se uma abordagem baseada em *plugins* de forma a permitir a adição de novos mecanismos de adaptação de maneira simplificada.

Mais especificamente, a infraestrutura possui as seguintes características:

- Alteração dinâmica das características dos fluxos de dados de acordo com estado da rede;

- Possibilidade de acoplamento de plugins de adaptação tanto a nível de transporte quanto de aplicação;
- Suporte à definição de políticas de adaptação de fluxos;
- Criação de uma camada de abstração entre a aplicação final e a infraestrutura de rede.

Como objetivo secundário deste trabalho, contribui-se com ferramentas e aplicações de código aberto voltadas para a realização de transmissões multimídia. De maneira mais específica, objetivou-se contribuir no desenvolvimento e disseminação do protocolo da camada de transporte DCCP. Este objetivo foi concretizado dentro do contexto do projeto E-Phone[2], através de *i)* implantação de novos recursos diretamente na implementação do DCCP no núcleo do sistema operacional Linux e *ii)* alteração de aplicações e APIs [3] para transmissões multimídia de forma a habilitar a utilização do protocolo DCCP.

Com o intuito de validar o trabalho desenvolvido, foram realizados dois estudos de casos. Inicialmente, buscou-se validar a capacidade de extensibilidade da infraestrutura através do desenvolvimento de novos *plugins* de adaptação. Para tal, foram desenvolvidos *plugins* de adaptação para os pontos de extensibilidade propostos, isto é, transporte, codificação e negociação. Após isso, buscou-se demonstrar o processo de desenvolvimento de aplicações finais. Para tal, foi desenvolvida uma aplicação para transmissão de áudio que faz uso da infraestrutura proposta.

1.3 Relevância

Nos dias atuais, o desenvolvedor de aplicações multimídia necessita ter ciência de um conjunto de fatores que vão além do escopo principal da aplicação. Por exemplo, o desenvolvedor de uma aplicação de *m-learning* [71] que utiliza recursos como áudio e vídeo deveria ter como principal esforço o desenvolvimento da lógica de negócio do sistema em si. Contudo, diante do estado da arte atual, ele precisa gerenciar alguns requisitos que vão além do objetivo principal da aplicação, como:

- Decidir qual codificação de vídeo e áudio são mais adequadas aos seus propósitos;

[2]Projeto E-Phone: http://garage.maemo.org/projects/ephone

[3]*Application Programming Interface*

- Dado o item anterior, efetuar decisões sobre como alterar as características dos fluxos enviado diante de uma diminuição ou aumento persistente da largura de banda disponível;

- Escolher qual protocolo da camada de transporte é mais adequado;

- Em situações em que existem mais de um algoritmo de controle de congestionamento disponível, determinar qual o mais apropriado baseando-se nas características do fluxo.

Desta forma, faz-se necessário o desenvolvimento de algum mecanismo que possibilite a abstração de vários detalhes envolvidos, deste modo, tornando o desenvolvimento de aplicações multimídia menos complexo do ponto de vista do desenvolvedor final.

O desenvolvimento deste trabalho de dissertação de mestrado apresenta relevância à área de redes multimídia em tempo real, pois reduz os esforços no desenvolvimento de aplicações multimídia e oferece uma infraestrutura extensível para novas fomas de adaptação.

Além disso, esta dissertação está inserida dentro do projeto Percomp, dando continuidade aos investimentos na área de multimídia do Laboratório de Sistemas Embarcados e Computação Pervasiva–Embedded, da Universidade Federal de Campina Grande.

1.4 Estrutura da Dissertação

O presente trabalho está organizado da seguinte forma:

- No Capítulo 2 é apresentada a fundamentação teórica relacionada a este trabalho. São discutidos aspectos relacionados ao desenvolvimento de aplicações multimídia. De maneira específica, são discutidos conceitos às camadas de aplicação e transporte, do modelo TCP/IP e seu envolvimento com as aplicações multimídia. Além disso, serão abordados conceitos relacionados ao processo de adaptação de fluxos multimídia.

- No Capítulo 3 são apresentados alguns trabalhos relacionados à solução proposta, sendo discutidos tanto os pontos positivos quanto negativos de suas soluções. No capítulo seguinte é apresentada a infraestrutura para transmissão de conteúdo multimídia com suporte a adaptação de fluxos. Inicialmente, é apresentada uma visão geral a respeito das principais características da infraestrutura, sendo discutidos o modelo

básico que levou ao desenvolvimento da arquitetura. A seguir, discute-se a respeito da arquitetura propriamente dita, sendo apresentado os módulos existentes e funcionamento da infraestrutura como um todo;

- No Capítulo 5 é apresentada a validação do trabalho desenvolvido, através da apresentação de um conjunto de estudos de caso desenvolvidos, visando demonstrar tanto a capacidade extensibilidade da infraestrutura quanto mostrar como aplicações multimídia podem fazer uso da infraestrutura desenvolvida de modo a prover transmissão e adaptação multimídia de maneira simples;

- No Capítulo 6 são apresentadas as considerações finais e os trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo é apresentado o embasamento teórico dos diversos conceitos apresentados neste trabalho. Neste capítulo será apresentada uma breve descrição a respeito das características gerais das aplicações multimídia, suas características, restrições etc. Além disto, serão abordados conceitos relacionados a infraestrutura de transmissão, em especial à camada de transporte e como se relacionam com as aplicações multimídia. Para finalizar o capítulo, será realizada uma discussão sobre métodos de adaptação de fluxos multimídia existentes.

2.1 Aplicações Multimídia

As aplicações multimídia têm apresentado um incrível crescimento nas mais diversas áreas, voltadas aos mais diversos propósitos, passando a fazer parte do dia-a-dia da maioria dos usuários assim como a Web e o e-mail. Para se ter uma ideia da difusão deste tipo de dado, atualmente grande parte dos recursos da Internet são utilizados na transferência de conteúdo multimídia através de aplicações de compartilhamento de arquivos, como BitTorrent, eMule, entre outros [61]. No entanto, a utilidade das aplicações multimídia vai muito além de simplesmente assistir a um filme localmente. Um dos grandes atrativos dessa tecnologia é a possibilidade de, por exemplo, realizar uma chamada telefônica ou uma sessão de videoconferência pela Internet ou assistir a um jogo que está sendo transmitido no momento. Ou seja, as aplicações multimídia são capazes de tornar a comunicação, de certa maneira, muito mais próxima do usuário, em vista de sua grande interatividade. No entanto, as transmissões

multimídia possuem muitas características e requisitos que tornam a sua implantação uma tarefa muito mais complexa que a transferência de um arquivo de texto, por exemplo.

As aplicações multimídia, em geral, são caracterizadas por serem aplicações sensíveis ao atraso [41]. Isto é, cada pedaço de informação transmitido possui um tempo limite para chegar ao seu destino e ser utilizado, ou então a informação passa a ser inútil. Este cenário vai totalmente ao encontro de aplicações como transferência de arquivos onde, embora seja interessante que o arquivo seja transmitido da maneira mais rápida quanto possível, não é um requisito crucial para o sucesso da operação. No entanto, esta sensibilidade ao tempo, conhecida como *elasticidade* da transmissão, não atinge todas as aplicações multimídia da mesma forma. Em resumo, a sensibilidade ao atraso será diretamente proporcional à interatividade pretendida para a aplicação. Por exemplo, em uma sessão de videoconferência ou chamada VoIP, onde os usuários desejam se comunicar da melhor maneira possível, sem eventos de falha ou atrasos, há um alto grau de sensibilidade ao tempo. Por outro lado, vídeo e áudio sob demanda possuem restrições de tempo menos severas. Apesar de continuarem a ser sensíveis ao tempo, os efeitos decorrentes de uma latência relativamente alta na entrega dos pacotes são menos expressivos.

Outro fator também relacionado ao tempo de entrega dos pacotes que possui considerável importância no contexto de transmissões multimídia é a variação existente no atraso de entrega dos pacotes, conhecido como *jitter*. O *jitter* é ocasionado pela alteração constante das condições dos roteadores da rede, fazendo com que uns pacotes sejam transmitidos mais rapidamente que outros. Uma consequência disto é a necessidade de adição de um atraso à reprodução que compense o *jitter* existente na transmissão. Tal atraso, dependendo da criticidade de tempo da transmissão, como em uma chamada VoIP, pode tornar a transmissão inviável.

Além da sensibilidade a atrasos, uma outra propriedade diferencia grandemente as aplicações multimídia de aplicações como a Web, que é a tolerância a perda de dados durante a transmissão. Uma aplicação multimídia é capaz de ser bem sucedida mesmo que parte da informação transmitida seja perdida, ocorrendo apenas uma queda na qualidade de apresentação da transmissão dependendo da quantidade de dados perdidos e também do método utilizado para codificação dos dados. Ademais, a utilização de técnicas de recuperação de

erros, como Correção de Erros de Repasse (FEC [1]) [41], Requisição-Resposta Automático (ARQ [2]) [61] e interpolação de dados tendem a minimizar as perdas ocorridas.

Classes de fluxos multimídia

Apesar das aplicações multimídia compartilharem muitas características, elas também possuem algumas diferenças que possibilitam agrupá-las em diferentes categorias, de acordo com a interatividade suportada e como o fluxo gerado é gerado e transportado. Basicamente, existem três classes de aplicações, que são descritas abaixo [41]:

Áudio e vídeo de fluxo contínuo armazenado: nesta abordagem, os clientes fazem a solicitação de arquivos multimídia a servidores. Tais arquivos foram armazenados em um momento anterior à solicitação, de modo que todo o arquivo está acessível ao servidor. Uma analogia a este cenário seria o cliente possuir um arquivo de vídeo localmente e poder fazer acessar qualquer parte do vídeo (avançar, retroceder etc.). No contexto de transmissão de fluxo armazenado, em geral, o cliente está apto a selecionar qual parte do fluxo deseja receber e o servidor se encarrega de realizar a transferência solicitada. Para possibilitar este controle, pelo cliente, sobre a mídia armazenada, os servidores implementam protocolos de controle de acesso à mídia, onde o mais amplamente utilizado é o *Real Time Streaming Protocol*–RTSP [64].

Em decorrência da falta de interatividade existente neste modelo de transmissão, dado que os fluxos foram pré-gravados, os requisitos de temporização tornam-se menos severos, visto que para evitar problemas de falta de pacotes para exibição (em decorrência do atraso na entrega), basta adicionar um *buffer* de execução de modo que a aplicação possa postergar o início da reprodução até que haja uma quantidade de dados bastante armazenados, de modo que a aplicação possa reproduzir o fluxo sem sofrer falhas. Para tal, o tamanho do *buffer* deve ser grande o bastante para suprir os atrasos decorrentes da entrega e o *jitter* médio existente na transmissão.

Áudio e vídeo de fluxo contínuo ao vivo: esta classe de aplicações assemelha-se bastante à transmissão de fluxos contínuos armazenados, diferenciando-se, porém, pelo fato de

[1] Do inglês *Forward Error Control*

[2] Do inglês *Automatic Repeat Request*

que a informação transmitida é gerada durante a transmissão. Como consequência, o usuário é inapto a realizar operações de avanço ou retrocesso [3]. Um exemplo de transmissões que façam uso deste modelo de comunicação seriam as rádios *online*. Neste tipo de transmissão, usuários entram e saem de canais de transmissão recebendo o fluxo transmitido. Da mesma forma que em uma rádio tradicional, o usuário não tem a possibilidade de realizar o avanço da programação.

Em relação ao requisito de temporização, este tipo de transmissão possui a mesma flexibilidade que o fluxo contínuo armazenado. Aplicações clientes podem implementar *buffers* de reprodução e inserir alguns segundos de atraso, antes de começar a reproduzir o fluxo sem prejudicar o expectador. No entanto, existe uma situação onde esse atraso na reprodução pode ser percebido, sendo no mínimo desagradável ao usuário, que é quando há um referencial de transmissão. Suponha o cenário em que um usuário está assistindo a um jogo de futebol via um canal de transmissão de vídeo *online*. Se tal usuário não tiver nenhum referencial de transmissão, a existência de atrasos de reprodução pode não ser sequer percebida pelo usuário. Se por outro lado, houver uma segunda fonte emissora desta transmissão (por exemplo, alguma televisão próxima estiver sintonizada neste mesmo jogo) culminará com a percepção do atraso existente e muitas vezes tornando a continuação da transmissão desmotivadora.

Transmissões multimídia em tempo real: esta classe de transmissões, foco principal deste trabalho, possui a maior exigência no requisito temporização de entrega da informação. Neste tipo de transmissão multimídia, atrasos de reprodução são facilmente perceptíveis, prejudicando e às vezes até impedindo completamente transmissão. Exemplos desta classe de transmissão multimídia seriam as sessões de videoconferência, tanto em sentido único quanto na transmissão em duas vias. Um outro exemplo bastante conhecido, seriam as chamadas VoIP, que apresentam um grande crescimento na utilização nos últimos anos e que são consideradas o futuro das chamadas telefônicas [31]. Para se ter ideia da restrição de tempo presente neste tipo de aplicação, atrasos entre 150 e 400 milissegundos numa transmissão de áudio já podem ser percebidos pelos interlocutores. Caso este atraso ultrapasse os 400 milissegundos, a comunicação

[3] Considerando o fluxo presente no servidor. Caso o cliente faça um armazenamento da informação transmitida, tal recurso poderia ser facilmente implementado.

já fica seriamente comprometida [41], tornando, em algumas situações, impossível o entendimento do que foi dito.

2.1.1 Protocolo para transmissão de fluxo multimídia em tempo real

Assim como em qualquer outro tipo de transferência de dados na Internet, o processo de transmissão ocorre a partir da divisão do fluxo completo (ou da informação disponível, no caso dos fluxos ao vivo), em pequenos pacotes de dados que são enviados pela rede. Deste modo, para possibilitar que o fluxo possa ser remontado e reproduzido corretamente no *host* remoto, respeitando o tempo de reprodução exato do fluxo, faz-se necessária a inclusão de marcas de tempo em cada pacote, de modo a possibilitar que o receptor possa saber quando (em relação ao pacote inicial) deverá reproduzir o pacote corrente. Todavia, marcas de tempo não são disponibilizadas por protocolos da camada de transporte (discutidos na Seção 2.2). Uma das razões para isto é que os protocolos da camada de transporte são projetados para transmissão de fluxos genéricos, que muitas vezes são independentes de tempo e, portanto, não necessitam de tal recurso. Objetivando padronizar a forma como as aplicações fazem a transmissão de fluxos multimídia, a *Internet Engineering Task Force* – IETF padronizou o protocolo para transmissões multimídia em tempo real, RTP.

O protocolo RTP (*Real Time Protocol*) [63] é o protocolo padrão para transmissão de conteúdo multimídia em tempo real na Internet. O principal objetivo do RTP é encapsular os pacotes do fluxo multimídia em uma camada adicional de dados, agregando as informações necessárias para reprodução no lado remoto, como informação do tipo de dado e marcas de tempo. Além disso, o RTP provê um conjunto de serviços que auxiliam o processo de transmissão como um todo, tais como [54]:

- Detecção e correção de perdas [57];
- Identificação da fonte de dado;
- Relatórios da qualidade da transmissão [52];
- Sincronização de fluxos;
- Gerenciamento de sessões.

A partir dos serviços providos pelo RTP, é possível a criação de sessões RTP, onde transmissores e receptores podem ser adicionados[4] e cada fonte de dado pode especificar o tipo de dado que irá transmitir permitindo a correta sincronia nos receptores. Para tal, o RTP utiliza, em cada sessão de transmissão, dois canais de comunicação: o primeiro, dedicado à transmissão dos dados da aplicação, por onde os pacotes de dados são encapsulados e enviados; e um segundo canal, responsável apenas pela passagem de dados de controle da sessão (canal RTCP [63]), por onde são trafegados relatórios a respeito das condições da transmissão, como informações de perda, sincronia de fluxos etc.

Devido à necessidade de dar suporte a uma ampla gama de codificadores de fluxo multimídia existentes e a diferentes classes de aplicações, o RTP foi especificado em uma estrutura em camadas, conhecido como *Application-Level Framing* [54], onde cada classe de aplicação define seu próprio perfil (*Profile*) podendo dar suporte a um ou mais tipos de mídia (*PayloadFormat*) [63]. Exemplos de perfis seriam o "Perfil para Áudio e Vídeo Conferência com Controle Mínimo" [62] e "Perfil para Transmissões em Tempo Real Seguras (SRTP)" [6]. Por outro lado, já foram definidas diversas especificações para formatos de mídia, como os formatos H.264 [76], MPEG-4 [37] para transmissão de vídeo e Speex [30] para áudio.

2.2 Camada de transporte

O principal propósito da camada de transporte na suíte TCP/IP é prover comunicação lógica entre dois processos em *hosts* distintos. Para tal, a camada de transporte faz uso dos serviços da camada de rede, que fornece comunicação entre *hosts*, e adiciona o serviço de multiplexação de processos, através do conceito de portas de comunicação [41] e verificação de erros.

Diferentemente do que acontece às camadas de rede e enlace, a camada de transporte é implementada apenas nos *hosts* finais, visto que do ponto de vista da infraestrutura de rede é desnecessário o conhecimento de para qual processo uma determinada mensagem está direcionada. Sendo assim, o processo de envio de uma mensagem segue basicamente os passos apresentados abaixo:

1. A camada de aplicação envia os dados à camada de transporte;

[4] O RTP oferece suporte tanto a comunicação *unicast* quanto *multicast*

2. A camada de transporte encapsula os dados da camada de aplicação, adicionando informações de identificação de processos (portas de origem e destino) e checagem de erros e envia os dados à camada de rede;

3. Ao chegar na camada de rede, esta adiciona o identificador da origem e destino (endereços IP) e o envia à camada de enlace que por sua vez injeta os dados na rede;

4. Durante a transmissão, os roteadores abrem o dado até o nível de camada de rede, obtendo o endereço de origem e utilizando esta informação para realizar o correto roteamento;

5. Quando o dado chega no *host* destino, a camada de rede extrai a carga útil do datagrama recebido entregando-o à camada de transporte;

6. A camada de transporte, por sua vez, identifica para qual processo aquela mensagem está destinada, a partir do número de porta de destino e entrega a mensagem à aplicação.

Dentre os protocolos da camada de transporte existentes, os protocolos TCP (*Transport Control Protocol*) e UDP (*User Datagram Protocol*) são considerados os principais protocolos da suíte TCP/IP. O primeiro oferecendo transmissão orientada à conexão, transporte confiável e controle de fluxo e congestionamento. Por outro lado, o UDP oferece um serviço de melhor esforço não orientado a conexões. Além destes dois protocolos, no contexto de transmissões multimídia, um terceiro protocolo, chamado DCCP (*Datagram Congestion Control Protol*) vem ganhando uma grande aceitação por ter características que auxiliam no processo de transmissão.

A seguir são descritos os protocolos DCCP, UDP e TCP, apresentando suas características gerais e sua relação com as aplicações multimídia, foco deste trabalho.

2.2.1 Protocolo UDP

O protocolo UDP, do inglês *User Datagram Protocol*, definido sob a RFC 768 [55] é um protocolo da camada de transporte voltado para a transmissão de datagramas. Por se tratar de um protocolo de transporte, o UDP realiza transmissão de dados entre processos de *hosts*

distintos. Contudo, este protocolo não utiliza a abstração de orientação a fluxos, utilizada pelo TCP, para tratar os dados transmitidos. Neste protocolo, cada pacote de dado é tratado semanticamente de maneira isolada, onde cada pacote é chamado de datagrama.

O protocolo UDP pode ser considerado um protocolo bastante simples, visto que não implementa muitas das funcionalidades providas por outros protocolos, como o TCP ou DCCP. Basicamente, o cabeçalho de um datagrama UDP inclui apenas 4 tipos de informações, conforme apresentado na Figura 2.1 e descrito na Tabela 2.1:

Tabela 2.1: **Descrição dos campos do cabeçalho de um datagrama UDP**

Nome do Campo	**Tamanho**	**Descrição**	**Campo Obrigatório**
Porta Origem (*Source Port*)	16 *bits*	Indica a porta de origem do pacote.	Não
Porta Destino (*Dest Port*)	16 *bits*	Indica a porta de destino no pacote.	Sim
Tamanho (*Length*)	16 *bits*	Tamanho total do datagrama UDP, incluindo cabeçalho e carga útil.	Sim
Soma de Verificação (*Checksum*)	16 *bits*	Soma de verificação, utilizada para verificar se o datagrama foi corrompido durante a transmissão. Obs.: existe uma variante do protocolo UDP, conhecida como UDP-Lite [42], onde é possível definir a cobertura do campo de checagem de erros. O objetivo disto é possibilitar que pacotes que possuam que recebidos com erros, sejam aceitos e encaminhados a camada de aplicação. Como discutido na Seção 2.1, as aplicações multimídia são tolerantes a erros e perdas de pacotes. Consequentemente, mesmo que um pacote esteja corrompido ele pode vir a ser útil numa transmissão multimídia.	Sim

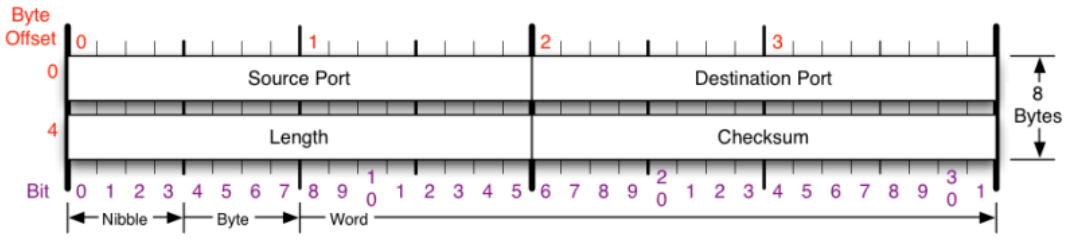


Figura 2.1: Estrutura de um datagrama UDP

Figura adaptada de http://nmap.org/book/images/hdr/MJB-UDP-Header-800x264.png

Abaixo, são apresentadas as principais características do protocolo UDP [41; 75]:

- Provê multiplexação de portas;
- Possibilita tanto transmissão *unicast* quanto *multicast*;
- É um protocolo não orientado à conexão. Sendo assim, nenhum tipo de informação de controle é trocada antes do início da transmissão;
- Não implementa o serviço de garantia de entrega de dados, nem mesmo de notificação sobre as informações perdidas. Além disso, não é anexada no cabeçalho dos datagramas enviados nenhuma informação sobre a sequência de envio dos pacotes. Com isso, o UDP não é capaz de detectar se houve reordenação de pacotes durante a transmissão;
- Não apresenta *buffers* de envio. Ou seja, o dado passado pela aplicação é diretamente enviado;
- Não implementa mecanismos de controle de fluxo. Cada *host* irá fazer o envio do máximo de informação disponível para envio que o adaptador de rede for capaz de processar, independente da capacidade de processamento do *host* remoto;
- Não implementa mecanismos de controle de congestionamento. Consequentemente, mesmo que a rede encontre-se em alto grau de congestionamento o UDP não irá reduzir sua taxa de envio se houver dados disponíveis para envio.

Apesar do UDP oferecer poucos serviços (se comparado ao TCP, por exemplo), o que, a princípio, poderia levar o desenvolvedor a evitar tal protocolo, o UDP apresenta uma sobrecarga de transmissão muito menor que o TCP. Em vista disso, aplicações sensíveis a tempo ou aplicações em que a carga útil envolvida na transmissão seja muito pequena em relação ao cabeçalho TCP podem fazer uso deste protocolo. Um exemplo disto seria o serviço de DNS [48]. O DNS (*Domain Name Service*) é um serviço para tradução de nomes para endereços de rede, utilizado na Internet. Cada solicitação de tradução é composta basicamente pela domínio que se deseja traduzir tendo como resposta o endereço IP correspondente. Em vista disto, utilizar um protocolo como o TCP mostra-se inadequado, devido ao percentual de informação de controle que é utilizada em relação à carga útil.

Utilização do protocolo UDP em Transmissões Multimídia (UDP + RTP)

Outra classe de aplicações que fazem grande uso do protocolo UDP são as aplicações multimídia. Diferentemente do serviço DNS, as aplicações multimídia, em geral, necessitam de uma relevante fatia dos recursos de rede para realizar a transmissão, fazendo uso, portanto, deste protocolo pelas seguintes razões: *i)* necessidade de entrega rápida das informações, *ii)* tolerância perdas, conforme discutido na Seção 2.1 e *iii)* dependendo do tipo da transmissão multimídia, suporte à transmissão *multicast* [5].

Devido às características apresentadas, cabe ao desenvolvedor da aplicação multimídia agregar os mecanismos necessários, a nível de aplicação, de modo a evitar que a transmissão sobrecarregue a rede, através da implementação dos mecanismos para controle do fluxo da informação transmitida. Todavia, como o UDP não provê nenhum serviço para sequenciamento de pacotes e relatório das propriedades correntes da transmissão, em geral utiliza-se o protocolo RTP [63] como uma camada adicional nos pacotes transmitidos.

Devido a tamanho sucesso, o conjunto RTP/UDP possui diversas pilhas de implementação disponíveis [68; 49], possibilitando que desenvolvedores façam uso deste conjunto de protocolos de maneira simples. No entanto, apesar de todos os recursos que o RTP agrega ao UDP para utilização em transmissões multimídia, ele não efetua controle de congestionamento sobre os dados transmitidos, cabendo, portanto, ao desenvolvedor a realização desta tarefa, tal como efetuado em [24].

2.2.2 Protocolo DCCP

O protocolo DCCP (do inglês *Datagram Congestion Control Protocol*), definido sob a RFC 4340 [39] é um protocolo da camada de transporte que provê comunicação *unicast* entre dois *hosts*, controle de congestionamento e entrega não confiável de pacotes. O DCCP é projetado para aplicações que desejam proporcionar um controle de congestionamento, visando a não degradação dos recursos da rede, assim como ocorre com o TCP, mas sem necessitar da garantia de entrega dos dados, semelhante ao UDP.

[5] Conforme discutido anteriormente, o foco deste trabalho está na realização de transmissões multimídia via comunicação *unicast*. Portanto, embora as transmissões *multicast* representem uma importante abordagem para aplicações multimídia, está fora do escopo deste trabalho. Para mais informações a respeito deste tipo de transmissão, recomenda-se a leitura dos trabalhos [7] e [59] como fontes iniciais de pesquisa

As principais motivações por trás do desenvolvimento do protocolo DCCP foram inicialmente definidas no documento *Problem Statement for the Datagram Congestion Control Protocol* [15] e resumem-se em:

- A realização de transmissão de dados sem qualquer mecanismo de controle de congestionamento tende a levar ao colapso da rede. Tal problema torna-se evidente em transmissões UDP em que não se realiza um controle da taxa de envio a nível de aplicação;
- É difícil implementar correta e eficientemente mecanismos de controle de congestionamento no nível de aplicação;
- Não havia uma forma padrão de ter informações sobre o estado atual de uma transmissão em um fluxo não confiável;
- Não havia tratamento para mecanismos de notificação explícita de congestionamento (ECN–*Explicit Congestion Notification*) [56][66] em fluxos não confiáveis;
- A necessidade de um mecanismo de controle de congestionamento TCP-*Friendly*;
- O problema de distribuir aplicações que realizem transmissões UDP na presença de *firewalls*. Muitos administradores de rede, visando evitar uma saturação dos recursos da rede, bloqueiam, em seus *firewalls*, a passagem de fluxos UDP, visto que estes tendem a utilizar todos os recursos disponíveis, degradando consequentemente os demais fluxos;
- A necessidade de poder negociar os parâmetros do controle de congestionamento em fluxos não confiáveis, independentemente do protocolo de sinalização utilizado em nível de aplicação.

Deste modo, o objetivo do DCCP é prover uma forma padronizada de implementar mecanismos de controle de congestionamento e negociação, voltado para aplicações de tempo real. O DCCP realiza estabelecimento, finalização e negociação de propriedades da conexão de maneira confiável e realiza a confirmação dos pacotes recebidos dando suporte a pacotes marcados no campo ECN.

Cada conexão DCCP estabelecida é bidirecional, isto é, é possível o tráfego tanto de pacotes de dados quanto de confirmação de entrega de pacotes (ACKs) em ambas as direções. No entanto, logicamente, é como se existissem sub-fluxos (mais conhecidos como *half-connection*), visto que, em cada direção da transmissão, são passados tanto dados da aplicação quanto de confirmação de pacotes, conforme apresentado na Figura 2.2. Ademais, cada sub-fluxo tem a possibilidade de definir o algoritmo de controle de congestionamento que irá utilizar na transmissão. Ou seja, dois diferentes algoritmos de controle de congestionamento podem ser utilizados na mesma conexão em direções diferentes. Contudo, apesar de virtualmente isolados, os dois sub-fluxos compartilham o mesmo canal de transmissão. Por exemplo, pacotes de confirmação podem ser utilizados para, além de confirmar pacotes recebidos, fazerem envio de novos pacotes de dados, utilizando a técnica conhecida como "*piggyback*" [41; 39].

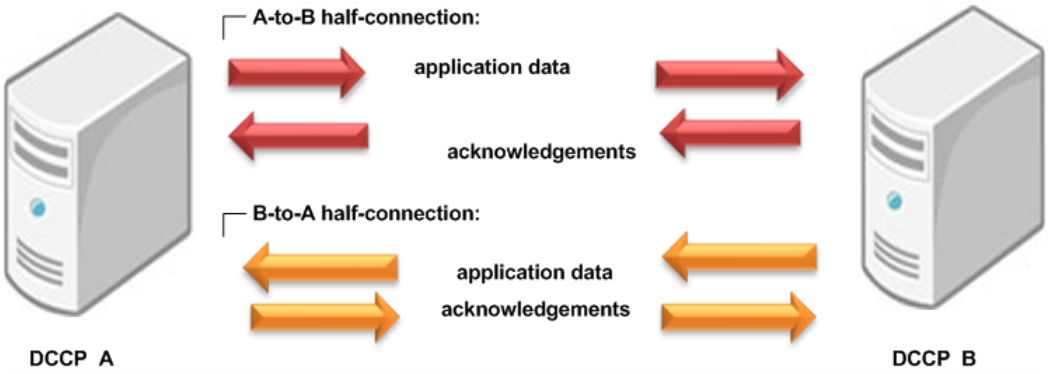


Figura 2.2: Modelo de conexão bidirecional do protocolo DCCP

Ciclo de vida de uma transmissão DCCP

Toda conexão DCCP pode ser dividida em três etapas, que são: estabelecimento de conexão, transferência de dados e término da conexão, conforme apresentado na Figura 2.3. Tal modelo assemelha-se ao realizado pelo protocolo TCP [41]. As três fases são descritas a seguir:

Fase de estabelecimento de conexão: Na fase de estabelecimento da conexão, há o *handshake* entre os dois *hosts* envolvidos. Assim como no TCP, a fase inicial de uma conexão no DCCP é composta por três procedimentos (*3-way handshake*). Um dos *hosts*, chamado "servidor", "escuta" conexões numa determinada porta. Por outro

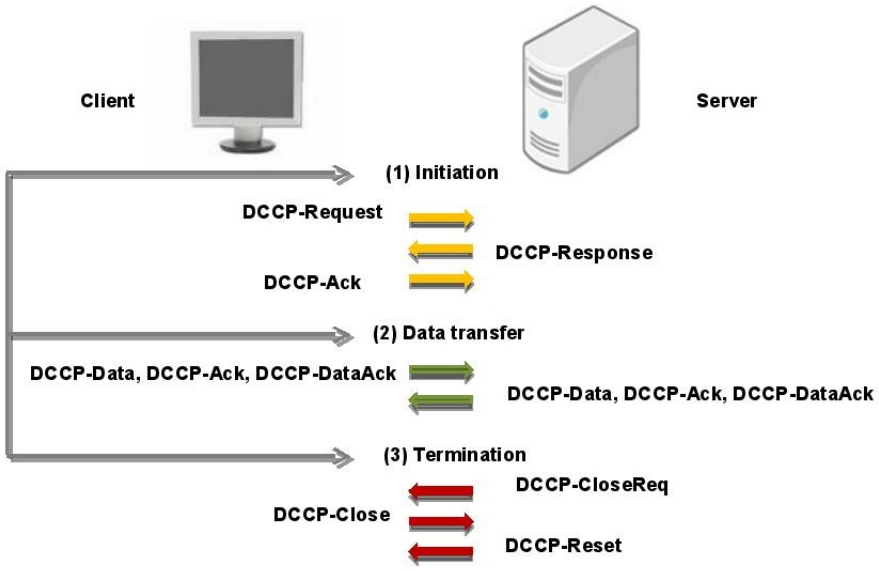


Figura 2.3: Fases de uma conexão DCCP

lado, o *host* denominado "cliente" solicita a abertura de conexão (DCCP-Request) e o servidor, por sua vez, confirma a solicitação (DCCP-Response) e tal mensagem é novamente confirmada pelo cliente (DCCP-Ack). A partir deste ponto, a transmissão encontra-se plenamente estabelecida e a transmissão move-se para a fase de transmissão de dados.

Fase de transmissão de dados: Nesta fase ambos os transmissores estão aptos a enviar e receber dados e pacotes de confirmação de entrega de dados (DCCP-Data, DCCP-Ack e DCCP-DataAck).

Fase de término da conexão: A fase de finalização pode ser realizada em duas ou três etapas, dependendo de qual *host* solicite o término da conexão. Caso seja o cliente a realizar esta tarefa, ele envia uma mensagem de término (DCCP-Close) ao servidor que, por sua vez, responde com um pacote de confirmação de finalização da conexão (DCCP-Reset). Se, por outro lado, o servidor deseja finalizar a conexão, ele irá solicitar ao cliente que finalize a conexão, o qual, por sua vez, efetuará o processo descrito acima.

Para realização de todo o processo de transmissão, o protocolo DCCP estabelece um total de 10 tipos de pacotes, que são descritos na Tabela 2.2:

Tabela 2.2: **Descrição dos diferentes tipos de pacotes DCCP**

Tipo do pacote	**Descrição**
DCCP-Request	Pacote enviado pelo cliente para solicitar abertura de conexão.
DCCP-Response	Pacote enviado pelo servidor como resposta a um pacote DCCP-Request.
DCCP-Ack	Pacote enviado como confirmação de recebimento de pacotes. Este pacote não carrega nenhuma carga útil de informação.
DCCP-DataAck	Pacote enviado como confirmação, mas que carrega carga útil como *piggyback*.
DCCP-Data	Pacote que contém apenas carga útil.
DCCP-CloseReq	Pacote enviado pelo servidor para o cliente para solicitar que este encerre a conexão.
DCCP-Close	Pacote enviado pelo cliente para término da conexão.
DCCP-Reset	Pacote enviado pelo servidor confirmando o término.
DCCP-Sync	Pacote enviado pelos *hosts* para realização de sincronia dos números de sequência após perdas em rajada.
DCCP-SyncAck	Pacote enviado para confirmar o recebimento de pacote do tipo DCCP-SyncAck.

Cabeçalho do pacote DCCP

A estrutura do cabeçalho de um pacote DCCP é dividida em três camadas, sendo um cabeçalho genérico a todos os tipos de pacotes, uma camada de campos específicos para cada tipo de pacote e uma terceira camada de opções globais ao pacote. Na Figura 2.4, apresenta-se a estrutura genérica de um pacote DCCP, onde cada pacote tem um cabeçalho com tamanho variando entre 12 e 1020 *bytes*.

O cabeçalho genérico tem como objetivo agregar as funções que são comuns a todos os pacotes, como indicação da porta de origem e destino dos pacotes, tamanho do pacote em questão. Tal cabeçalho pode assumir duas formas diferentes, conforme apresentado na Figura 2.5 e 2.6. A diferença entre os dois tipos de pacotes está no tamanho do campo de número de sequência, visto que o cabeçalho menor apresenta um tamanho de número de sequência de 24 *bits* enquanto que o maior apresenta o campo de número de sequência com 48 *bits*. O cabeçalho genérico estendido é utilizado em alguns tipos de pacotes de controle ou em pacotes que carregam carga útil e cuja conexão apresenta uma alta taxa de transferência [39].

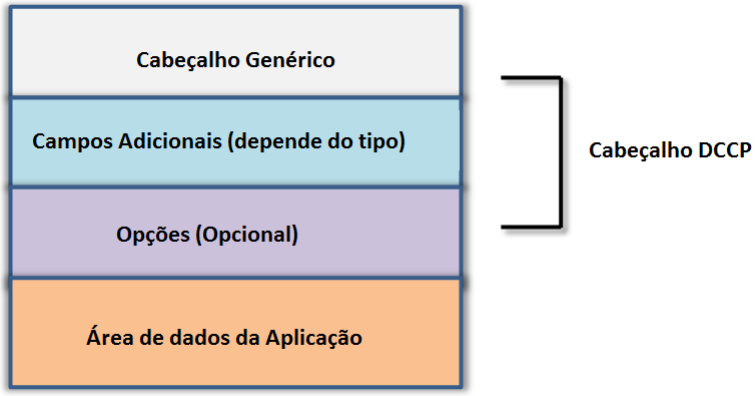


Figura 2.4: Estrutura de um datagrama DCCP

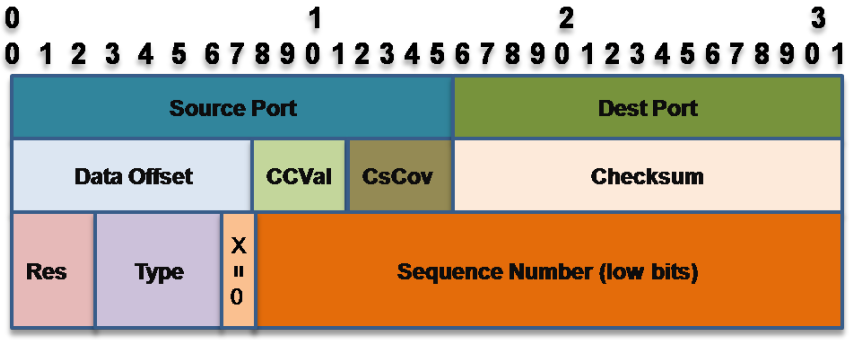


Figura 2.5: Cabeçalho genérico de um pacote DCCP

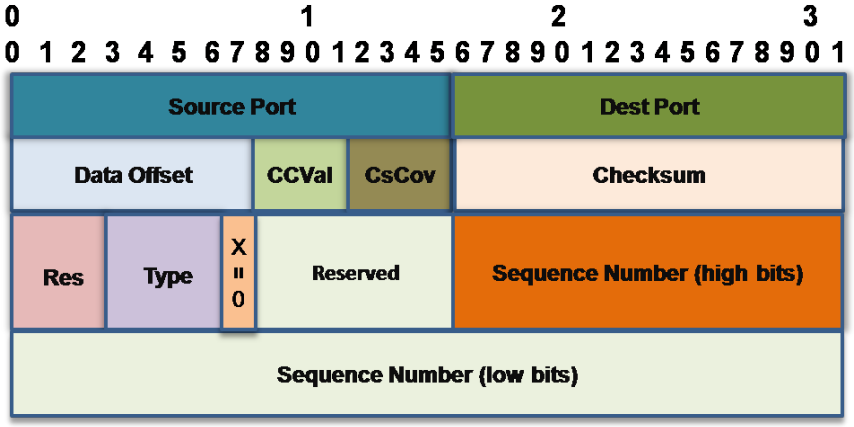


Figura 2.6: Cabeçalho genérico de um pacote DCCP (estendido)

Na Tabela 2.3 é apresentada a descrição dos campos do cabeçalho genérico de um pacote DCCP.

Tabela 2.3: **Descrição dos campos do cabeçalho genérico de um datagrama DCCP**

Nome do Campo	**Tamanho**	**Descrição**	**Campo Obrigatório**
Source Port	16 *bits*	Indica a porta de origem do pacote.	Sim
Dest Port	16 *bits*	Indica a porta de destino no pacote.	Sim
Data Offset	8 *bits*	Especifica o tamanho do cabeçalho.	Sim
CCVal	4 *bits*	Utilizado pelo transmissor no processo de controle de congestionamento.	Sim
CsCov	4 *bits*	Especifica quais partes do pacote DCCP será utilizada para especificar o campo de *Checksum*. Inclui obrigatoriamente o cabeçalho do pacote DCCP e opcionalmente a área de carga útil do pacote. A opção de não incluir o campo de dados no processo de checagem de erros, deve-se ao fato de que alguns tipos de aplicações como decodificadores de fluxos multimídia são capazes de realizar recuperação dos erros e portanto aproveitar os dados corrompidos [43][58].	Sim
Checksum	16 *bits*	Campo utilizado para checagem de erros no pacote DCCP. Pode cobrir todo o pacote ou apenas uma parte dele.	Sim, mas a abrangência deste campo depende do campo **CsCov**
Res	3 *bits*	Campo reservado para uso futuro.	Não
Type	4 *bits*	Especifica o tipo do pacote (DCCP-Request, DCCP-Response, DCCP-Data etc.).	Sim
Extended Sequence Numbers(X)	1 *bit*	Indica se o cabeçalho genérico é do tamanho padrão (x = 0) ou estendido (x = 1).	Sim
Sequence Number	24 ou 48 *bits*	Número de sequência do pacote.	Sim, mas o tamanho do campo depende do valor do campo **X**.

Para maiores informações sobre a estrutura específica de cada tipo de pacote DCCP, consulte a RFC deste protocolo [39].

Controles de Congestionamento

Uma importante característica presente no DCCP é a possibilidade de definir qual mecanismo de controle de congestionamento irá gerenciar o envio de dados em cada sub-fluxo da transmissão, podendo ser definido tanto antes quanto durante a transmissão. O principal ob-

jetivo deste mecanismo é permitir que fluxos com características diferentes possam utilizar o controle de congestionamento mais adequado. Para possibilitar o acoplamento e a utilização dos mecanismos de controle de congestionamento na abordagem supracitada, uma estrutura modular foi projetada de forma a separar o núcleo do protocolo dos mecanismos de controle de congestionamento presentes, possibilitando também que novos mecanismos possam ser adicionados facilmente ao DCCP.

Para permitir selecionar um controle de congestionamento entre diferentes mecanismos, cada mecanismo é referenciado por um identificador, conhecido como CCID (*Congestion Control Identifiers*) [39]. A atribuição destes identificadores é realizada pela IANA (*Internet Assigned Numbers Authority*), onde valores nas faixas de 0 a 1 e 5 a 247 encontram-se disponíveis para utilização.

Atualmente, dois mecanismos de controle de congestionamento estão padronizados e um terceiro encontra-se em etapa experimental. São eles: CCID-2 (*TCP Like*) [17], CCID-3 (*TCP Friendly*) [19] e CCID-4 (*TCP Friendly for Small Packets*) [20]. Tais mecanismos utilizam informações de perdas de pacotes e marcações, via ECN, para determinar congestionamento e, por conseguinte, realizar alteração na taxa de envio. A seguir são apresentadas as principais características dos CCIDs especificados.

CCID-2 (*TCP Like*)

O mecanismo de controle de congestionamento CCID-2 (*Congestion Control ID 2*) [17] utiliza uma abordagem semelhante aos algoritmos de controle de congestionamento do TCP [17], sendo voltado para aplicações que são capazes de se adaptar a rápidas mudanças nas condições de transmissão e que estão aptas a fazer uso de toda largura de banda disponível. A seguir são apresentadas as principais características deste mecanismo de controle de congestionamento:

É um algoritmo baseado em janela de congestionamento: é mantida uma variável que indica o número máximo de pacotes em trânsito permitidos. Esta variável é alterada de acordo com as confirmações e perdas dos pacotes transmitidos;

Realiza esquema de crescimento aditivo e decaimento multiplicativo (AIMD): este mecanismo é semelhante ao TCP Reno, onde, a cada evento de perda, a janela de conges-

tionamento é reduzida à metade;

Suporte a confirmação seletiva de pacotes (SACK): a partir deste recurso, é possível identificar quais pacotes foram perdidos numa série. Utilizando-se, para isso, de um vetor de pacotes de confirmação (*AckVector*);

Utiliza esquema de partida lenta e prevenção de congestionamento: de maneira semelhante ao TCP, divide o período de envio em duas etapas. Na etapa de partida lenta, o remetente começa transmitindo a uma taxa lenta, mas aumenta sua taxa de envio exponencialmente até o primeiro evento de perda, passando então para a fase de prevenção de congestionamento, onde o crescimento da janela de congestionamento passa a ser linear;

Realiza controle de congestionamento reverso e entrega confiável de pacotes Ack: a taxa de envio de pacotes de confirmação é controlada de acordo com as condições da rede.

Exemplos de aplicações multimídia que se adaptam melhor ao modelo de transmissão do CCID-2 são jogos *online* e fluxos de vídeo que fazem uso do *codec* XVID [51].

CCID-3 (*TCP Friendly*)

O CCID-3 [19] apresenta uma abordagem para controle de congestionamento bastante diferente se comparado ao CCID-2 ou aos mecanismos de controle de congestionamento do TCP, utilizando como base o algoritmo de controle de congestionamento TFRC (*TCP Friendly Rate Control*) [27; 16]. Este mecanismo tem por objetivo minimizar mudanças repentinas na taxa de envio em resposta a eventos de perda.

Para estabelecer a taxa máxima de envio, o CCID-3 faz uso de uma equação chamada *TCP Throughput Equation*, apresentada na Equação 2.1.

$$X = \frac{s}{R\sqrt{\frac{2bp}{3}} + 12R\sqrt{\frac{3bp}{8}}p(1+32p^2)} \tag{2.1}$$

onde,

- *X* é a taxa de transmissão em *bytes*/segundos;

- *s* é o tamanho do pacote em *bytes*;
- *R* é o valor do *Round Trip Time*;
- *p* é a taxa de eventos de perdas em relação ao número de pacotes transmitidos;
- *b* é o número de pacotes confirmados por relatório de perdas.

O resultado da utilização desta equação é que as alterações na taxa de envio serão mais suaves se comparadas ao que acontece no CCID-2. Uma outra característica bastante importante deste mecanismo de controle de congestionamento é que o processo de confirmação dos pacotes recebidos não é realizado por pacote, mas sim, por relatórios de *feedback* que engloba vários pacotes.

Diante das características presentes neste mecanismo de controle de congestionamento ele é mais indicado a alguns fluxos de áudio e vídeo, como fluxos MPEG [51], por necessitarem de uma alteração menos repentina da largura de banda disponível para envio.

CCID-4 (*TCP Friendly for Small Packets*)

O CCID-4 [20] utiliza uma variante do algoritmo TFRC, conhecido como TFRC-SP (*TCP Friendly Rate Control for Small Packets*) [18] como mecanismo de controle de taxa de envio. Assim como o CCID-3, este mecanismo de controle de congestionamento foi projetado para aplicações que necessitam de uma alteração suave na taxa de envio de informações. No entanto, o CCID-4 tem como particularidade o fato de ser projetado para ser utilizado por aplicações que efetuem transmissões de pacotes com uma pequena carga útil de informação ou que sejam capazes de alterar a sua taxa de envio a partir da alteração do tamanho do pacote. Aplicações de voz sobre IP (VoIP) tendem a se adaptar bem a este mecanismo de controle de congestionamento, visto que, em geral, os codificadores de áudio utilizados geram pacotes de dados pequenos e, além disso, são capazes de alterar a qualidade do fluxo transmitido, tendo como consequência o aumento e a diminuição do tamanho do pacote [14].

2.2.3 Protocolo TCP

O protocolo TCP foi desenvolvido em meados da década de 80 com o intuito de prover transmissão confiável de dados em links de comunicação que não apresentam garantias. Trata-se

de um protocolo da camada de transporte (modelo TCP/IP) que roda nos hosts finais, isto é, para o núcleo da rede a execução do TCP é transparente. Embora, em tese, o TCP possa ser executado sobre qualquer protocolo da camada de rede, na prática houve uma grande difusão da utilização do TCP em união ao IP, de tal forma que uma das principais suítes de protocolos de rede leva o nome da união destes dois protocolos (TCP/IP). De um lado, o IP provê um serviço de melhor esforço e comunicação entre hosts e do outro o TCP provê garantias de entrega e multiplexação de processos.

O TCP (do inglês *Transmission Control Protocol*) é um protocolo orientado à conexão, definido sob as RFCs 793 [2], 1122 [9], 1323 [36], 2018 [45], 2581 [5] e 3390 [4] que provê comunicação entre processos localizados em *hosts* distintos. Além disso, este protocolo provê comunicação *full-duplex*, permitindo que os dois *hosts* envolvidos na transmissão possam enviar e receber dados simultaneamente, com suporte a controle de fluxo, controle de congestionamento e retransmissão de dados perdidos.

Devido às características do TCP, ele é indicado sobretudo para aplicações elásticas e não tolerantes a perdas, como WWW, SMTP, Telnet, SSH, FTP [41]. Em algumas situações o TCP pode ser utilizado para o transporte de fluxos multimídia. Contudo, os cenários de aplicação são bastante restritos, sendo portanto utilizados quando há uma abundância da largura de banda disponível ou então há uma necessidade específica da aplicação, como permitir que o fluxo atravesse alguns *firewalls* e *gateways*[6] [43]. Para esses tipos de aplicações, em geral, a interatividade provida é bastante reduzida, sendo mais aplicada a simples exibição do fluxo. Um exemplo deste cenário seria o realizado pelo YouTube, onde o usuário solicita um vídeo e aguarda até que uma parte do vídeo seja transferida para que tenha início a exibição.

Entretanto, as aplicações multimídia, de maneira geral, não se adaptam bem as propriedades de transmissão do protocolo TCP, visto que, devido aos atrasos impostos pelos esquemas de retransmissão e reordenação, tais aplicações acabam penalizadas por serem sensíveis ao tempo de entrega dos pacotes.

[6] Alguns administradores de redes, visando impedir uma saturação na utilização da largura de banda, limitam os tipos de fluxos permitidos em seus domínios administrativos.

Estrutura de um pacote TCP

Uma unidade de dados no TCP, chamada segmento TCP é dividida em duas regiões bem específicas, o cabeçalho que carrega informações de controle e uma carga útil que carrega a real informação das aplicações, como uma requisição HTTP, por exemplo.

Na Figura 2.7 é apresentada a estrutura de um segmento TCP, sendo discutidas as funcionalidades de cada campo na Tabela 2.4.

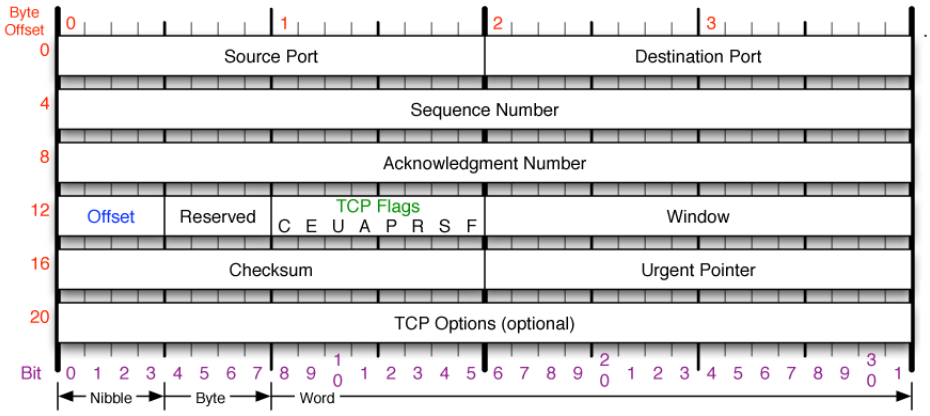


Figura 2.7: Estrutura de um segmento TCP

Figura adaptada de http://nmap.org/book/images/hdr/MJB-TCP-Header-800x564.png

Tabela 2.4: **Descrição dos campos do cabeçalho de um segmento TCP**

Nome do Campo	**Tamanho**	**Descrição**	**Campo Obrigatório**
Source Port	16 *bits*	Indica a porta de origem do pacote.	Sim
Dest Port	16 *bits*	Armazena a porta de destino da conexão TCP. Este campo, bem como o anterior, juntamente com os endereços de origem e destino (presentes no datagrama IP [1]) são utilizados para identificar unicamente os dois processos que estão conectados nos sistemas finais na conexão TCP [41].	Sim
Acknowledgment Number	32 *bits*	Este campo é ajustado para o número do último byte confirmado no destino acrescido uma unidade, isto é, contém o número do próximo byte de carga útil esperado.	Válido apenas se *Flag* "A" estiver setada.

Continuação na próxima página

Tabela 2.4 – continuação da página anterior

Nome do Campo	Tamanho	Descrição	Campo Obrigatório
Data offset	4 *bits*	Determina o tamanho do cabeçalho ocupado no segmento TCP. Os valores podem variar de 20 a 60 *bytes*. A partir deste campo, é possível descobrir onde termina o cabeçalho e onde começa a carga útil do segmento.	Sim
Reserved	6 *bits*	Espaço reservado no cabeçalho TCP para utilizações futuras. Uma das possíveis utilizações destes campos seria o indicativo do tipo de carga útil que o segmento está carregando, de forma a permitir algum tipo de qualidade de serviço.	Não
Flag Congestion Window Reduced - CWR (C)	1 *bit*	*Flag* utilizada pelo emissor para indicar que ele recebeu segmentos com a *flag* ECE selecionada [56].	Não
Flag ECN Echo-ECE (E)	1 *bit*	Esta *flag* possui duas utilizações. Se a *flag* SYN está habilitada, serve para indicar que o emissor oferece suporte a ECN (*Explicit Congestion Notification*). Caso contrário, indica que o pacote foi marcado durante a transmissão.	Não
Flag Urgent-URG (U)	1 *bit*	Indica se o campo "*Urgent pointer*" deve ser considerado.	Não
Flag Acknowledgment-ACK (A)	1 *bit*	Informa que o campo "*Acknowledgment Number*" é válido e deve ser considerado. Quando este campo é marcado, significa que o pacote está confirmando o recebimento de um segmento anterior.	Não
Flag Push-PSH (P)	1 *bit*	Indica ao receptor que ele deve transmitir todos os dados do *buffer* para a camada superior.	Não
Flag Reset (R)	1 *bit*	*Flag* utilizada para indicar que a conexão será finalizada devido a um evento inesperado. Por exemplo: o recebimento de um pacote de confirmação referente a um segmento nunca enviado.	Não
Flag SYN (S)	1 *bit*	É utilizado no primeiro segmento TCP para inicialização de uma conexão.	Apenas no início da conexão
Flag FIN (F)	1 *bit*	Utilizada no último segmento TCP para indicar a finalização de um fluxo da conexão.	Apenas na finalização da conexão
Window	16 *bits*	Indica o tamanho máximo da janela de recepção.	Sim
Checksum	16 *bits*	Soma de verificação, utilizado pelo receptor, para checar se o segmento TCP está corrompido. Esse campo de verificação inclui todo o cabeçalho TCP, bem como a carga útil.	Sim

Continuação na próxima página

Tabela 2.4 – continuação da página anterior

Nome do Campo	Tamanho	Descrição	Campo Obrigatório
Urgent Pointer	16 *bits*	Este campo é utilizado juntamente com a flag URG para identificar a parte da carga útil que precisa ser transferida imediatamente para a aplicação.	Válido apenas se *Flag* "U" estiver setada.
Options	Múltiplo de 32 *bits*	pode ser utilizado para diversos propósitos, entre os quais, a negociação do tamanho máximo do segmento (MSS). Este campo deve ser um múltiplo de 32 *bits*. Caso o campo não seja completado com dados válidos, ele terá de ser preenchido com *bits* adicionais.	Opcional

Processo de estabelecimento de conexão

O processo de estabelecimento de conexão no TCP utiliza um esquema de apresentação em três vias (conhecido como *3-way handshake*), conforme apresentado na Figura 2.8.

Tipicamente, numa conexão TCP existe um *host* designado de servidor (que aguarda passivamente por conexões) num extremo e o cliente no outro. O cliente inicia a conexão enviando um pacote TCP com a *flag* SYN ativa e espera-se que o servidor aceite a conexão enviando um pacote SYN+ACK. Se, durante um determinado espaço de tempo, esse pacote não for recebido ocorre um *timeout* e o pacote SYN é reenviado. O estabelecimento da conexão é concluído por parte do cliente, confirmando a aceitação do servidor respondendo-lhe com um pacote ACK.

Durante estas transmissões iniciais, são trocados números de sequência iniciais entre os *hosts* os quais identificam os dados ao longo do fluxo, bem como servem como contadores dos *bytes* transmitidos durante a fase de transferência de dados (sessão).

Controle de fluxo

O TCP provê um serviço de controle de fluxo às suas aplicações para eliminar a possibilidade de o remetente saturar o *buffer* do destinatário. Assim, o controle de fluxo visa realizar uma sincronização entre a velocidade de envio e a velocidade com que o destinatário é capaz de tratar os segmentos TCP.

O TCP provê este serviço fazendo com que o cliente de uma conexão TCP armazene uma variável, chamada janela de recepção. Esta variável é utilizada para informar ao TCP qual o espaço ainda disponível no *buffer* do destinatário. Como o TCP é *full-duplex*, ambos os lados necessitam manter seu *buffer* e a variável sobre o *buffer* do *host* remoto.

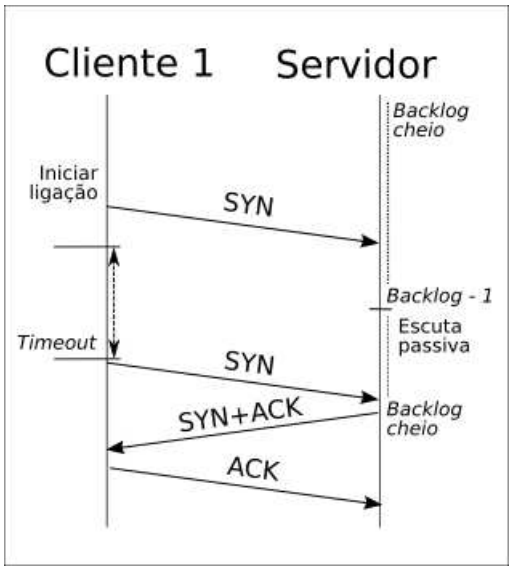


Figura 2.8: 3-way handshake
Figura extraída de http://upload.wikimedia.org/wikipedia/commons/a/a1/TCP_establishment.png

Controle de Congestionamento

O TCP utiliza uma abordagem para controle de congestionamento na qual ele restringe a taxa com que o emissor injeta dados na rede. O valor dessa taxa irá depender do nível de congestionamento percebido na rede. Se o emissor identificar que não há congestionamento aumentará a taxa de envio. Se por outro lado, houver algum nível de congestionamento na rede ele diminuirá tal taxa. Basicamente, o TCP utiliza uma abordagem conhecida como "aumento aditivo, diminuição multiplicativa" (AIMD[7]) [41] para definir como o controle de congestionamento irá realizar o aumento e diminuição da taxa de envio. Nesta abordagem, o crescimento da taxa de envio é dado a partir da adição de um fator à taxa atual (Equação 2.2) enquanto que a diminuição, decorrente de eventos de perdas, é realizado, a partir da multiplicação da taxa atual por um segundo fator (Equação 2.3) [75].

$$x(t+1) = x(t) + \alpha; \quad 0 < \alpha \tag{2.2}$$

$$x(t+1) = \beta x(t); \quad 0 < \beta < 1 \tag{2.3}$$

onde,

[7] do inglês *Additive Increase Multiplicative Decrease*

- $x(t)$ é o valor atual da taxa de envio;
- $x(t+1)$ será a próxima taxa de envio a ser utilizada;
- α é o fator de incremento da taxa de envio;
- β é o fator de diminuição da taxa de envio;

Para realização do processo de controle de congestionamento, o TCP mantém uma variável de estado que indica o fluxo máximo de dados que pode ser enviado pela rede e detecta o congestionamento com base no recebimento de pacotes de confirmação duplicados (indicando que alguns pacotes estão faltando na série), pelo não recebimento de pacotes de confirmação (*timeout*) ou com o recebimento de pacotes com a *flag* "ECE".

A forma como TCP irá incrementar ou decrementar sua janela de congestionamento depende de qual implementação está sendo utilizada. Existem algumas variações da implementação do TCP, conhecido como *flavors*, que implementam suas próprias estratégias para controle da janela de congestionamento. Esses *flavors* utilizam dois estados de crescimento da janela de congestionamento, que são: partida lenta e prevenção de congestionamento, descritos abaixo [33]:

Partida lenta - Este estado é executado em dois momentos durante uma sessão TCP. No início da transmissão e após eventos de *timeout*. Ela recebe este nome devido ao fato de que a taxa inicial deste processo sempre é o menor valor de envio permitido pelo TCP. O principal objetivo deste estado é possibilitar que a sessão TCP atinja a máxima capacidade de transmissão no menor espaço de tempo. Para tal, a cada pacote de confirmação recebido (ACK) envia dois pacotes de dados. Ou seja, basicamente, o TCP dobra sua taxa de envio a cada RTT sem perda. Este estado irá perdurar até o primeiro evento de perda detectado. Após isso, o TCP irá mover-se para o estado de "Prevenção de congestionamento".

Prevenção de congestionamento - Este estado tem por objetivo manter o fluxo de envio em um nível mais otimizado quanto possível, isto é, utilizar todos os recursos disponíveis e evitar perdas de pacotes. Para tal, o TCP irá acrescer um segmento à taxa de envio corrente a cada RTT sem perda.

A seguir serão apresentadas breves descrições de alguns mecanismos de controle de congestionamento mais conhecidos do TCP.

TCP Tahoe

O Tahoe [35] foi um dos primeiros algoritmos de controle de congestionamento para o TCP desenvolvidos. Neste algoritmo, caso um evento de perda seja detectado, a janela de congestionamento é reduzida para 1 MSS[8] e o estado de partida lenta é acionado.

Por se tratar de um algoritmo muito primitivo, possui diversas deficiências que acarretam baixo desempenho, como o contínuo ajuste da taxa de envio ao mínimo, mesmo que haja uma perda ocasional de dados.

TCP Reno

O TCP Reno pode ser considerado a versão mais conhecida do protocolo TCP. Este algoritmo foi desenvolvido visando sanar alguns dos problemas detectados no TCP Tahoe, onde a janela de congestionamento era reduzida a 1 MSS sob quaisquer circunstâncias.

Diferentemente do Tahoe, o Reno não trata todos os eventos de perda de maneira uniforme. No caso de eventos de perda detectados via *timeout*, o tratamento dado é o mesmo que no Tahoe. Contudo, quando eventos de perda são detectados a partir do recebimento de pacotes de confirmação duplicados considera-se que, embora esteja havendo perdas durante a transmissão, alguns pacotes continuam a chegar no seu destino. Como consequência, a taxa de transmissão não é reduzida para 1 MSS, mas sim reduzida à metade da taxa atual de envio, entrando numa fase conhecida como recuperação rápida [67], onde o crescimento da janela de congestionamento passa a ser linear, aumentando 1 MSS a cada RTT.

TCP Cubic

O algoritmo de controle de congestionamento Cubic [26], como o próprio nome diz, utiliza uma função polinomial de terceira ordem, ou mais precisamente uma função cúbica, para controlar como a janela de congestionamento será alterada. Este algoritmo é considerado uma evolução do algoritmo TCP BIC [77], por ser menos agressivo que seu predecessor,

[8] Abreviatura de *Maximum Segment Size*, refere-se a quantidade máxima de dados da aplicação que pode adicionada a um segmento TCP [41] e como consequência ao tamanho mínimo da janela de congestionamento.

possibilitando um melhor compartilhamento da largura de banda com os demais fluxos, mantendo, no entanto, a estabilidade e escalabilidade [33] de seu predecessor.

A função cúbica utilizada tem como base o último evento de perda ocorrido, de modo a estipular qual será a nova taxa de envio, conforme definido na Equação 2.4.

$$W(t) = C(t - \sqrt[3]{\frac{W_{max}\beta}{C}})^3 + W_{max} \tag{2.4}$$

onde,

- $W(t)$ é a nova taxa de envio;
- C é uma constante escalar;
- t é o tempo decorrido desde o último evento de perda;
- β é o fator de redução;
- W_{max} é o tamanho da janela antes do último evento de perda.

O TCP Cubic é, atualmente, o algoritmo de controle de congestionamento padrão do protocolo TCP no sistema operacional Linux. Nesta implementação, os valores β e C assumem respectivamente os valores 0.2 e 0.4 [26].

2.3 Adaptação Multimídia

Embora existam propostas para adaptação multimídia envolvendo as camadas mais baixas da pilha TCP/IP [44], em geral as abordagens de adaptação voltadas para transmissão na Internet têm como foco as camadas de aplicação e/ou transporte, visto que, em níveis mais baixos da suíte TCP/IP há uma heterogeneidade muito grande nas características das rede, tornando inviável o desenvolvimento de uma solução ideal para todos os casos.

Nesta seção serão abordadas as principais formas de adaptação em nível de camada de transporte e aplicação.

2.3.1 Adaptação a nível de camada de aplicação

O processo de adaptação em nível de aplicação tem como principal objetivo a alteração das propriedades do fluxo, em termos de codificação, de modo que o fluxo resultante atenda aos requisitos de largura de banda necessária para transmissão [61]. Este processo pode incluir tarefas como: redução da resolução do fluxo (*Spatial Scalability*), descartes de *frames* (*Temporal Scalability*), variação da taxa de codificação, variação da qualidade de codificação etc.

A seguir são apresentadas algumas das mais conhecidas abordagens para realização de adaptação multimídia em nível de aplicação.

Variable Bit Rate–VBR

Variable Bit Rate ou Taxa Variável de *Bits* é uma técnica para codificação de fluxos de áudio e vídeo que consiste na especificação da qualidade esperada do fluxo resultante. A partir desta informação, os codificadores irão tentar realizar a codificação, mantendo a qualidade especificada. Uma consequência disto é a variação da quantidade de dados gerados por unidade de tempo, visto que no processo de codificação, *frames* com diferentes informações consomem diferentes quantidades de *bits* [43].

Esta técnica é bastante difundida, sendo utilizada por diversos codificadores multimídia, entre os quais apontam-se: MP3, H.264, Speex [70] etc. Esta abordagem possui vantagens e desvantagens se comparadas ao *Constant Bit Rate*. Como ponto positivo mostra-se a capacidade de manter uma qualidade média do fluxo gerado, o que resulta numa melhor percepção para o usuário. Por outro lado, há uma flutuação na largura de banda necessária para transmissão (visto que não é possível especificar uma taxa de *bits* média), podendo resultar em perda de informações em *links* congestionados.

Uma abordagem utilizada para solucionar o problema supracitado, é utilizar uma variante do VBR, conhecida como ABR (*Average Bit Rate*). Nesta abordagem, é possível definir uma taxa média esperada a partir da qual os codificadores irão tentar otimizar a qualidade do dado gerado. Diferentemente do CBR, nesta abordagem a taxa final não é fixa mas variável. Contudo, ao longo do tempo, tende a ser mantida uma média da taxa de fluxo gerada.

Constant Bit Rate–CBR

Nesta abordagem de codificação, os codificadores irão tentar comprimir o fluxo multimídia utilizando uma taxa constante, especificada *a priori*. Como consequência, haverá uma variação na qualidade do fluxo gerado. Esta variação pode tornar-se relevante em fluxos multimídia que apresentam uma heterogeneidade durante sua transmissão. Por exemplo, suponha que desejamos codificar um vídeo que apresente tanto cenas relativamente estáticas quanto cenas com grande movimentação, onde há uma grande variação entre os *frames*. Se codificarmos este vídeo utilizando a abordagem CBR e estipulando que a taxa pretendida seja alta, haverá um desperdício na taxa de *bits*, visto que algumas cenas irão ser codificadas a uma qualidade mais alta que o necessário. Se por outro lado, definirmos que a taxa utilizada seja baixa, a qualidade de algumas cenas ficará comprometida.

Temporal Scalability

Esta técnica tem como objetivo realizar a adaptação de fluxos de vídeo pré-codificados, sendo considerada uma das técnicas mais conhecidas e utilizadas no processo de adaptação.

Basicamente, esta abordagem realiza o descarte de alguns *frames* no decorrer da transmissão objetivando reduzir a taxa de transmissão final. Para tal, ela faz uma varredura, buscando os *frames* que apresentam uma menor importância na transmissão.

Alguns codificadores de vídeo, como o MPEG-4, ao realizar o processo de codificação definem três tipos diferentes de *frames*, onde tem-se um frame base (conhecido como *I-Frame*) e dois tipos de *frames* incrementais (*P-Frame* e *B-Frame*) [43]. O primeiro tipo, necessita de uma maior quantidade de *bits* para ser codificado, porém é capaz de ser decodificado sem nenhuma informação adicional. Por outro lado, os *frames* P e B necessitam de uma menor quantidade de *bits* para codificação mas têm como requisito um *frame* base. Neste contexto, o processo de *Temporal Scalability* realiza o descarte de alguns *frames* incrementais (que possuem uma menor importância), visto que embora haja uma queda na qualidade final do fluxo, ainda assim será possível a correta decodificação dos demais *frames* do fluxo.

2.3.2 Adaptação a nível de camada de transporte

A adaptação a nível de camada de transporte é realizada a partir da alteração de algumas propriedades que irão determinar como a informação passada pela camada de aplicação deverá ser enviada. Estas propriedades incluem desde a escolha do algoritmo de controle de congestionamento mais adequado até a agregação de meta informações sobre os pacotes de dados que irão interferir em como o processo de transmissão deverá ser realizado.

A seguir são apresentadas duas das mais conhecidas abordagens para adaptação a nível de camada de transporte: *Priorização* e *Temporizaçao* de pacotes.

Priorização de pacotes

O objetivo desta abordagem de adaptação é possibilitar uma atribuição de grau de importância aos pacotes adicionados à fila de envio, de modo que, em caso de congestionamento da rede ou quando um determinado limiar é atingido, seja possível realizar a diminuição da quantidade de dados a serem enviados a partir do descarte de alguns pacotes com baixa prioridade.

Nesta abordagem, a atribuição de prioridades é deixada sob responsabilidade da aplicação que, ao enviar os dados para a camada de transporte, define o quão importante é sua informação. Para tal, diferentes critérios podem ser utilizados. Por exemplo, suponha que dois usuários decidam estabelecer uma sessão de videoconferência. Neste tipo de aplicação, embora ambos os usuários estejam interessados em ter tanto as informações de áudio quanto de vídeo, em caso de congestionamento da rede, pode-se optar por enviar apenas as informações de áudio, de modo a possibilitar que, mesmo que esteja havendo um sério problema na comunicação, ambas as partes consigam se comunicar.

Uma outra abordagem que poderia ser adotada ao se definir os critérios para adaptação seria utilizar as informações dos tipos de *frames* codificados (I,B,P) para definir quais *frames* são mais importantes para transmissão. Esta abordagem assemelha-se bastante à realizada no método de adaptação *Temporal Scalability* apresentado anteriormente, tendo como vantagem o fato da possibilidade de decisão tardia se o processo de adaptação será realizado. Visto que a decisão de descarte ou não só será realizada momentos antes do envio, tendo-se como consequência um melhor panorama das condições da rede para transmissão.

Esta abordagem de adaptação tem sido empregada em alguns trabalhos na literatura, como os apresentados nas referências [25], [72] e [47]. Este último trabalho utiliza uma variação da abordagem priorização de pacotes que inclui aspectos de temporização, a serem discutidos a seguir.

Temporização de pacotes

Uma das propriedades mais conhecidas das aplicações multimídia é a sensibilidade ao tempo, conforme discutido na Seção 2.1, onde cada pedaço de informação possui seu próprio tempo de execução, que caso não seja respeitado, perde seu valor no destinatário e geralmente é descartado. Em vista disso, a abordagem de temporização de pacotes consiste no descarte de certos pacotes que tenham seu tempo de utilização expirados e, portanto, são inúteis.

A abordagem de temporização de pacotes utiliza a informação de marca de tempo para verificar se o pacote a ser enviado será ou não válido no seu destino. Para tal, a aplicação, ao enviar o pacote à camada de transporte, especifica seu tempo de expiração que será verificado antes do envio. Caso seu tempo tenha sido expirado (considerando também o tempo necessário para transmissão) ele é descartado.

2.4 Conclusões do Capítulo

Neste capítulo foi apresentado a fundamentação teórica relativa a este trabalho. Pode-se constatar que viabilizar transmissões multimídia em redes de melhor esforço, como a Internet, envolve diversos fatores, de modo a obter uma adaptação do fluxo satisfatória. Neste contexto, diversas abordagens para adaptação foram propostas, envolvendo diferentes camadas da suíte de protocolos TCP/IP. No entanto, nenhuma abordagem mostra-se ótima para todos os cenários, visto que dependendo das condições da rede no momento da transmissão uma abordagem pode obter melhores resultados que outras [43]. Como consequência, faz-se necessária a utilização de diversas abordagens em conjunto objetivando uma melhoria no processo de adaptação como um todo.

Capítulo 3

Trabalhos Relacionados

Neste capítulo é apresentada uma discussão a respeito de alguns trabalhos relacionados à esta dissertação. Pode-se constatar que, apesar de existirem na literatura abordagens de adaptação de fluxos multimídia, grande parte delas consiste de métodos de adaptação pontuais, sendo ainda pequeno o número de trabalhos voltados para o gerenciamento do processo de adaptação multimídia, em especial que englobe as camadas de transporte e aplicação.

3.1 *An Efficient QoS Negotiation Mechanism for IPTV Service in Heterogeneous Networks*

O trabalho apresentado em [38] tem como foco o processo de adaptação de fluxos em transmissões IPTV. A principal motivação deste trabalho é a realização de adaptação levando-se em consideração *i)* as restrições dos dispositivos envolvidos na transmissão e preferências do usuário, como capacidade de processamento resolução da tela dos dispositivos, qualidade mínima pretendida etc ; *ii)* e variação nas condições de transmissão. Para tal, propõe-se um mecanismo de adaptação de fluxos, que consiste de uma função de adaptação da qualidade e um mecanismo para negociação das propriedades entre os *hosts* envolvidos. Ademais, este trabalho define as abordagens *Temporal Scalability* e *Spatial Scalability* como métodos de adaptação possíveis.

O mecanismo de adaptação desta abordagem possui semelhanças com as desenvolvidas nessa dissertação, visto que, para realização da adaptação, divide o processo de adap-

tação em duas etapas: uma de configuração, realizada durante o processo de inicialização da transmissão, que tem por objetivo definir os parâmetros iniciais da transmissão; e uma segunda, realizada durante o decorrer da transmissão visando realizar o processo de adaptação de acordo com as variações de condições de transmissão. As etapas de configuração e adaptação, definidas neste trabalho como *Session Setup Phase* e *In-Session Phase*, respectivamente, são apresentadas na Figura 3.1.

```
// Session Setup Phase
    CASE D_usr == CELL PHONE
        IF Q_i < BW_est
          THEN set R_i = Q_i
    CASE D_usr == PDA
        IF Q_j < BW_est
          THEN set R_j = Q_j
    CASE D_usr == NOTEBOOK
        IF Q_k < BW_est
          THEN set R_k = Q_k
    (where, Q_i < Q_j < Q_k)

// In-Session Phase
    IF Current R_n with Q_n > BW_est
      THEN set R_n+1 = Q_n-1
    ELSE Current R_n with Q_n ≤ BW_est
      {
        IF Q_n+1 ≥ BW_est
          THEN set R_n+1 = Q_n
        ELSE Q_n+1 < BW_est
          THEN set R_n+1 = Q_n+1
      }
```

Figura 3.1: Função de negociação de parâmetros de qualidade.
Figura extraída de [38].

Para realização do processo de negociação das propriedades de transmissão, neste trabalho utiliza-se o protolo SIP [29] em conjunto com o protocolo SDP [28] para troca dos parâmetros de configuração da transmissão. Cada mensagem de adaptação pode conter até quatro tipos de informações diferentes, que são: tipo do dispositivo (ex.: *notebook*, PDA, celular), nome do recurso adaptado (ex.: memória, CPU, nível de bateria etc.), valor do atributo de adaptação e largura de banda. Com o intuito de validar o trabalho desenvolvido, os autores realizaram uma simulação visando demonstrar o processo de adaptação através da transmissão de um fluxo IPTV.

Apesar deste trabalho apresentar uma interessante abordagem para adaptação de fluxos, ele provê uma flexibilidade no processo de adaptação, visto que: *i)* define uma política de adaptação única, tendo como consequência a impossibilidade de uma customização do processo de adaptação, de modo a atender a cenários específicos; *ii)* utiliza um conjunto de parâmetros de avaliação limitados, *iii)* restringe o processo de adaptação a apenas duas abor-

dagens possíveis. Adicionalmente, não foram apresentados detalhes de como o modelo proposto pode ser estendido visando a implementação de novas abordagens de adaptação.

3.2 *ViTooKi - The Video ToolKit*

ViTooKi [40] é um *framework* de código aberto para transmissão multimídia desenvolvido na *Universität Klagenfurt*. Este *framework* tem como objetivo possibilitar o envio, adaptação e renderização de fluxos de vídeo e áudio armazenados. Possui suporte a adaptação multimídia em nível de aplicação realizando diversos tipos de adaptação, tais como: *Temporal Scalability*, *Spatial Scalability*, redução de cores, redução de qualidade etc. Além disso, no processo de transmissão, utiliza o protocolo UDP juntamente com o RTP, implementando a sua própria abordagem de controle de congestionamento TCP-*Friendly*.

Visando possibilitar extensibilidade, utiliza-se uma abordagem modular onde diferentes métodos de adaptação podem ser acoplados à infraestrutura, ainda que seja necessário a recompilação do sistema cliente de modo a habilitar o novo método.

No entanto, embora o **ViTooKi** possua uma estrutura modular, que permita um fácil acoplamento de novos métodos de adaptação multimídia, não existe um enfoque no sentido de efetuar uma separação de responsabilidades entre o desenvolvedor da aplicação final e o desenvolvedor das políticas de adaptação. Como consequência, o desenvolvedor-cliente continua responsável por definir como os diferentes mecanismos de adaptação irão interagir, tendo, portanto, a necessidade de conhecer as características das diferentes abordagens de adaptação, tornando o processo de desenvolvimento de aplicações complexo.

3.3 *Dynamic Multimedia Stream Adaptation and Rate control for Heterogeneous Networks*

No trabalho descrito em [69] é apresentada uma arquitetura para adaptação de transmissões multimídia que leva em consideração tanto as restrições dos dispositivos envolvidos na transmissão quanto as condições de transmissão corrente. Nesta arquitetura é definido um engenho de adaptação, que realiza adaptação do fluxo transmitido baseado numa política de adaptação. Para realização do processo de adaptação é considerada a utilização de um *proxy*

localizado entre o transmissor e o receptor, chamado *Content Adaptation Node*. Tal *proxy* ficaria responsável por verificar as condições da transmissão para realização dos devidos ajustes, através tanto da alteração da codificação do fluxo enviado quanto do controle da taxa de envio, a partir da utilização de um algoritmo de controle de congestionamento próprio baseado no algoritmo TFRC.

Embora a proposta apresentada se mostre como uma interessante forma de prover uma adaptação de fluxos multimídia em redes heterogêneas, ela apresenta algumas limitações, como a restrição do mecanismo de controle de congestionamento utilizado, que muitas vezes não é o mais adequado a todos os tipos de fluxos multimídia. Além disso, a necessidade de adição de um componente entre os elementos transmissores e receptores impõe uma dificuldade a mais no processo de implantação desta proposta, tanto em termos de escalabilidade (se for considerado que um *proxy* irá servir para todos os fluxos existentes) ou então a necessidade de controle sobre o domínio administrativo para garantir a presença deste elemento.

3.4 *Network Adaptation Layer*

O projeto *Tailor-made Congestion Control* [74] desenvolvido pela Universidade de *Innsbruck*/Áustria tem como enfoque o desenvolvimento de mecanismos que provejam qualidade de serviço fim-a-fim (QoS) em redes de melhor esforço. Para tal, é proposta uma camada de adaptação, localizada entre a camada de aplicação e a camada de transporte e rede. Esta camada tem como objetivo gerenciar a infraestrutura de transmissão através do controle das propriedades da transmissão como o protocolo de transporte utilizado, mecanismos de controle de congestionamento etc. Ademais, o processo de adaptação da transmissão é baseado na especificação dos requisitos do tráfego definido pelas aplicações. Na Figura 3.2 é apresentada a arquitetura da solução proposta e as interações da nova camada de adaptação com as camadas superiores e inferiores.

A arquitetura desta abordagem foi desenvolvida de tal forma que aplicações já existentes podem se beneficiar desta camada de adaptação sem a necessidade de qualquer reprogramação. Tal característica é devida ao fato desta camada de adaptação ter sido definida para ser implementada abaixo da interface de soquetes utilizada pelas aplicações, adicionando apenas a interface de adaptação necessária [32], tornando-se, portanto, transparente para as

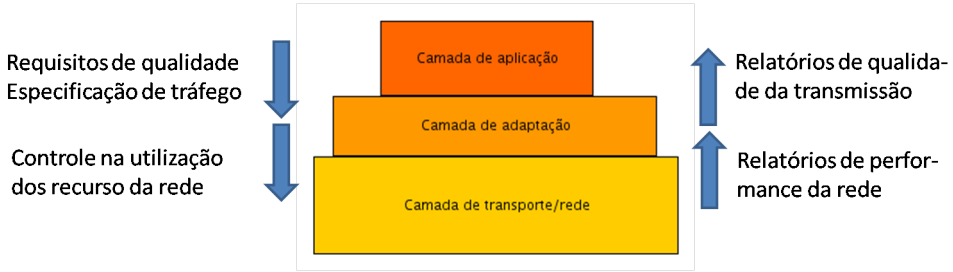


Figura 3.2: Proposta para uma camada de adaptação multimídia
Figura adaptada de [74]

aplicações preexistentes. Contudo, a utilização desta abordagem possui um revés, que é a impossibilidade de definição de uma interface única de adaptação que seja capaz de possibilitar uma adaptação de todos os tipos de mecanismos de transporte de maneira específica. Além disso, o trabalho não deixa claro como será o processo de descrição dos requisitos de qualidade de serviço, de modo a atender às especificidades dos diferentes tipos de aplicações.

3.5 Considerações sobre os Trabalhos Relacionados

Tendo como base as observações extraídas a partir dos trabalhos apresentados neste capítulo, pode-se perceber que as infraestruturas para adaptação de fluxo existentes, em geral, têm como foco a adaptação em camadas isoladas (isto é, camada de aplicação ou transporte), com exceção de [69]. Além disso, não se observou nenhum trabalho que contemple ao mesmo tempo tanto a descrição de políticas de adaptação quanto a flexibilidade na adição de novos mecanismos de adaptação de fluxos.

Capítulo 4

Uma Infraestrutura para Transmissão de Conteúdo Multimídia com Suporte à Adaptação de Fluxos

O objetivo deste capítulo é apresentar a infraestrutura desenvolvida, discutindo as principais características e seu funcionamento. Inicialmente, será apresentada uma visão geral a respeito das características da infraestrutura. Após isso, apresenta-se a arquitetura da solução proposta, mostrando quais são e como se relacionam os diferentes componentes da infraestrutura. Para finalizar descreve-se o funcionamento da infraestrutura como um todo, de modo a prover adaptação do fluxo transmitido.

4.1 Visão Geral da infraestrutura

A ideia básica da infraestrutura proposta é criar uma abstração entre as aplicações finais e os recursos necessários para o estabelecimento de transmissões multimídia e do processo de adaptação de fluxos, conforme ilustrado na Figura 4.1. Deste modo, almeja-se diminuir a complexidade envolvida no processo de desenvolvimento de tais aplicações.

Como ponto inicial para resolução dos problemas levantados na Seção 1.1 foram identificadas as etapas que fazem parte do ciclo de desenvolvimento de uma aplicação multimídia com suporte à adaptação de fluxos. O objetivo desta tarefa é tornar claro quais são os atores envolvidos neste processo, isolando as devidas responsabilidades. Na Figura 4.2 é apresen-

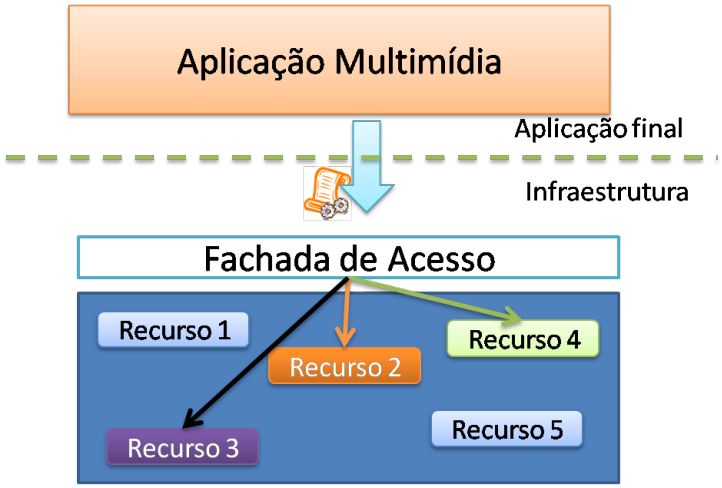


Figura 4.1: Abstração de acesso aos recursos da infraestrutura de adaptação.

tada a interação entre os diferentes papéis discutidos a seguir:

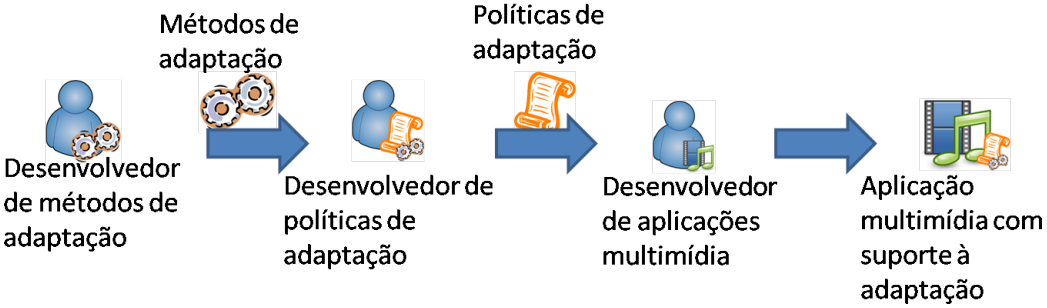


Figura 4.2: Interação entre os papéis.

Desenvolvimento de métodos de adaptação - Esta etapa pode ser considerada a mais básica de todo processo. A partir dela são criadas as técnicas de adaptação que serão utilizadas no processo de transmissão de fluxos multimídia. Esta etapa está intrinsecamente relacionada ao processo de desenvolvimento da tecnologia a que ela dará suporte, visto que as características da tecnologia irão definir como poderá ser possível realizar adaptações. Por exemplo, considere o codificador para transmissão de áudio *Speex* [70]. Este *codec* foi desenvolvido tendo como foco transmissões de voz no contexto de VoIP. Uma das características desse *codec* é a possibilidade do ajuste da qualidade do áudio transmitida de acordo com a taxa de transmissão máxima estipulada, aumentando e diminuindo a qualidade do fluxo de modo a atender ao requisito de largura de banda utilizada. Sendo assim, a variável de ajuste da taxa de transmissão

máxima permitida passa a ser uma variável moduladora deste mecanismo de codificação de transmissões de áudio, ou seja, pode-se, a partir dela, efetuar as adaptações necessárias em seu funcionamento. Um outro exemplo seria a adaptação de como os pacotes de informações serão transmitidos em nível de camada de transporte atribuindo a eles prioridades ou tempos de expiração [47].

A partir dos exemplos apresentados, pode-se concluir que, apesar dos mecanismos de adaptação possibilitarem a alteração das características dos fluxos durante a transmissão, o controle de como este processo deve ser realizado depende de uma gerência em um nível superior, visto que tanto os recursos de transmissão quanto o propósito da transmissão podem variar. Deste modo faz-se necessária a especificação de políticas objetivando descrever como o processo de adaptação deve ser gerenciado durante a transmissão.

Um outro aspecto importante a ser mencionado é o fato de que nenhum método de adaptação de fluxo mostra-se ideal para todas as situações. Este fator é devido às próprias características das entidades a serem adaptadas. Diante disto, faz-se necessário criar mecanismos que possibilitem uma fácil extensibilidade de modo a permitir que novas formas de adaptação possam ser adicionadas rapidamente. Tal objetivo foi alcançado a partir da adoção de uma abordagem baseada no conceito de *plugins* [8], onde se torna possível a adição flexível de novos componentes de adaptação à infraestrutura.

Desenvolvimento de políticas de adaptação - Partindo do pressuposto de que os mecanismos de adaptação foram definidos e podem ser utilizados, faz-se necessário definir como tais mecanismos serão utilizados durante uma transmissão. Normalmente tal "inteligência" é adicionada à aplicação final criando uma sobrecarga no processo de desenvolvimento. Um outro aspecto negativo desta abordagem é a falta de flexibilidade no processo de customização da política de adaptação, tendo como consequência uma limitação na possibilidade de alteração da política de adaptação e, portanto, a disponibilidade das aplicações em diferentes ambientes.

Por exemplo, considere a seguinte situação: uma aplicação de videoconferência que deve ser utilizada em ambientes bastante heterogêneos, como uma rede *Gigabit* ou um

canal ADSL ou ainda realizar a transmissão em um ambiente de rede sem-fio onde as perdas são decorrentes principalmente da corrupção de dados pelo meio não-confiável de transmissão. Neste contexto, os fatores largura de banda e perda/recuperação de erros devem ser levados em consideração para definir a política de adaptação que será utilizada. Uma outra situação que poderia ser apresentada seria uma aplicação para transmissão de vídeo utilizada em diferentes propósitos, como a transmissão de um discurso ou então a transmissão de uma atividade esportiva. No primeiro caso, se a rede estiver em um alto grau de congestionamento, deveria ser dada prioridade aos pacotes de áudio de forma que o discurso possa ser entendido mesmo que a imagem do locutor não possa ser transmitida de forma compreensível. No segundo cenário, a principal mensagem transmitida é a mensagem visual, isto é a imagem do que está sendo exibido. Nesta situação os pacotes de vídeo passam a ter uma maior prioridade na transmissão. Neste exemplo, a importância de cada pacote transmitido depende da semântica da aplicação.

Com base nos exemplos apresentados anteriormente, percebe-se a necessidade de uma flexibilização na definição de políticas de adaptação visto que nenhuma política mostra-se ótima para todas as situações. Uma outra questão trabalhada foi a forma como as políticas possam ser descritas objetivando diminuir a complexidade envolvida neste processo e possibilitar que haja uma separação de responsabilidades entre quem define as políticas e quem as utiliza e a interação desta com a infraestrutura. Neste sentido, foi utilizada uma abordagem baseada em modelo declarativo.

Desenvolvimento de aplicações finais - A partir da definição de uma infraestrutura que possibilite uma fácil adição e remoção de métodos de adaptação e do estabelecimento de um modelo para descrição das políticas de adaptação, a próxima etapa é fazer uso de tais mecanismos para de fato realizar a construção da aplicação multimídia.

Esta etapa deve ser realizada por um desenvolvedor (chamado aqui de desenvolvedor-cliente para distingui-lo do desenvolvedor da infraestrutura), interessado em implementar alguma aplicação multimídia. Neste sentido, ele está na posição de cliente da aplicação, acessando-a apenas via uma API de acesso. Por exemplo, para realizar o estabelecimento de uma transmissão multimídia, o desenvolvedor-cliente escolhe a

política desejada e a infraestrutura fica responsável por realizar todas as configurações necessárias.

A partir do apresentado, abstrai-se do desenvolvedor-cliente um conjunto problemas durante o desenvolvimento. Ele não precisa mais se preocupar com a maneira como a transmissão deve ser configurada para ser realizada com sucesso nem os procedimentos que devem ser realizados em caso de alteração das condições da rede que afetem a transmissão.

4.2 Arquitetura

A arquitetura para desenvolvimento de aplicações multimídia com suporte à adaptação de fluxos é composta por quatro componentes principais: um módulo de descrição e tratamento de políticas de adaptação, um gerenciador de sessões de transmissão, um mecanismo para negociação das propriedades do fluxo com o *host* remoto e uma estrutura que provê suporte à integração de *plugins* de adaptação. Tal arquitetura é ilustrada na Figura 4.3 e os diferentes módulos são discutidos detalhadamente a seguir.

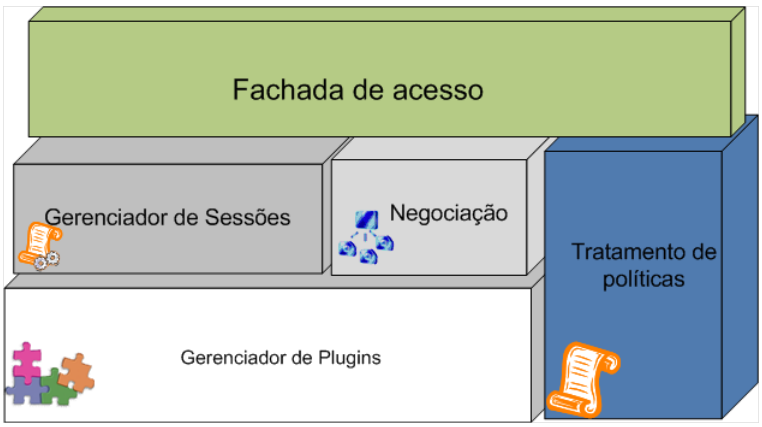


Figura 4.3: Arquitetura da infraestrutura

4.2.1 Infraestrutura para Carregamento de *Plugins* de Adaptação

Conforme apresentado na Seção 4.1, fez-se necessária a implementação de uma estrutura que possibilite a extensibilidade da arquitetura, de modo a permitir que novos mecanismos de

adaptação possam ser adicionados à infraestrutura. De modo a atender tal requisito, utilizou-se uma abordagem baseada em *plugins*.

O desenvolvimento baseado em *plugins* é uma abordagem para desenvolvimento de software que tem como objetivo possibilitar a extensibilidade das funcionalidades do sistema a partir da adição, em tempo de execução, de novos módulos, chamados *plugins* [8][46]. De maneira específica, uma aplicação baseada em *plugin* define pontos de extensão, conhecidos como *Hook*, a partir do qual, um *plugin* pode implementar as funcionalidades descritas nesta interface, sendo possível o carregamento dinâmico de tais módulos. A seguir são apresentadas as principais motivações para o desenvolvimento baseado em *plugins* [23]:

- Possibilitar a extensão do software em tempo de execução, mesmo desconhecendo *a priori* quem irá prover o serviço;
- Permitir uma grande modularização do sistema, diminuindo a complexidade como um todo;
- Desenvolver componentes do sistema sem a modificação de outros módulos ou necessidade de recompilação do núcleo do sistema.

Neste contexto, foram definidos três pontos de extensibilidade na infraestrutura e como consequência, três tipos diferentes de *plugins*, descritos a seguir.

***Plugin* de adaptação da camada de transporte -** o objetivo desta classe de *plugins* é realizar o processo de adaptação em nível de camada de transporte. Em vista disto, é de responsabilidade deste tipo de *plugin* o estabelecimento do processo de comunicação, em nível de transporte, bem como da adaptação propriamente dita, que será realizada a partir da alteração das propriedades a nível de transporte.

***Plugin* de adaptação da camada de aplicação -** os *plugins* para adaptação em nível de *stream* são responsáveis pelo processo de recuperação e codificação dos dados de aplicação. Neste tipo de *plugin* deverão ser implementadas técnicas visando a alteração da codificação do fluxo de acordo com as condições de transmissão.

***Plugin* de negociação -** *plugins* responsáveis pelo processo de negociação das propriedades da transmissão. A adoção de um modelo de *plugins* também para o processo

de negociação possibilita a utilização de diferentes padrões de negociação (discutido na Seção 4.2.4).

Projeto de Baixo Nível

O módulo de carregamento de *plugins* utiliza como base o padrão de projeto *Plugin*, descrito em [23]. Este padrão consiste da definição de uma entidade gerenciadora de *plugins* - responsável pelo carregamento, descarregamento e disponibilização dos serviços dos *plugins* ao restante do sistema, por uma classe que define as atribuições básicas de um *plugin* e pelos *plugins* propriamente ditos. Em virtude da infraestrutura desenvolvida possibilitar diferentes pontos de extensão, fez-se necessária a especialização da interface do *plugin* básico em três diferentes tipos de *plugins* especializados de acordo com suas funcionalidade, conforme apresentado na Figura 4.4.

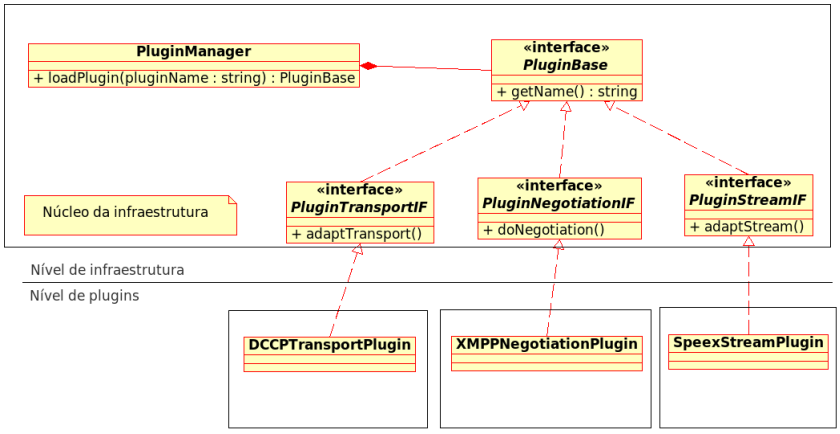


Figura 4.4: Modelo de carregamento de *plugins*

4.2.2 Módulo de Descrição de Políticas de Adaptação

Conforme discutido na Seção 4.1, foi desenvolvido um mecanismo capaz de possibilitar a descrição de diferentes políticas de adaptação, objetivando a personalização do fluxo multimídia. Neste contexto, cada política de adaptação é responsável por gerenciar como o processo de adaptação deverá ser realizado. De maneira específica, uma política apresenta uma estrutura baseada em linguagem XML, onde são descritas as propriedades básicas para realização de uma transmissão multimídia tanto na configuração inicial quanto os eventos de

adaptação que poderão ser disparados no decorrer do tempo, de acordo com as condições da rede.

Uma questão trabalhada no modelo de descrição de políticas desenvolvido foi como estruturar a política de modo o possibilitar uma flexibilidade na descrição das propriedades de adaptação. Este recurso faz-se necessário diante da arquitetura baseada em *plugins* da infraestrutura, onde diferentes *plugins* de adaptação podem ser desenvolvidos. O intuito deste processo foi possibilitar que diferentes mecanismos de adaptações pudessem ter parâmetros diferentes de configuração. Um exemplo disto seria um *plugin* para adaptação de transmissões que utilizem o protocolo de transporte DCCP. Como discutido na Seção 2.2.2, o protocolo DCCP possibilita a alteração do algoritmo de controle de congestionamento utilizado durante a transmissão. Sendo assim, faz sentido que tal propriedade seja acessível em um *plugin* de adaptação para este protocolo de transporte. Por outro lado, o protocolo TCP não possui tal mecanismo de chaveamento de controle de congestionamento e o protocolo UDP sequer dá suporte a mecanismos de controle de congestionamento. Como consequência desta heterogeneidade não faz sentido definir um conjunto de propriedades fixas, mas sim tornar flexível o bastante para que cada *plugin* defina suas próprias propriedades.

O passo inicial para o desenvolvimento deste módulo foi a definição de como uma política de adaptação multimídia poderia ser descrita em termos de funcionalidades. Para tal, considerou-se que qualquer fluxo multimídia possui duas fases de execução durante o ciclo de vida que seria uma etapa de pré-configuração do fluxo e uma segunda etapa de adaptação, como ilustrado na Listagem 4.1.

```
1  inicio politica
2          início etapa_de_configuração:
3              sessao_transporte:
4                  param1 = value1
5                  param2 = value2
6                  param3 = value3
7              fim sessao_transporte:
8
9              sessao_stream:
10                 param4 = value4
11             fim sessao_stream:
12         fim etapa_de_configuração
```

```
13
14        início etapa_de_adaptação:
15            sessao_transporte:
16                param RTT (rtt) :
17                        se rtt < rtt0 entao
18                                // Define-se valores para limiares
                                      mínimos
19                                param5 = value5
20                                param6 = value6
21                                param7 = value7
22                                se jitter == jitter0 entao
23                                        paramK = xn
24                                fim_se
25                                //Outras configurações de transporte
26                        fim_se
27
28                        se rtt1 < rtt0 < rtt2 entao
29                        // Define-se valores para limiares intermediários
30                                ...
31                        fim_se
32                        ...
33                        se rtt > rttN entao
34                        // Define-se valores para limiares máximos
35                                ...
36                        fim_se
37                fim param_RTT
38
39                param Largura_de_banda (lb)
40                        //Configurações com base na largura de banda
41                        ...
42                fim param_Largura_de_banda
43                // Outras configurações
44                        ....
45            fim sessao_transporte:
46
47            sessao_stream:
48                // Define-se as configurações em nível de stream
```

```
49              fim sessao_stream:
50          fim etapa_de_configuração
51  fim politica
```

Listagem 4.1: Modelo de descrição de políticas de adaptação.

Etapa de pré-configuração: A primeira etapa é considerada uma etapa de configuração do fluxo, a ser realizada antes da transmissão multimídia ter início. Nesta etapa, são definidos os parâmetros iniciais para o estabelecimento do fluxo multimídia como codificadores de áudio e vídeo que serão iniciados para gerar o fluxo inicial e o protocolo de transporte que será utilizado. Nesta fase, a configuração apresenta um caráter mais estático se comparado com a etapa de adaptação, visto que não é possível levar em consideração nenhuma informação transiente do ambiente, como o RTT ou largura de banda disponível no momento. Além disso, é possível definir se o processo de adaptação estará ativo ou não. Como consequência, torna-se possível, reduzir a transmissão em uma transmissão comum, isto é, sem qualquer processo de adaptação.

Etapa de adaptação *on-the-fly*: a segunda etapa de configuração é executada durante a transmissão multimídia propriamente dita. Nesta fase o processo de adaptação se dá de forma iterativa, de acordo com alteração de valores de um conjunto de variáveis monitoradas. Tais variáveis são atualizadas a partir de uma lista de "provedores de informação", discriminadas na política. Desta forma, os parâmetros de configuração passam a ser descritos baseados nos estados das variáveis desejadas. Para descrição da regra para cada variável, é necessária a definição dos limiares mínimos e máximos e opcionalmente de valores intermediários, como apresentado na Listagem 4.2.

```
1 variavel (r):
2         se r < limiar_minimo
3         ...
4         se limiar_minimo1 < r < limiar_minimo2
5         se limiar_minimo3 < r < limiar_minimo4
6         ...
7         se r > limiar_maximo
```

Listagem 4.2: Definição de limiares em políticas.

Para maiores detalhes a respeito dos parâmetros de configuração de políticas de adaptação, consultar o Apêndice A.

Implementação

Para descrição do modelo de descrição de políticas de adaptação proposto, optou-se pela utilização do formato XML [73]. XML (*Extensible Markup Language*) é uma linguagem para descrição de dados em formato hierárquico, definida pela W3C, bastante conhecida. XML possibilita a definição de um arquivo para validação da estrutura dos dados definidos. Deste modo, foi possível descrever o modelo de estrutura de uma política de adaptação com a qual as políticas são validadas.

Sendo assim, implementou-se um arquivo *XMLSchema* baseado na estrutura apresentada na Listagem 4.1. Em seguida foi construído o analisador para recuperação das informações descritas nas políticas de adaptação. Para tal, utilizou-se a ferramenta XSD [12] para facilitar o processo de geração do analisador sintático.

4.2.3 Módulo Gerenciador de Sessões

O Gerenciador de Sessões pode ser considerado o ponto central de funcionamento da infraestrutura para adaptação de fluxos multimídia. É de responsabilidade deste módulo realizar o processo de gerência da sessão multimídia, bem como monitorar as condições de transmissão, realizando as ações de adaptação necessárias de acordo com a política de adaptação. Para a realização de tais funções, o Gerenciador de Sessões faz uso dos módulos de negociação, descritor de políticas e da infraestrutura para carregamento de *plugins*. A seguir são descritas as funcionalidades desempenhadas por este módulo.

Criação de sessão de transmissão - O processo de estabelecimento de uma sessão é realizado a partir da invocação pela aplicação cliente, via fachada, de uma solicitação de criação de uma sessão ao Gerenciado de Sessões.

Durante a inicialização, este módulo recebe como entrada um arquivo contendo a política de adaptação, e inicializa, por conseguinte, o módulo de tratamento de políticas. A partir disto, será possível recuperar o conjunto de informações necessárias para a criação da sessão de comunicação. A criação da sessão consistirá da inicialização de

três componentes, que são: *plugin* de transmissão, *plugin* de codificação multimídia e *plugin* de negociação. Tais *plugins* serão carregados pelo **Gerenciador de *Plugins***, conforme discutido na Seção 4.2.1.

De maneira específica são recuperadas as seguintes informações:

- Configuração do *plugin* de transporte: a partir do qual será definido o *plugin* para transporte bem como o protocolo da camada de transporte que será utilizado. Além disso, serão realizadas as configurações iniciais deste mecanismo de adaptação, como o controle de congestionamento, por exemplo;
- Configuração do módulo de codificação multimídia: irá possibilitar a definição do tipo de transmissão a ser realizada (transmissão de áudio ou áudio e vídeo), e do *codec* a ser utilizado. Assim como na inicialização do *plugin* de transporte, algumas adaptações iniciais poderão ser realizadas, dependendo da política de adaptação;
- Configuração do módulo de negociação: esta informação irá possibilitar o carregamento do mecanismo de sinalização de sessão pretendido e sua correta inicialização.

Após o carregamento dos *plugins* supracitados, deverão ser registrados os observadores de eventos, tanto provedores de informações quanto receptores e a política de adaptação. Após a realização desta etapa, a sessão estará apta ao envio e recepção do fluxo de transmissão.

Recuperação e fornecimento de informações dos *plugins* - Conforme discutido na Seção 4.2.2, uma das características presentes em uma política é a configuração de quais componentes da infraestrutura serão responsáveis por prover algumas informações necessárias ao processo de adaptação (através dos nós ***provides***), como informações sobre o comportamento da rede, por exemplo. Tais informações serão usadas para alimentar uma memória de trabalho utilizada pelo processo de inferência de políticas descrito no item abaixo.

Neste sentido, o processo de recuperação das informações será realizado a partir de requisições periódicas aos *plugins*. A periodicidade da busca por tais informações será

variável, de acordo com a política de adaptação. Tal abordagem é necessária, visto que as informações providas necessitam de períodos diferentes de atualização.

Além da recuperação de informações dos *plugins*, o gerenciador de sessões também realiza uma tarefa de notificação aos *plugins*. Dependendo das características do método de adaptação utilizado, um *plugin* pode vir a necessitar de uma alimentação constante de algumas informações de modo a realizar o processo de adaptação corretamente. Por exemplo, na abordagem de adaptação CBR (descrita na Seção 2.3.1), o processo de adaptação consiste na alteração da qualidade do fluxo gerado, objetivando atender a largura de banda corrente da transmissão. Sendo assim, um *plugin* deste tipo necessitaria de uma atualização constante do valor da largura de banda atualmente disponível.

De maneira semelhante à recuperação, o processo de notificação de variáveis aos *plugins* é realizado periodicamente, sendo configurado o tempo de atualização na política de adaptação.

Processo de inferência de políticas - Uma das principais atribuições do módulo gerenciador de sessões, é a realização do processo de tomada de decisão sobre as políticas de adaptação. Para tal, ele realiza o processo de inferência utilizando as regras de adaptação descritas na política de adaptação e uma memória de trabalho, constituída de valores recuperados a partir dos *plugins* de adaptação. Abaixo são apresentadas as etapas realizadas no processo de inferência:

1. Atualização da memória de trabalho a partir da recuperação das informações dos *plugins*. Caso o valor antigo seja diferente do valor atual recuperado, o processo de inferência é disparado;
2. Busca-se os conjuntos de regras que tem como base a variável atualizada e verifica se alguma regra definida torna-se verdadeira para o valor corrente da variável. Em caso afirmativo, são disparados os eventos de adaptação daquela regra específica. Vale ressaltar que no processo de inferência outras variáveis definidas na memória de trabalho podem ser utilizadas como condições na regra;
3. Realiza o processo de notificação da adaptação ocorrida, de modo que o *host* remoto possa se atualizar em relação às novas propriedades da transmissão.

Recuperação, codificação e transmissão fluxo - Por fim, cabe ao gerenciador de sessões o controle do envio e o recebimento de dados da transmissão. Tal controle inclui a recuperação dos dados, codificação (via *plugin* de codificação multimídia) e envio (através do *plugin* de transporte). Por outro lado, no sentido inverso, o dado recebido é decodificado e entregue à aplicação cliente para reprodução.

4.2.4 Módulo de negociação

O módulo de Negociação tem como objetivo realizar o processo de negociação das propriedades da transmissão. De maneira específica, é de responsabilidade deste módulo a troca de informações sobre eventos de adaptação realizados tanto localmente quanto remotamente. O propósito deste mecanismo é possibilitar que *hosts* remotos estejam cientes do processo de adaptação realizados pelo *host* local. Por exemplo, suponha que durante uma transmissão a infraestrutura necessita realizar uma adaptação no fluxo, alterando o *codec* de vídeo utilizado. Deste modo, faz-se necessária a notificação no *host* remoto, de forma que ele possa alterar o processo de decodificação utilizado.

Uma segunda atribuição deste módulo é a possibilidade de, além de receber eventos de adaptação, poder realizar a recuperação de algumas propriedades do *host* remoto. Como discutido na Seção 4.2.2, no processo de descrição de políticas de adaptação, é possível especificar qual componente ficará responsável por prover alguma informação necessária à política, através dos campos **"provides"**. Como consequência, ao se especificar que o *plugin* de negociação deverá fornecer o valor de alguma variável, ele efetuará o procedimento de requisitar ao *host* remoto tal informação. Esta funcionalidade é importante em situações onde a informação necessária para adaptação localiza-se no lado remoto da transmissão.

Na Tabela 4.1 são apresentadas as ações suportadas pelo módulo de negociação e suas descrições.

Tabela 4.1: **Descrição das ações suportadas pelo *plugin* de Negociação**

Nome do Ação	**Host Responsável**	**Descrição**	**Tipo da Mensagem**
startSession	Local	Ação realizada pelo *host* local, solicitando o início de uma transmissão. Nesta operação, o *host* irá informar quais suas configurações iniciais de adaptação.	Requisição

Continuação na próxima página

Tabela 4.1 – continuação da página anterior

Nome do Ação	Host Responsável	Descrição	Tipo da Mensagem
retrieveInformation	Local	Tem como objetivo a recuperação de informações de variáveis no *host* remoto. O tempo de invocação desta operação é definida na política de adaptação.	Requisição
notifyAdaptation	Local	Evento de adaptação, onde o *host* local informa ao *host* remoto sobre alguma adaptação no fluxo efetuada pela infraestrutura.	Notificação
endSession	Local	Requisita a finalização da sessão.	Requisição
handleStartSessionRequest	Remoto	Efetua o tratamento de solicitações de abertura de sessão. Efetua a configuração do fluxo de entrada esperado e efetua seu próprio processo de adaptação, notificando ao *host* remoto (*host* que solicitou a abertura da conexão) as configurações do fluxo, através da ação **notifyAdaptation**. .	Notificação
handleRetrieveInformationRequest	Remoto	Recebe uma solicitação para recuperação de informação de variáveis. Solicita a infraestrutura tal informação e retorna como mensagem de resposta.	Resposta
handleAdaptationRequest	Remoto	Efetua o tratamento de eventos de adaptação realizados no *host* remoto, configurando o fluxo de entrada esperado.	Notificação
handleEndSessionRequest	Remoto	Finaliza a sessão no *host* remoto, informando a infraestrutura a finalização tal evento.	Notificação

Projeto de Baixo Nível

Objetivando possibilitar que diferentes mecanismos de sinalização pudessem ser utilizados como base para negociação de propriedades da transmissão, o Módulo de Negociação foi estruturado na forma de *plugins*, de modo que diferentes abordagens de sinalização pudessem ser utilizadas, tais como: XMPP [53], SIP [29] ou H.245 [3]. Nesta abordagem, a interface-base do *plugin* de negociação será composta pela descrição dos tipos de mensagens desejadas, cabendo a cada *plugin* realizar o transporte e a apresentação das mensagens apresentadas na Tabela 4.1, de acordo com sua estrutura.

4.3 Conclusões do Capítulo

Este capítulo teve como objetivo discutir os principais aspectos da infraestrutura desenvolvida, principal foco deste trabalho. Foram descritos quais são e como é realizada a interação entre os diferentes módulos existentes. Para validar a infraestrutura desenvolvida, foram desenvolvidos estudos de casos abordados no Capítulo 5.

Capítulo 5

Estudos de Casos

Neste capítulo é apresentada a validação da infraestrutura desenvolvida a partir da apresentação de alguns estudos de casos. Conforme apresentado na Seção 4.1, a infraestrutura possui três diferentes papéis relacionados ao processo de desenvolvimento de aplicações multimídia, que são: *i)* desenvolvedor de métodos de adaptação, *ii)* desenvolvedor de políticas de adaptação e *iii)* desenvolvedor de aplicações finais. Em vista disto, dividiu-se este capítulo em duas seções principais. A primeira tem por objetivo demonstrar como a infraestrutura para transmissão de fluxos multimídia pode ser estendida, através da adição de *plugins* de adaptação. Por outro lado, na Seção 5.2 apresenta-se como a infraestrutura pode ser utilizada para a construção de aplicações multimídia. Para tal, foi desenvolvida uma aplicação para transmissão de fluxos de voz, que faz uso da infraestrutura. Para esta aplicação, foi definida uma política de adaptação de fluxos a partir dos *plugins* apresentados a seguir.

5.1 Desenvolvimento de *plugins* de adaptação

Conforme discutido na Seção 4.2.3, para a realização de transmissões multimídia, a infraestrutura faz uso de três tipos de *plugins*, que são: *plugin* para transporte dos dados, *plugin* para codificação multimídia e, finalmente, *plugin* para negociação das propriedades da transmissão. Neste contexto, foram implementados três *plugins*, a fim de serem disponibilizados à aplicação descrita na Seção 5.2. De forma a exemplificar o processo de adaptação de fluxos, foi desenvolvido em cada *plugin*, à exceção do *plugin* para negociação, um conjunto de mecanismos de adaptação. Para utilização de tais mecanismos, cada *plugin* disponibilizou um

conjunto de constantes, de modo que pudesse ser explorado pela infraestrutura, através da política de adaptação utilizada.

A seguir, são apresentados detalhes a respeito dos *plugins* desenvolvidos.

5.1.1 *Plugin* de transporte

Para possibilitar a implementação do *plugin* para adaptação em nível de transporte, foi definida, na infraestrutura, uma interface, chamada **PluginTransportIF**. Esta interface define as ações que um elemento de transporte deve realizar para prover transmissão e adaptação de dados. Na Figura 5.1 é apresentada a estrutura da classe **PluginTransportIF**, sendo seus métodos descritos na Tabela 5.1.

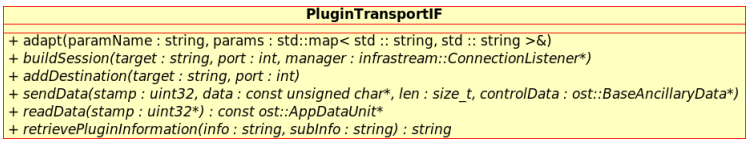


Figura 5.1: Interface do *Plugin* de transporte.

Tabela 5.1: **Descrição da classe base para *plugin* de transporte**

Nome do Método	**Descrição**
adaptTransport	Método responsável pela implementação do processo de adaptação. Este método será invocado pela infraestrutura a cada evento de adaptação realizado. O conjunto de atributos irá variar de acordo com as definições do próprio *plugin*.
buildSession	Realiza a criação de uma sessão servidora, para transmissão. Recebe como parâmetro uma classe observadora que será utilizada para informar à infraestrutura que um transmissor remoto se juntou a sessão.
addDestination	Realiza a criação de uma sessão cliente, para transmissão. Recebe como parâmetro um endereço que irá possibilitar efetuar a correta conexão à sessão remota.
sendData	Realiza o envio de dados pelo canal de transmissão. Além da carga útil transferida, algumas informações de controle devem ser passadas de modo a serem processadas pelas camadas RTP e de transporte.
readData	Método responsável pela leitura dos dados do canal de transmissão.
retrievePluginInformation	Método responsável pela recuperação de informações dos *plugins*. Tais informações serão utilizadas pela infraestrutura para realização do processo de inferência das políticas de adaptação, conforme discutido na Seção 4.2.3.

A partir da interface descrita acima, desenvolveu-se um *plugin* para transmissão que teve como base os protocolos RTP e DCCP de modo a prover o encapsulamento e envio de dados,

a partir da criação de sessões RTP.

O *plugin* desenvolvido provê duas formas de adaptação de fluxo, que são o descarte seletivo de pacotes baseado em políticas e a alteração do mecanismo de controle de congestionamento utilizado pela transmissão, discutidos abaixo.

Descarte seletivo de pacotes baseado em políticas - A principal motivação para a utilização do descarte seletivo de pacotes é possibilitar que técnicas como *Temporização* e *Priorização* de pacotes (discutida nas Seções 2.3.2 e 2.3.2, respectivamente) em nível de transporte possam ser implementadas com sucesso. Para tal, utilizou-se uma proposta para controle de descarte de pacotes chamada *QPolicy*. Este mecanismo possibilita a utilização de diferentes políticas que gerenciam como os pacotes recebidos pela camada de transporte são tratados. Atualmente, encontra-se definida uma política que realiza descarte de pacotes, baseado em prioridade. Nesta política, define-se um limiar de descarte para a fila de envio e quando este limiar é ultrapassado o pacote que possuir a menor prioridade é descartado.

Além disso, encontram-se em processo de implantação duas novas políticas, baseadas no algoritmo discutido em [47]. Este algoritmo combina priorização e temporização de pacotes, tendo obtido bons resultados em transmissões de fluxo de vídeo.

No apêndice B são discutidos maiores detalhes a respeito do *framework QPolicy* e das etapas necessárias para a criação de políticas de descartes.

Alteração dinâmica do mecanismo de controle de congestionamento - O segundo mecanismo de adaptação provido pelo *plugin* é a externalização da escolha do algoritmo de controle de congestionamento utilizado na transmissão. A partir disso, a infraestrutura estará apta a alterar o mecanismo de controle de congestionamento, segundo as definições da política de adaptação.

Com base no apresentado, deu-se suporte aos CCIDs atualmente padronizados, isto é, CCID-2 e CCID-3 e a uma versão experimental do CCID-4. Além disso, visando oferecer um conjunto maior de opções de controle de congestionamento, adicionou-se suporte a dois CCIDs experimentais:

- CCID-0 (*UDP-Like*): não oferece qualquer mecanismo de controle de congestio-

namento. Ele pode ser utilizado em situações onde a largura de banda disponível para envio é superior ao fluxo necessário para transmissão, tendo como benefício um menor *overhead* de transmissão;

- CCID-5 (DCCP *Cubic*): implementação do algoritmo de controle de congestionamento *Cubic*, padrão do TCP do Kernel do Linux, como um novo CCID. O principal objetivo foi possibilitar que aplicações se beneficiem da eficiência do algoritmo *Cubic* em um ambiente sem garantias de entrega de dados.

  No apêndice C são apresentados maiores detalhes a respeito desta nova proposta para CCID desenvolvido dentro do contexto desse trabalho. De maneira específica, são apresentadas, no referido apêndice, a motivação para o desenvolvimento do CCID-5, uma visão geral sobre a implementação e, finalmente, o estágio atual de desenvolvimento.

5.1.2 *Plugin* para codificação multimídia

O segundo *plugin* desenvolvido teve como foco a exemplificação do processo de codificação de fluxo multimídia, em especial, a codificação de fluxos de áudio. Para tal, considerou-se a implementação de um *plugin* que possibilitasse adaptar o fluxo de acordo com a largura de banda disponível para envio, através da utilização de mecanismos como VBR, CBR etc. Em vista disso, utilizou-se como base o formato de codificação de fluxos de áudio ***Speex***.

O padrão Speex [70] é um formato aberto para compressão de pacotes de áudio, sendo indicado especialmente para a compactação de pacotes de voz. Este *codec* foi projetado tendo como ambiente de transmissão alvo um meio onde não há garantia de entrega de informações, como a Internet. Uma das grandes vantagens da utilização do Speex na codificação de fluxos de áudio é a sua grande flexibilidade, suportando diversos níveis de qualidade de codificação e taxa gerada, de modo a atender aos requisitos de transmissão. Além disso, o Speex possibilita uma grande customização, via biblioteca de programação (chamada ***libspeex***), do modo como o *codec* irá operar. Desta forma, tornou-se possível a realização de um grande conjunto de adaptações no fluxo gerado. A seguir são apresentados os principais mecanismos de adaptação, que são providos pela biblioteca *Speex* e que foram adicionados no *plugin* desenvolvido:

- Escolha do método de codificação utilizado (CBR, VBR ou ABR);
- Escolha da qualidade máxima pretendida;
- Otimização do processo de codificação em relação à taxa de perda de pacotes;
- Especificação da taxa de bit máxima permitida para codificação;
- Definição da complexidade máxima possível na codificação. A alteração deste valor repercute tanto na qualidade quanto na taxa gerada de bits.

O ciclo de execução deste módulo é constituído basicamente pela criação de uma sessão de codificação - onde são definidos os parâmetros básicos para codificação, pelo processo de codificação e decodificação de amostras de dados (que pode ser alterado por eventos de adaptação) e, finalmente, pela finalização da sessão, quando os recursos são desalocados. Na Tabela 5.2 são descritas as principais ações providas pelo *plugin* de codificação multimídia.

Tabela 5.2: **Descrição das principais propriedades da classe base para *plugin* de codificação multimídia**

Nome do Método	Descrição
buildSession	Método responsável pela criação de uma sessão de codificação. O tipo da sessão irá depender dos parâmetros recebidos.
endSession	Finaliza a sessão de codificação de fluxo, liberando os recursos alocados.
adapt	Método responsável pela implementação do processo de adaptação. Este método será invocado pela infraestrutura a cada evento de adaptação realizado. O conjunto de atributos irá variar de acordo com as definições do próprio *plugin*.
encode	Realiza a codificação de um *buffer* de dados. Além do dado codificado, tem como retorno possivelmente um conjunto de informações de controle de forma a possibilitar uma adaptação no envio da informação.
decode	Processo inverso à codificação, sendo utilizado para tratamento dos dados recebidos na transmissão.
retrievePluginInformation	Método responsável pela recuperação de informações dos *plugins*. Assim como no *plugin* de transporte, tais informações serão utilizadas pela infraestrutura para realização do processo de inferência das políticas de adaptação.

5.1.3 *Plugin* para negociação

Conforme discutido anteriormente, o *plugin* de negociação tem por objetivo realizar a negociação das propriedades de transmissão, notificando os *hosts* remotos a respeito das al-

terações ocorridas na transmissão. Para exemplificação do processo de desenvolvimento do *plugin* de negociação, utilizou-se como base o protocolo para sinalização e troca de mensagens, XMPP.

O XMPP [1] [53] é um protocolo aberto para comunicação em tempo real que pode ser utilizado em diversos tipos de aplicações, incluindo mensagens instantâneas e sinalizadores de presença. Uma das grandes vantagens deste protocolo é que ele é baseado em XML, possibilitando uma grande extensibilidade, de forma a ser utilizado aos mais diversos propósitos. Por isso, o XMPP tem sido amplamente difundindo, sendo implementado por um grande conjunto de aplicações, como Google Talk [2], Kopete [3] e Pidgin [4]. O processo de comunicação neste protocolo é baseado no conceito de cliente/servidor, onde se define um servidor, responsável por realizar processo de sinalização de presença, de modo que os clientes possam identificar o endereço de origem de outros clientes e iniciar uma transmissão diretamente.

Em modo a contornar tal limitação, definiu-se que em cada transmissor e receptor deve haver a presença de um servidor XMPP, de modo que quando a sessão for iniciada, o cliente local automaticamente se conecta neste servidor, ficando no aguardo pela conexão do cliente remoto.

5.2 Desenvolvimento de aplicação para transmissão de áudio

Esta seção tem por objetivo apresentar uma aplicação desenvolvida a partir da infraestrutura para adaptação de fluxos multimídia. Tal aplicação faz uso da infraestrutura desenvolvida para possibilitar chamadas de voz entre dois usuários.

Como o principal objetivo do desenvolvimento desta aplicação foi validar a infraestrutura desenvolvida, simplificou-se o desenvolvimento da aplicação, através da eliminação de alguns procedimentos, que em geral, são realizados antes do início da transmissão, como o processo de sinalização para descoberta dos endereços de transmissão, realizados pelo protocolo SIP, por exemplo. No cenário implementado, considerou-se que tais informações já

[1] Abreviação de *Extensible Messaging and Presence Protocol*.

[2] `http://www.google.com/talk/`

[3] `http://kopete.kde.org/`

[4] `http://www.pidgin.im/`

seriam conhecidas *a priori*.

Apesar do processo de definição de políticas de adaptação fazer parte de uma etapa anterior ao desenvolvimento de aplicações finais, optou-se, neste estudo de caso, por apresentá-los conjuntamente. Com base nesta premissa, o passo inicial para o desenvolvimento da aplicação pretendida foi a estruturação da política de adaptação a ser utilizada. Esta política utiliza como base os *plugins* descritos na sessão anterior, que são:

- *Plugin* baseado no protocolo DCCP para transmissão;
- *Plugin* de negociação, baseado no protocolo XMPP;
- *Plugin* para codificação multimídia de fluxos de áudio, baseado na biblioteca Speex.

O passo seguinte à definição do conjunto de *plugins* que serão utilizados na política de adaptação é especificar o tipo da transmissão que a política descreve (isto é, se é uma transmissão de áudio ou vídeo). Para tal, deve-se configurar o tipo de transmissão pretendido, na política de adaptação, através da utilização de uma das seguintes entradas **video-transmission** ou **audio-transmission**, como ilustrado na Figura 5.2.

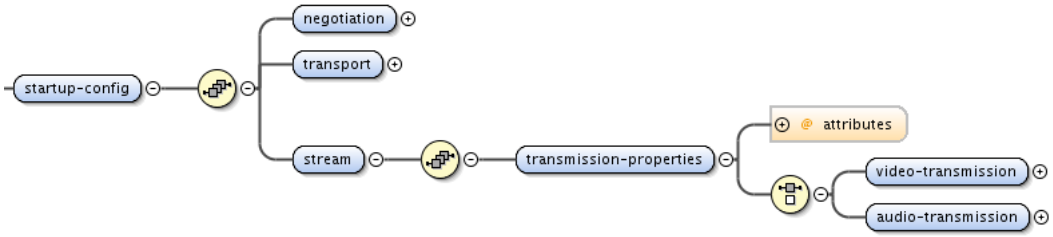


Figura 5.2: Definição do tipo de transmissão na política de adaptação.

A etapa final de descrição da política de adaptação consiste na definição dos eventos de adaptação pretendidos. Conforme descrito na Seção 4.2.2, para cada variável monitorada será definido um conjunto de eventos de adaptação, conforme apresentado na Listagem 4.1. Na política de adaptação apresentada na listagem supracitada considerou-se, entre outros, a utilização dos valores de taxa de envio corrente e taxa de perdas de dados da transmissão, onde em eventos em que a taxa de envio tornar-se menor que 100kpbs, um conjunto de ações de adaptação deverá ser disparado.

```
1        <stream>
```

```
2           <adaptation-method>Speex</adaptation-method>
3
4               ...
5
6           <policy>
7               <comparisson-attibute>sendrate</comparisson-attibute>
8               <lower threshold="100">
9                   <simpleproperty key="SpeexComplexity">9</
                        simpleproperty>
10                  <!-- aumentando complexidade de codificação, para
                        diminuir largura de banda necessaria -->
11
12                  <simpleproperty key="SpeexVAD">1</simpleproperty>
13                  <!-- Habilitando detectação de silencio, para evitar
                        envio desnecessario de dados -->
14
15                  <simpleproperty key="SpeexMode">NarrowBand</
                        simpleproperty>
16                  <!-- Configura a utilização do modo de codificação
                        que consome menor quantidade de dados, mas tendo
                        perda de qualidade  -->
17
18              </lower>
19          </policy>
20          <policy>
21              <comparisson-attibute>lossrate</comparisson-attibute>
22              <greater threshold="0.1">
23                  <simpleproperty key="SpeexPLCTuning">0.1</
                        simpleproperty>
24                  <!-- Otimiza codificador para quando taxa de perda
                        exceder 10%  -->
25              </greater>
26          </policy>
27
28              ...
29
30      </stream>
```

```
31        <transport>
32                <policy>
33                        <comparisson-attibute>lossrate </comparisson-
                              attibute>
34                        <greater threshold="0.25">
35                                <simpleproperty key="DCCPTXCCID">5</
                                      simpleproperty>
36                        </greater>
37                </policy>
38        </transport>
```

Listagem 4.1: Critérios de adaptação definidos para transmissão de áudio. As configurações dos demais parâmetros para adaptação foram omitidas.

Após a definição da política, o próximo passo é a criação da sessão de transmissão. A criação de uma sessão é realizada a partir da invocação do método **createClientSession**. Neste método, são instanciados e configurados os *plugins* de transporte, negociação e codificação, segundo as informações extraídas da política de adaptação recebida por parâmetro, conforme ilustrado na Listagem 4.2. Após a etapa de configuração dos *plugins*, tem início a captura e recepção do fluxo multimídia, onde o controle do fluxo de execução fica por conta da infraestrutura. Para tal, a aplicação desenvolvida deverá fornecer à infraestrutura objetos responsáveis pela captura e reprodução dos dados dos dispositivos (**GenericSenderSocket** e **GenericReceiverSocket**, respectivamente). A partir disso, há uma abstração da fonte de dados que a infraestrutura irá acessar, conforme apresentado na Figura 5.3.

```
1 SessionManager* Facade::createClientSession(string targetIp, int
      targetPort, string controlPort
2                 GenericSenderSocket* sender, GenericReceiverSocket*
                      receiver,
3                 string policyPath, map<string, string> * additionalParams
                      )
4                 throw(CannotCreateSessionException,
                      CannotLoadPolicyException) {
5
6         PolicyEngine* engine = 0;
7         SessionManager* session = 0;
8         PolicyConfigurationType* policyDesc = 0;
9         TransportSession* trSession = 0;
```

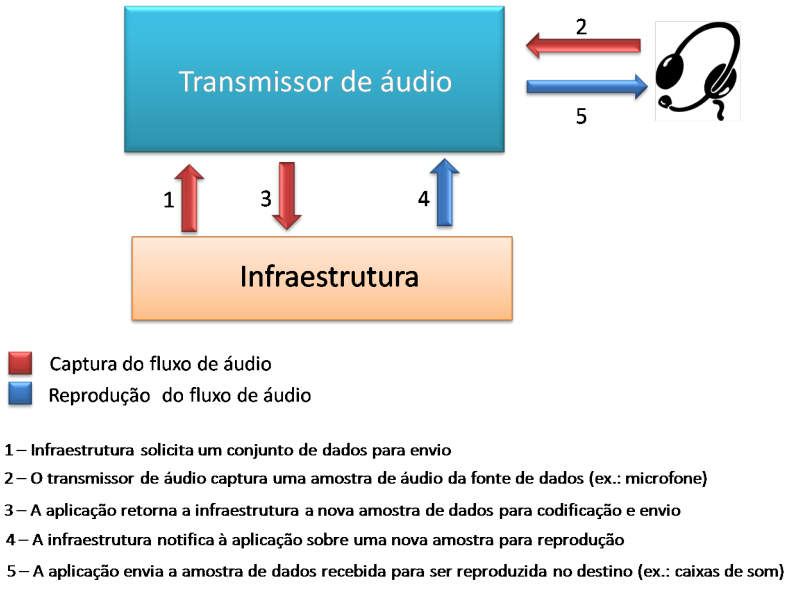


Figura 5.3: Fluxo de captura e entrega de dados à aplicação final

```
10          StreamSession* ssession = 0;
11
12          try {
13                  policyDesc = new PolicyConfigurationType(*(loadPolicy(
                        policyPath).get()));
14
15                  engine = new PolicyEngine;
16                  session = new SessionManager(sender, receiver, engine,
                        policyDesc);
17
18
19                  PluginNegotiationPtrlIF* negotiation =
20                                          InfraFactory::getInstance()->
                                                buildNegotiationSession(
                                                targetIp, controlPort);
21
22                  session->setNegotiationSession(negotiation);
23
24                  trSession = InfraFactory::getInstance()->
                        buildTransportSession(0,
25                                  policyDesc, *engine, targetIp, targetPort
                                        ,
```

```
26                                  InfraFactory::CLIENT_SESSION, session);
27
28                  session->setTransportSession(trSession);
29
30                  ssession = InfraFactory::getInstance()->
                        buildStreamSession(0,
31                                  policyDesc, *engine);
32
33                  session->setStreamSession(ssession);
34
35                  session->startSession();
36          } catch (...) {
37                  if (session) {
38                          delete session;
39                          log_error("Exception on creating session.
                                Deleting Session manager object");
40                  }
41
42                  if (engine) {
43                          delete engine;
44                          log_info("Exception on creating session. Deleting
                                engine");
45                  }
46
47                  if (policyDesc) {
48                          delete policyDesc;
49                          log_error("Exception on creating session.
                                Deleting plugin object");
50                  }
51
52                  throw;
53          }
54          return session;
55  }
```

Listagem 4.2: Criação de uma sessão de transmissão.

Com base nos passos apresentados acima, foi possível a construção da aplicação, apresentada na Figura 5.4. Nesta aplicação o usuário final está apto a iniciar e finalizar uma

sessão de transmissão, a partir do endereço do usuário remoto ao qual ele deseja se conectar.

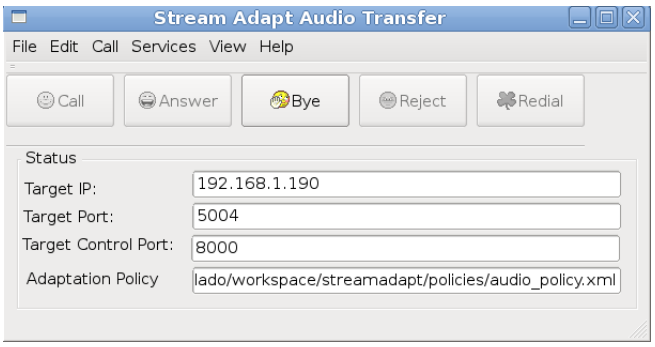


Figura 5.4: Tela da aplicação de transmissão de fluxos de áudio.

A partir da utilização da aplicação para transmissão de áudio desenvolvida, foram realizadas transmissões em um ambiente que emulasse as condições de transmissão semelhante ao da Internet, utilizando-se para tal, a ferramenta Netem. O Netem [21] é uma ferramenta implementada como um módulo no kernel do Linux que possibilita a definição de disciplinas de escalonamento de pacotes, de modo a possibilitar a emulação de ambientes de transmissão real, como a Internet. Esta ferramenta permite emular eventos na rede, tais como: atrasos, *jitter*, perdas de dados, corrupção de pacotes etc.

Para a realização das transmissões, foi montado um cenário de transmissão composto por: *i)* um *host* transmissor, a partir do qual se utilizou a infraestrutura para a realização do processo de adaptação multimídia por meio da política desenvolvida; *ii)* um *host* receptor, que irá receber o fluxo transmitido, reportando os devidos relatórios da transmissão; *iii)* e um terceiro *host*, responsável por simular o comportamento de um roteador de repasse de pacotes e sob o qual foi aplicado as disciplinas de escalonamento de pacotes do Netem. Na Figura 5.5 é ilustrado o cenário desenvolvido, onde o *host* **transmissor** e o **receptor** estão conectados através do *host* que desempenha papel de roteador.

Figura 5.5: Ambiente de execução da infraestrutura.

A partir do cenário construído, foram realizadas transmissões entre os *hosts* especificados, emulando-se, no decorrer da transmissão, a alteração da taxa de perda de pacotes, de

modo a verificar se eventos de adaptação foram lançados de acordo com o especificado na política de adaptação. Para tal, definiu-se dois eventos de alteração desta variável: o primeiro executado no trigésimo segundo de transmissão, ajustando o valor da taxa de perda de pacotes para 15%; e um segundo evento de alteração desta variável, realizado no sexagésimo segundo, reajustando a variável supracitada para 30% de perdas. Como consequência, a infraestrutura realizou a detecção da alteração das condições de transmissão e efetuou o lançamento de eventos de adaptação, de acordo com o especificado na política de adaptação, apresentado na Listagem 4.1. No primeiro evento de adaptação disparado pela infraestrutura, foi realizado a alteração da variável **SpeexPLCTuning**, de acordo com o especificado na política de adaptação, que determina a alteração desta variável quando a taxa de perda for superior a 10% (Listagem 4.3).

```
1 <policy>
2         <comparisson-attibute>lossrate</comparisson-attibute>
3         <greater threshold="0.1">
4                 <simpleproperty key="SpeexPLCTuning">0.1</simpleproperty>
5                 <!-- Otimiza codificador para quando taxa de perda
                      exceder 10% -->
6         </greater>
7 </policy>
```

Listagem 4.3: Primeiro evento de adaptação disparado pela infraestrutura.

Por outro lado, quando a taxa de perda de pacotes foi elevada à 30% (após 60 segundos de transmissão), o segundo evento de adaptação foi disparado pela infraestrutura. Neste evento de adaptação foi realizado a alteração do mecanismo de controle de congestionamento, passando-se a utilizar o CCID-5 (Listagem 4.4).

```
1 <policy>
2         <comparisson-attibute>lossrate</comparisson-attibute>
3         <greater threshold="0.25">
4                 <simpleproperty key="DCCPTXCCID">5</simpleproperty>
5         </greater>
6 </policy>
```

Listagem 4.4: Segundo evento de adaptação disparado pela infraestrutura.

5.3 Conclusões do Capítulo

Este capítulo teve como objetivo validar a infraestrutura desenvolvida sob as perspectivas dos diferentes de atores envolvidos no processo de desenvolvimento de aplicações multimídia que oferecem suporte à adaptações multimídia. Para tal, buscou-se demonstrar como o processo de adição de métodos de adaptação, pode ser realizado, de modo a permitir a utilização de diferentes abordagens de adaptação. Por outro lado, mostrou-se como a infraestrutura pode ser utilizada para o desenvolvimento de aplicações multimídia, de modo a reduzir a complexidade envolvida em tal tarefa. Neste sentido, foi desenvolvido uma aplicação para transmissão de fluxos de voz, que fez uso da infraestrutura para transmissão de fluxos multimídia a partir da interface de programação disponibilizada. Como consequência pôde-se demonstrar o processo de adaptação realizado pela infraestrutura durante transmissões.

Capítulo 6

Conclusões e Trabalhos Futuros

As aplicações multimídia têm recebido grande atenção nos últimos anos, devido à sua grande aplicabilidade, sendo utilizadas nos mais diversos domínios. Todavia, a natureza de "melhor esforço" da Internet torna o processo de transmissão de fluxo multimídia uma tarefa complexa, devido a diversos requisitos presentes neste tipo de aplicação. Como consequência, faz-se necessária a utilização de mecanismos que realizem a adaptação do fluxo multimídia de forma a se adequear as condições da transmissão.

Neste trabalho, foi apresentada uma infraestrutura para o desenvolvimento de aplicações multimídia com suporte à adaptação de fluxo. Nesta infraestrutura, foram identificados os diferentes papéis relacionados ao ciclo de desenvolvimento deste tipo de aplicação, que são: desenvolvimento de métodos de adapação, desenvolvimento de políticas de adaptação e, finalmente, desenvolvimento de aplicações finais. A partir da identificação dos papéis supracitados, foi possível o desenvolvimento de uma estrutura que permitisse a separação entre as responsabilidades de cada papel, onde:

- a inserção de novos métodos de adaptação é realizada a partir de uma infraestrutura baseada em *plugins*, possibilitando que novos métodos de adaptação possam ser acoplados facilmente à infraestrutura, simplesmente implementando uma interface de acesso pré-definida;

- o desenvolvimento de políticas de adaptação passa a ser realizado a partir de um modelo declarativo, onde as propriedades das políticas de adaptação são especificadas num arquivo XML, facilitando tal processo;

- uma interface simplificada de desenvolvimento é disponibilizada pela infraestrutura, de modo que desenvolvedores possam implementar as aplicaçoes finais. Nesta interface, não é oferecido acesso aos métodos de adaptação existente, visto que o processo de adaptação será realizado internamente pela infraestrutura, a partir de uma política de adaptação definida pela aplicação cliente.

A partir da utilização da infraestrutura desenvolvida, o processo de desenvolvimento de aplicações multimídia tornar-se-á menos complexo, visto que o desenvolvedor final estará livre de uma série de responsabilidades, que muitas vezes está além do seu conhecimento. Além disso, a separação entre a política de adaptação adotada e o real processo de adaptação possibilita uma personalização de como o processo de adaptação multimídia será realizado, permitindo, por exemplo, uma adaptação que leve em consideração as restrições dos dispositivos envolvidos na transmissão.

Como contribuições marginais a este trabalho, vale salientar as contribuições realizadas às ferramentas que dão suporte ao desenvolvimento de aplicações multimídia, tanto em nível de bibliotecas de programação quanto de módulos para o Kernel do Linux. Especificamente pode-se listar:

- implementação do algoritmo de controle de congestionamento *Cubic* no protocolo DCCP;

- contribuições no desenvolvimento do algoritmo de controle de congestionamento CCID-4;

- a adição de novas funcionalidades ao framework *QPolicy*, também implementado no Kernel do Linux;

- contribuição no desenvolvimento de algumas bibliotecas de código aberto, utilizados pelos *plugins* de adaptação, a partir da adição do suporte ao protocolo DCCP. De maneira específica, pode-se citar a melhoria do suporte oferecido pela biblioteca que oferece recursos de *sockets* de alto nível, CommonCpp (`http://www.gnu.org/software/commoncpp/`), através da adição de novos recursos; e adição do suporte à biblioteca ccRTP (`http://www.gnu.org/software/ccrtp/`).

Como trabalhos futuros pretende-se realizar um estudo a respeito do desenvolvimento de políticas de adaptação otimizadas. Tal estudo terá como base as característsticas dos métodos de adaptação, de forma a verificar sob quais situações um método de adaptação mostra-se mais eficaz.

Em paralelo, pretende-se investigar como a infraestrutura desenvolvida pode ser utilizada em ambientes de distribuição massiva de conteúdo, de maneira escalável. Como exemplo, pode-se apontar a distribuição de fluxos IPTV para diversos clientes. Neste contexto, técnicas como transmissões *multicast* podem vir a ser trabalhadas, de forma a possibilitar a escalabilidade do modelo desenvolvido.

Bibliografia

[1] Internet Protocol. http://www.faqs.org/rfcs/rfc791.html, Setembro 1981. Último acesso Janeiro 2010.

[2] TCP: Tranmission Control Protocol, 9 1981. Último acesso em Outubro 2009.

[3] H.245 : Control protocol for Multimedia Communication. Recomendação itu-t, International Telecommunication Union, Junho 2008. Último acesso Janeiro 2010.

[4] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. http://www.faqs.org/rfcs/rfc3390.html, Outubro 2002. Último acesso Janeiro 2010.

[5] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. http://www.ietf.org/rfc/rfc2581.txt., Abril 1999. Último acesso Janeiro 2010.

[6] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. The Secure Real-time Transport Protocol (SRTP). http://tools.ietf.org/html/rfc3711, Março 2004. Último acesso Janeiro 2010.

[7] Abderrahim Benslimane, editor. *Multimedia Multicast on the Internet*. ISTE, 2007.

[8] Dorian Birsan. On Plug-ins and Extensible Architectures. *Queue*, 3(2):40–46, 2005.

[9] R. Braden. Requirements for Internet Hosts - Communication Layers. http://www.faqs.org/rfcs/rfc1122.html, Outubro 1989. Último acesso Janeiro 2010.

[10] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. *SIGCOMM'94, London, U.K.*, 1:24–35, 10 1994.

[11] Torsten Braun, Michel Diaz, Jos Enrquez-Gabeiras, and Thomas Staub. *End-to-End Quality of Service Over Heterogeneous Networks*. Springer Publishing Company, Incorporated, 2008.

[12] Codesynthesis. Xsd: Xml data binding for c++. http://www.codesynthesis.com/products/xsd/, 2009. Último acesso Janeiro 2010.

[13] G.J. Conklin and S.S. Hemami. A Comparison of Temporal Scalability Techniques. *Circuits and Systems for Video Technology, IEEE Transactions on*, 9(6):909–919, Sep 1999.

[14] Leandro Melo de Sales. Avaliação Experimental do Protocolo DCCP para Transmissão de Conteúdos Multimídia em Redes sem Fio 802.11g e na Internet. Master's thesis, Universidade Federal de Campina Grande, 2008.

[15] S. Floyd, M. Handley, and E. Kohler. Problem Statement for the Datagram Congestion Control Protocol (DCCP). http://www.ietf.org/rfc/rfc4336.txt, Março 2006. Último acesso Janeiro 2010.

[16] S. Floyd, M. Handley, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. http://tools.ietf.org/html/rfc5348, Setembro 2008. Último acesso Janeiro 2010.

[17] S. Floyd and E. Kohler. Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control. http://www.ietf.org/rfc/rfc4341.txt, Março 2006. Último acesso em Outubro 2009.

[18] S. Floyd and E. Kohler. TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant. http://tools.ietf.org/html/rfc4828, Abril 2007. Último acesso Janeiro 2010.

[19] S. Floyd, E. Kohler, and J. Padhye. Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC). http://www.ietf.org/rfc/rfc4342.txt, Março 2006. Último acesso em Outubro 2009.

[20] Sally Floyd and Eddie Kohler. Profile for Datagram Congestion Control Protocol (DCCP) Congestion ID 4: TCP-Friendly Rate Control for Small Packets (TFRC-SP). http://tools.ietf.org/html/rfc5622, Agosto 2009. Último acesso Outubro 2009.

[21] Linux Foundation. netem - network emulation. http://www.linuxfoundation.org/collaborate/workgroups/networking/netem. Último acesso Fevereiro 2010.

[22] Cheng Peng Fu and Soung C. Liew. TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks. *IEEE Journal On Selected Areas In Communications*, 21(2):216–228, 2 2003.

[23] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Padrões de Projeto: Soluções Reutilizáeis de Software Orientado a Objetos*. Bookman, 1995.

[24] Ladan Gharai. RTP with TCP Friendly Rate Control. http://tools.ietf.org/html/draft-ietf-avt-tfrc-profile-10, Julho 2007. Último acesso Janeiro 2010.

[25] Eren Gurses, Gozdel Bozdag Akar, and Nail Akar. Selective Frame Discarding for Video Streaming in TCP/IP Networks. In *Packet Video Workshop, Nantes, France*, 2003.

[26] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: a new TCP-Friendly high-speed TCP Variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, 2008.

[27] M. Handley, S. Floyd, J. Padhye, and J. Widmer. Tcp friendly rate control (tfrc): Protocol specification. http://www.ietf.org/rfc/rfc3448.txt, Janeiro 2003. Último acesso Janeiro 2010.

[28] M. Handley and V. Jacobson. Sdp: Session Description Protocol. http://www.faqs.org/rfcs/rfc2327, Abril 1999. Último acesso em Fevereiro 2010.

[29] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. SIP: Session Initiation Protocol. http://www.ietf.org/rfc/rfc2543.txt, 3 1999. Último acesso em Outubro 2009.

[30] G. Herlein, J. Valin, A. Heggestad, and A. Moizard. Rtp payload format for the speex codec. http://tools.ietf.org/html/rfc5574, Junho 2009. Último acesso Janeiro 2010.

[31] Olivier Hersent, Jean-Pierre Petit, and David Gurle. *Beyond VoIP Protocols: Understanding Voice Technology and Networking Techniques for IP Telephony (Hardcover)*. Wiley, 1 edition, March 2005.

[32] Sven Hessler and Michael Welzl. Seamless Transport Service Selection by Deploying a Middleware. *Elsevier Computer Communications*, 30(3):630–637, February 2007. Special Issue on Emerging Middleware for Next Generation Networks.

[33] Geoff Huston. Gigabit TCP. *The Internet Protocol Journal*, 9(2):2–26, Junho 2006.

[34] A. Huszak and S. Imre. Source Controlled and Delay Sensitive Selective Retransmission Scheme for Multimedia Streaming. *12th IEEE Symposium on Computers and Communications (ISCC 2007)*, 1:191–196, July 2007.

[35] V. Jacobson. Congestion Avoidance and Control. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 314–329, New York, NY, USA, 1988. ACM.

[36] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. http://www.ietf.org/rfc/rfc1323.txt, Maio 1992. Último acesso Janeiro 2010.

[37] Y. Kikuchi, T. Nomura, S. Fukunaga, Y. Matsui, and H. Kimata. Rtp payload format for mpeg-4 audio/visual streams. http://tools.ietf.org/html/rfc3016, Novembro 2000. Último acesso Janeiro 2010.

[38] Sangki Ko, Seoungjun Oh, and Kwangsue Chung. An Efficient QoS Negotiation Mechanism for IPTV Service in Heterogeneous Networks. In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, pages 575–579, Aug. 2009.

[39] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). http://www.rfc-editor.org/rfc/rfc4340.txt, March 2006. Último acesso em Outubro 2009.

[40] Michael Kropfberger. Multimedia Streaming Over Best Effort Networks Using Multi-Level Adaptation and Buffer Smoothing Algorithms. Master's thesis, Universit¨at Klagenfurt, 2004.

[41] James F. Kurose and Keith W. Ross. *Redes de Computadores e a Internet: Uma Abordagem Top-Down*. Addison Wesley, São Paulo, trad. 3 ed. edition, 2006.

[42] L-A. Larzon, M. Degermark, S. Pink, L-E. Jonsson, and G. Fairhurst. The Lightweight User Datagram Protocol (UDP-Lite). http://www.faqs.org/rfcs/rfc3828.html, Julho 2004. Último acesso Janeiro 2010.

[43] Jack Lee. *Scalable Continuous Media Streaming Systems: Architecture, Design, Analysis and Implementation (Hardcover)*. Wiley, 2005.

[44] Sunhun Lee and Kwangsue Chung. Combining the Rate Adaptation and quality Adaptation Shemes for Wireless video streaming. *J. Vis. Comun. Image Represent.*, 19(8):508–519, 2008.

[45] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. http://www.faqs.org/rfcs/rfc2018.html, Outubro 1996. Último acesso Janeiro 2010.

[46] Johannes Mayer, Ingo Melzer, and Franz Schweiggert. Lightweight Plug-in-based Application Development. In *NODe '02: Revised Papers from the International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World*, pages 87–102, London, UK, 2003. Springer-Verlag.

[47] Ian McDonald and Richard Nelson. Application-Level QoS: Improving Video Conferencing Quality through Sending the Best Packet Next. *Next Generation Mobile Applications, Services and Technologies, International Conference on*, 0:507–513, 2008.

[48] P. Mockapetris. Domain Names - Implementation and Specification. http://www.faqs.org/rfcs/rfc1035.html, Novembro 1987. Último acesso Janeiro 2010.

[49] Simon Morlat. ortp, a Real-time Transport Protocol (RTP,RFC3550) library. http://www.linphone.org/index.php/eng/code_review/ortp, Setembro 2006. Último acesso Janeiro 2010.

[50] Ostermann J. Nguyen, D. T. Streaming and Congestion Control using Scalable Video Coding based on H.264/AVC. In *Proceedings of the 15th International Packet Video Workshop Journal of Zhejiang University - Science A*, volume 7, pages 749–754, Abril 2006.

[51] Frederic Nivor. Experimental Study of DCCP for Multimedia Applications. In *CoNEXT '05: Proceedings of the 2005 ACM conference on Emerging network experiment and technology*, pages 272–273, New York, NY, USA, 2005. ACM.

[52] J. Ott, S. Wenger, N. Sato, C. Burmeister, and J. Rey. Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-based Feedback (RTP/AVPF). http://tools.ietf.org/html/rfc4585, Julho 2006. Último acesso em Janeiro 2010.

[53] Ed P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. http://www.ietf.org/rfc/rfc3920.txt, Outubro 2004. Último acesso em Outubro 2009.

[54] Colin Perkins. *RTP: Audio and Video for the Internet*. Addison-Wesley Professional, 2003.

[55] J. Postel. UDP: User Datagram Protocol. http://tools.ietf.org/html/rfc0768, 8 1980. Último acesso Janeiro 2010.

[56] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. http://www.ietf.org/rfc/rfc3168.txt, Setembro 2001. Último acesso Janeiro 2010.

[57] J. Rey, D. Leon, A. Miyazaki, V. Varsa, and R. Hakenberg. RFC 4588: RTP Retransmission Payload Format. http://www.ietf.org/rfc/rfc4588.txt, Julho 2006. Último acesso em Janeiro, 2010.

[58] Iain Richardson and Iain E. G. Richardson. *H.264 and MPEG-4 Video Compression: Video Coding for Next Generation Multimedia*. Wiley, 2003.

[59] Valter Roesler. *Sam: Um Sistema Adaptativo para Transmissão e Recepção de Sinais Multimídia em Redes de Computadores*. PhD thesis, Universidade Federal do Rio Grande do Sul, 2003.

[60] Scott Ruthfield. The Internet's History and Development From Wartime Tool to the Fish-Cam. http://www.acm.org/crossroads/xrds2-1/inet-history.html. Último acesso em Outubro 2009.

[61] Mihaela van der Schaar and Philip A. Chou. *Multimedia over IP and Wireless Networks: Compression, Networking, and Systems*. Academic Press, Inc., Orlando, FL, USA, 2007.

[62] H. Schulzrinne and S. Casner. RTP Profile for Audio and Video Conferences with Minimal Control. http://tools.ietf.org/html/rfc3551, Julho 2003. Último acesso Janeiro 2010.

[63] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-time Applications. http://www.ietf.org/rfc/rfc3550.txt, 7 2003. Último acesso Outubro 2009.

[64] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol. http://www.ietf.org/rfc/rfc2326, Abril 1998. Último acesso Janeiro 2010.

[65] Wes Simpson. *Video Over IP IPTV, Internet Video, H.264, P2P, Web TV, and Streaming: A Complete Guide to Understanding the Technology*. Elsevier, 2008.

[66] N. Spring and D. Ely. Robust Explicit Congestion Notification (ECN) Signaling with Nonces. http://www.faqs.org/rfcs/rfc3540.html, Junho 2003. Último acesso Janeiro 2010.

[67] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. http://www.faqs.org/rfcs/rfc2001.html, Janeiro 1997. Último acesso Janeiro 2010.

[68] David Sugar, Federico Montesino Pouzols, and Werner Dittman. GNU ccRTP. http://www.gnu.org/software/ccrtp/, Novembro 2002. Último acesso Janeiro 2010.

[69] A. Szwabe, A. Schorr, F. J. Hauck, and A. J. Kassler. Dynamic Multimedia Stream Adaptation and Rate Control for Heterogeneous Networks. In *15th International Packet Video Workshop, (PV'06)*, volume 7, pages 63–69, Maio 2006.

[70] Jean-Marc Valin. *The Speex Codec Manual*. Xiph.Org Foundation, 1.2 beta 3 edition, Dezembro 2007. Último acesso Janeiro 2010.

[71] P. Vate-U-Lan. Mobile Learning: Major Challenges for Engineering Education. pages T4F–11–T4F–16, Oct. 2008.

[72] Ricardo N. Vaz and Mário S. Nunes. Selective Frame Discard for Video Streaming over IP Networks. In *Proceedings of the 7th Conference on Computer Networks*, 2004.

[73] W3C. Extensible Markup Language (XML). http://www.w3.org/XML/, Novembro 2008. Último acesso em Janeiro 2010.

[74] Michael Welzl. A Case for Middleware to Enable Advanced Internet Services. In *Proceedings of Next Generation Middleware Workshop (NGNM'04)*, Atenas, Grécia, Maio 2004. Springer.

[75] Michael Welzl. *Network Congestion Control: Managing Internet Traffic*. Wiley, 2005.

[76] S. Wenger, M.M. Hannuksela, T. Stockhammer, M. Westerlund, and D. Singer. RTP Payload Format for H.264 Video. http://tools.ietf.org/html/rfc3984, Fevereiro 2005. Último acesso Janeiro 2010.

[77] Lisong Xu, K. Harfoush, and Injong Rhee. Binary Increase Congestion Control (BIC) for Fast Long-distance Networks. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2514–2524 vol.4, March 2004.

Apêndice A

Configuração de políticas de adaptação

Neste apêndice é apresentada uma descrição detalhada dos parâmetros de configuração de políticas de adaptação, bem como um exemplo de uma política de adaptação.

A.1 Parâmetros de políticas de adaptação

Na Tabela A.1 é apresentada uma descrição dos principais campos presente em uma política de adaptação.

Tabela A.1: **Atributos de uma política de adaptação**

Nome do Nó	**Nó Base**	**Descrição**	**Possui nós filhos?**	**Granularidade**
policy-name	/	Identifica o nome da política. Cada política terá como foco um tipo de aplicação, voltado para cenários específicos. Por exemplo, aplicações VoIP voltado para dispositivos embarcados.	Não.	1
policy-description	/	Possibilita a descrição do propósito da política. Tal campo é utilizado com propósito de documentação.	Não.	1
startup-config	/	Neste nó será inserido as configurações iniciais para uma transmissão, conforme já discutido. Para tal, faz-se necessário a definição de três nós filhos de configuração, que irão servir para configurar os *plugins* de Adaptação, Transporte e Negociação.	Sim.	1
negotiation	*/startup-config*	Define as propriedades utilizadas para inicialização do canal de negociação. Este canal deve ser um canal fora de banda, de preferência com serviço de entrega confiável.	Sim.	1

Continuação na próxima página

Tabela A.1 – continuação da página anterior

Nome do Nó	**Nó Base**	**Descrição**	**Possui nós filhos?**	**Granularidade**
transport	*/startup-config*	Define as propriedades de configuração do *plugin* para transporte. A partir destas propriedades, será possível estabelecer o canal de configuração, incluindo informações de protocolo de transporte, algoritmos de controle de congestionamento, se disponível, etc.	Sim.	1
stream	*/startup-config*	Possibilita a descrição das propriedades iniciais de codificação. Inclui informações como: tipo de mídia, *codecs* utilizados etc.	Sim.	1
plugin-name	*/startup-config/**	Especifica o nome do *plugin* utilizado. Necessário para efetuar verificação se *plugin* carregado é o correto.	Não.	1, ocorrendo nos nós: • *negotiation*; • *transport*; • *stream*.
library-name	*/startup-config/**	Define o nome da biblioteca implementadora do *plugin* de adaptação. Exemplo: **libspeex.so** indicando um *plugin* de adaptação de fluxo. Esta propriedade pode ser sobrescrita durante a inicialização de uma sessão de maneira dinâmica	Não.	1, ocorrendo nos nós: • *negotiation*; • *transport*; • *stream*.
library-directory	*/startup-config/**	Propriedade semelhante a anterior, mas que especifica o diretório de busca do *plugin*. Assim como a propriedade anterior, pode ser sobrescrita durante a inicialização de uma sessão de maneira dinâmica	Não.	1, ocorrendo nos nós: • *negotiation*; • *transport*; • *stream*.
provides	*/startup-config/**	Possui um conjunto finito de nós-filhos “provide”, utilizado para descrever todos os recursos disponibilizados por cada *plugin*.	Sim.	0,1

Continuação na próxima página

Tabela A.1 – continuação da página anterior

Nome do Nó	Nó Base	Descrição	Possui nós filhos?	Granularidade
provide	*/startup-config/*/provides*	Cada elemento deste tipo especifica um tipo de informação que será obtida a partir do *plugin*. Exemplo de propriedades poderiam ser RTT, número de perdas etc. Além do nome da informação provida, será especificado um valor padrão, utilizado caso haja problemas na recuperação, e o tempo padrão de atualização desta informação.	Não.	1...N
requires	*/startup-config/**	Possui um conjunto finito de nós-filhos "require", utilizado para descrever todos os recursos necessários a cada *plugin*.	Sim.	0,1
require	*/startup-config/*/requires*	Cada elemento deste tipo especifica um tipo de informação que deve ser informada ao *plugin* a cada intervalo de tempo especificado. Exemplo de propriedades poderiam ser RTT, número de perdas etc. Além do nome da informação provida, será especificado um valor padrão, utilizado caso haja problemas na recuperação, e o tempo padrão de atualização desta informação. Um cenário onde este recurso pode ser configurado, seria um *plugin* de adaptação que necessita receber continuamente a informação de largura de banda disponível.	Não.	1...N
enable-adaptation	*/startup-config/tranport* ou */startup-config/stream*	Especifica se o processo de adaptação estará ativo.	Não.	1
protocol	*/startup-config/protocol*	Indica qual protocolo da camada de transporte será utilizado. Informações como número de portas, e endereços de origem e destino não serão descritos na política mas sim passados pela aplicação.	Não.	1
transmission-properties	*/startup-config/stream*	Indica como deve ser o processo de transmissão em nível de aplicação. A partir desta estrutura será definido se a transmissão será apenas de áudio, ou de áudio/vídeo.	Sim.	1
adapt-config	/	Entrada utilizada para a descrição das políticas de adaptação. Este nó será composto pelas descrições de adaptação tanto a nível de transporte como de aplicação.	Sim.	0,1.

Continuação na próxima página

Tabela A.1 – continuação da página anterior

Nome do Nó	Nó Base	Descrição	Possui nós filhos?	Granularidade
policy	*/adapt-config/transport* ou */adapt-config/stream*	Define uma política de adaptação a ser efetuada, tendo como base um parâmetro de avaliação (Ex.: número de perdas, largura de banda atual etc.).	Sim	1...N

A.2 Exemplo de política de adaptação

Na Listagem A.1 é apresentado um exemplo de uma política de adaptação que pode ser validada e reconhecida pelo módulo de descrição de políticas de adaptação desenvolvido.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <adaptation-policy xmlns="AdaptationPolicy"
3  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4  xsi:schemaLocation="AdaptationPolicy file:/home/ivocalado/workspace/
       streamadapt/policies/policy.xsd">
5     <policy-name xmlns="">politica Teste</policy-name>
6     <policy-description xmlns="">O objetivo desta politica é realizar
          testes com o plugin de transporte</policy-description>
7     <startup-config xmlns="">
8         <negotiation>
9             <plugin-name>xmpp</plugin-name>
10            <library-name>xmpp.so</library-name>
11        </negotiation>
12        <transport>
13            <protocol>DCCP</protocol>
14            <plugin-name>DCCPSocketPlugin</plugin-name>
15            <library-name>libplugins-transport.so</library-name>
16            <library-directory>/home/ivocalado/workspace/plugins-
                  transport/Debug/</library-directory>
17            <enable-adaptation>true</enable-adaptation>
18             <provides>
19                 <provide default-value="2.6" update-time="1000">RTT</
                       provide>
20                 <provide default-value="30" update-time="500">bandwidt</
```

```
                    provide>
21              </provides>
22              <simpleproperty key="SDESItem">rtpsend demo app.</
                    simpleproperty>
23              <simpleproperty key="ExpireTimeout">1000000</simpleproperty>
24              <simpleproperty key="PayloadFormat">0</simpleproperty>
25              <simpleproperty key="SchedulingTimeout">10000</simpleproperty
                    >
26          </transport>
27          <stream>
28              <plugin-name>SpeexPlugin</plugin-name>
29              <library-name>libplugins-stream.so</library-name>
30              <library-directory>/home/ivocalado/workspace/plugins-stream/
                    Debug/</library-directory>
31              <transmission-properties transmission-type="full-duplex">
32                  <audio-transmission enable-preprocessing="true">
33                      <codec-name>WideBand</codec-name>
34                  </audio-transmission>
35              </transmission-properties>
36              <enable-adaptation>false</enable-adaptation>
37              <requires>
38                  <require default-value="100">bandwidth</require>
39              </requires>
40          </stream>
41      </startup-config>
42      <adapt-config xmlns="">
43          <transport>
44              <adaptation-method>qpolicy</adaptation-method>
45              <policy>
46                  <comparisson-attibute>RTT</comparisson-attibute>
47                  <lower threshold="2.5">
48                      <if>
49                          <equals Lhs="bandwidt" Rhs="30.0"/>
50                          <operations>
51                              <simpleproperty key="DCCPTXCCID">5</
                                    simpleproperty>
52                          </operations>
```

```
53                          </if>
54                          <simpleproperty key="PayloadFormat">1</simpleproperty
                                >
55                      </lower>
56                  </policy>
57              </transport>
58          </adapt-config>
59  </adaptation-policy>
```

Listagem A.1: Exemplo de descrição de política de adaptação.

Apêndice B

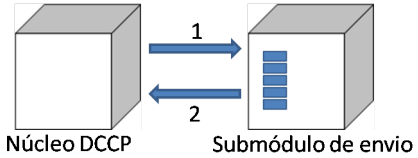
Desenvolvimento de políticas de descarte de pacotes para o *Framework QPolicy*

O *QPolicy* é um *framework* definido no núcleo da implementação do protocolo DCCP no kernel do Sistema Operacional Linux[1], que oferece uma estrutura componentizável para adição de mecanismos voltados para a realização de descartes de pacotes no protocolo DCCP. A partir da utilização deste *framework* torna-se possível a definição de diferentes modelos de adaptação multimídia em nível de camada de transporte.

A abordagem realizada pelo *QPolicy* para descartes de pacotes consiste em gerenciar a fila de pacotes a serem enviados pelo DCCP. Sendo assim, é de atribuição do *QPolicy* efetuar a adição dos pacotes recebidos para envio na posição correta da fila de envio, e por outro lado, retornar ao núcleo do DCCP o próximo pacote que deve ser enviado. Nas Figuras B.1(a) e B.1(b) são ilustrados os processos do escolha dos pacote de envio, tanto no cenário padrão (sem o *QPolicy*) quanto no cenário onde o *QPolicy* é presente.

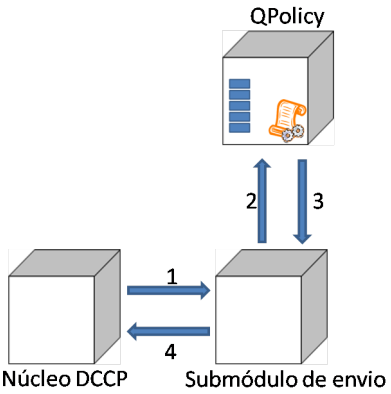
Cada política de descarte tem a responsabilidade de definir como um pacote recebido deverá ser tratado, utilizando para tal, informações de controle recebidas juntamente com os pacotes enviados. Exemplos de políticas incluem a utilização de níveis de prioridade, onde descarta-se pacotes de baixa prioridade em cenários em situações onde a fila de envio atinge um limiar pré-determinado; ou a temporização de pacotes, na qual estipula-se um tempo de

[1] O *QPolicy* encontra-se atualmente disponível em uma árvore de testes do kernel do Linux. Para mais detalhes, ver `http://eden-feed.erg.abdn.ac.uk` e `http://linux-foundation.org/en/Net:DCCP_Testing`.



1. O núcleo DCCP solicita o próximo pacote a ser enviado;
2. O Submódulo de envio recupera o primeiro elemento da lista de envio e retorno ao núcleo do DCCP

(a) Abordagem padrão para escolha do pacote a ser enviado.



1. O núcleo DCCP solicita o próximo pacote;
2. O Submódulo de envio requisita ao Qpolicy o pacote a ser enviado;
3. O QPolicy aciona a política ativa para definição do pacote que está no "topo" da fila de envio e descarta os desnecessários;
4. Retorna o pacote de envio.

(b) Atuação do *QPolicy*: define qual pacote deve ser enviado, com base numa política de descartes.

Figura B.1: Processo de decisão do próximo pacote a ser enviado no DCCP

expiração para cada pacote e caso este não seja enviado até a expiração, ele é descartado.

A criação de novas políticas de adaptação é definida a partir da implementação de três ações que efetuarão o controle da adição, remoção e controle do tamanho da fila de envio, conforme apresentadas na Listagem A.1.

```
1 void qpolicy_push(struct sock *sk, struct sk_buff *skb)
2 {
3         //Ação de adição de novos pacotes a fila de envio
4 }
5
6 bool qpolicy_full(struct sock *sk)
7 {
8         //Verificação da ocupação máxima da file de envio
9 }
10
```

```
11 struct sk_buff *qpolicy_top(struct sock *sk)
12 {
13         //Retirada do próximo pacote a ser enviado
14 }
```

Listagem A.1: exemplo de política de descarte de pacotes no *QPolicy*.

Apesar do *QPolicy* representar uma interessante abordagem para possibilitar a utilização de diferentes políticas de descarte de pacotes em uma transmissão, ele não oferece suporte ao processo de alteração de política de descarte durante uma transmissão. Neste sentido, contribuiu-se com o *QPolicy*, adicionando suporte a diâmica da política de descarte durante a transmissão. A partir da implementação deste recurso, é possível obter uma maior flexibilidades em como o processo de adaptação no contexto do protocolo DCCP pode ser realizado.

Apêndice C

Implementação do algoritmo *Cubic* no protocolo DCCP

Neste apêndice é apresentada uma breve descrição a respeito da implementação do algoritmo de controle de congestionamento *Cubic* como um novo CCID no protocolo DCCP. Para tal, serão discutidos a motivação por trás deste desenvolvimento, como o novo algoritmo foi desenvolvido e como habilitá-lo; e finalmente qual o estágio atual de desenvolvimento deste CCID.

C.1 Motivação para o desenvolvimento

Conforme discutido na Seção 2.2.3, o algoritmo de controle de congestionamento *TCP Cubic* é caracterizado por ser menos agressivo que o TCP BIC, mantendo, contudo, a estabilidade e escalabilidade. O sucesso gerado a partir da utilização do *TCP Cubic* fez com que ele passasse a ser o algoritmo de controle de congestionamento padrão do kernel do Linux a partir da versão 2.6.18, vindo a substituir o algoritmo BIC.

No trabalho apresentado em [14], é realizado uma avaliação experimental dos algoritmos de controle de congestionamento do DCCP em transmissões compartilhadas, envolvendo tanto fluxos UDP quanto TCP. Neste trabalho, constatou-se que o algoritmo *Cubic* apresenta um desempenho bastante satisfatório, mostrando-se ao mesmo tempo ser amigável em relação aos demais fluxos, seja UDP, DCCP ou TCP e agressivo em termos de utilização da largura de banda disponível. Além disso, observou-se que em relação aos algoritmos de con-

trole de congestionamento, atualmente padronizados para o DCCP, isto é, CCID-2 e CCID-3, o algoritmo *Cubic* apresentou um desempenho superior.

As conclusões obtidas pelo trabalho supracitado serviu como base para motivar o desenvolvimento de um novo CCID com base no algoritmo *Cubic*, de forma a aumentar o conjunto algoritmos de controle de congestionamento que podem ser utilizados para realização de uma transmissão DCCP.

C.2 Passo a Passo da implementação

O desenvolvimento do CCID-5 utilizou como base a implementação do TCP *Cubic* sempre que possível. Esta opção foi adotada, visando tornar o processo de desenvolvimento mais dinâmico e também diminuir as chances de surgimento de *bugs* na implementação, visto que a versão do *Cubic* para TCP se encontra estável e bastante otimizada.

O processo de desenvolvimento, pode ser dividido basicamente em três etapas, a serem discutidas a seguir:

Criação da estrutura básica para adição de um novo CCID - O passo inicial para a implementação do CCID-5, foi a adição de uma estrutura básica de um CCID. Tal estrutura, contém os requisitos mínimos para quem um novo CCID possa ser compilado com sucesso no kernel, incluindo a configuração de diferentes arquivos, de forma a registrar o novo CCID e a definição de um arquivo, chamado ccid5.c, contendo a atribuição do identificador do novo mecanismo de controle de congestionamento, conforme apresentado na Listagem B.1. De maneira geral, esta etapa foi a mesma realizada para a distribuição de um CCID sem controle de congestionamento (CCID-0)

```
1 struct ccid_operations ccid5_ops = {
2         .ccid_id        = DCCPC_CCID5,
3         .ccid_name      = "Cubic-DCCP",
4 };
5
6 MODULE_AUTHOR("Ivo Calado");
7 MODULE_DESCRIPTION("The Cubic DCCP (based on TCP Cubic)");
8 MODULE_LICENSE("GPL");
9 MODULE_ALIAS("net-dccp-ccid-5");
```

Listagem B.1: estrutura mínima da implementação de um CCID.

Cópia dos módulos internos ao algoritmo *Cubic* - Como pode ser observado na implementação do algoritmo TCP *Cubic*, alguns módulos existentes são responsável pela realização de alguns cálculos internos do algoritmo cúbico, e portanto são independentes do protocolo de transporte. A implementação do TCP *Cubic* foi planejada de tal forma que tal divisão pudesse ser respeitada. Como consequência, foi possível uma rápida detecção de tais módulos de forma que pudesse ser rapidamente integrado a implementação do CCID-5.

Adaptação dos módulos dependentes de protocolo de transporte - Esta etapa apresentou a maior complexidade para execução, dentre as supracitadas. Basicamente, fez-se necessário a realização de um mapeamento entre as propriedades do *socket* TCP, utilizadas pelo algoritmo *Cubic*, para propriedades análogas no DCCP.

Uma segunda tarefa realizada, foi a implementação de um mecanismo de detecção de eventos de perda de pacotes. O motivo da realização desta tarefa deve-se ao fato de que, diferentemente do TCP, o DCCP não implementa nenhum mecanismo de detecção de perda no núcleo do protocolo. Tal tarefa é delegada pelo DCCP aos mecanismos de controle de congestionamento, visto que diferentes CCIDs apresentam abordagem diferentes, no que confere a detecção de perda de pacotes. Como versão inicial da implementação, utilizou-se como abordagem para detecção de perda o reconhecimento de conjunto de pacotes de reconhecimento repetidos. Estudos posteriores estão sendo realizados, visando o aprimoramento desta técnica, como a utilização de reconhecimento seletivo, por exemplo.

C.3 Habilitando o CCID-5 no kernel do Linux

Para que o CCID-5 seja habilitado durante a compilação, a opção *"CCID-5 (Cubic DCCP) (EXPERIMENTAL) (NEW)"* deve estar habilitada no menu de configuração para compilação do kernel, conforme apresentada na Figura C.1[1]

[1] A aparência do menu pode divergir ligeiramente, de acordo com o contínuo desenvolvimento do kernel do Linux.

Networking support —>

Networking options —>

<M> The DCCP Protocol (EXPERIMENTAL) —>

DCCP CCIDs Configuration (EXPERIMENTAL) —>

[*] CCID-5 (Cubic DCCP) (EXPERIMENTAL) (NEW)

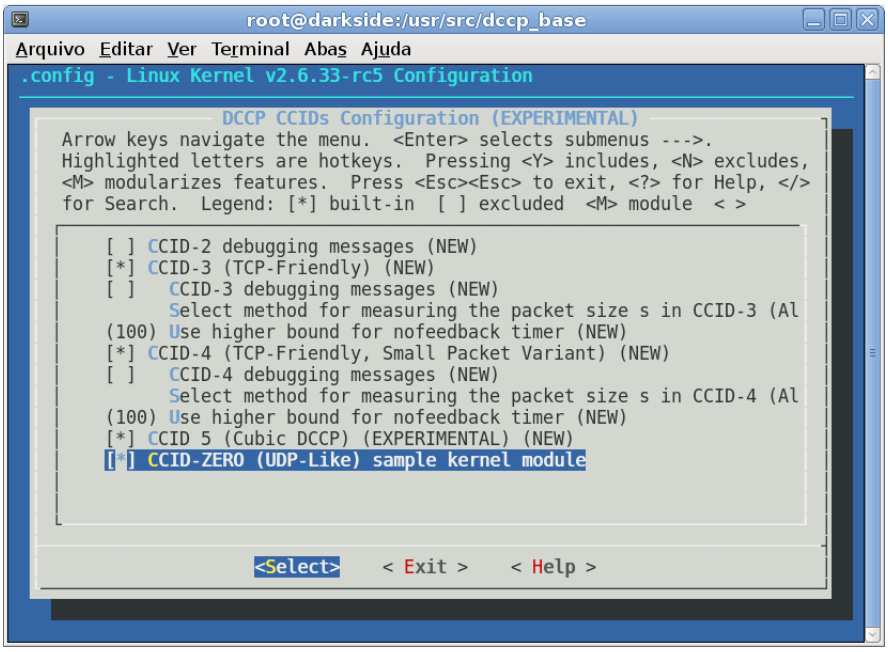


Figura C.1: Habilitando o CCID-5 no kernel do Linux

C.4 Estágio atual de desenvolvimento

Atualmente a implementação do algoritmo *Cubic* encontra-se em estágio experimental de desenvolvimento, onde estão sendo realizados testes para eliminação *bugs* e verificações de desempenho.

Para obtenção da última versão da implementação do CCID-5, foi disponibilizada uma árvore experimental modificada do kernel do Linux, podendo ser acessada via Git [2] em `http://embedded.ufcg.edu.br/~ephone/`. Além disso, estão sendo realizados contatos com os mantenedores do protocolo DCCP no kernel do Linux, em especial Gerrit Renker, de forma a tornar a implementação desenvolvida padrão no kernel do Linux.

[2] Sistema de controle de versão distribuído utilizado no gerenciamento de diferentes projetos, inclusive do kernel do Linux. Para maiores informações, acessar `http://git-scm.com/`.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo