PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Análise de Desempenho do Protocolo SCTP Utilizando DiffServ

Constantino Augusto Dias Neto

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http: ftp.cin.ufpe.br
ftp://ftp.cin.ufpe.br/pub/posgrad

RECIFE, DEZEMBRO/2002

UNIVERSIDADE FEDERAL DE PERNAMBUCO-UFPE
CENTRO DE INFORMÁTICA - CIN
MESTRADO EM CIÊNCIAS DA COMPUTAÇÃO

Análise de Desempenho do Protocolo SCTP Utilizando DiffServ

Constantino Augusto Dias Neto

Dissertação apresentada como parte dos requisitos para obtenção do grau de Mestre em Ciências da Computação.

Área de Concentração: Redes de Computadores

Orientador (a): Profº. Judith Kelner

RECIFE, DEZEMBRO/2002

Dias Neto, Constantino Augusto
Análise de desempenho do protocolo SCTP utilizando DiffServ / Constantino Augusto Dias Neto. – Recife : O Autor, 2002.
xv, 107 folhas: il., fig., tab., graf.

Dissertação (mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, 2002.

Inclui bibliografia e anexos.

1. Redes de computadores – Sistemas distribuídos. 2. Protocolo de transporte (SCTP) – Análise de desempenho. 3. Computação – Redes – Qualidade de serviço (DiffServ). I. Título.

004.7 CDU (2.ed.) UFPE
004.62 CDD (21.ed.) BC2003-030

Agradecimentos

Agradeço a Deus, por me dar forças nos momentos de dificuldades e permitir concluir essa dissertação com muito esforço e dedicação.

Agradeço os meus pais, que sempre me deram apoio e incentivo através de uma educação de boa qualidade, mostrando que os obstáculos que aparecem no caminho é para serem superados para que possamos alcançar nossos objetivos e sonhos.

Agradeço aos meus irmãos Alisson, Valeska e Katyuscia que sempre me incentivaram nesse novo desafio na minha vida.

Agradeço a minha namorada Nayana pela compreensão, carinho e incentivo.

Agradeço a minha orientadora Prof. Dra. Judith Kelner que sempre esteve pronta a me ajudar.

Agradeço ao Prof. Dr. Djamel Sadok por ser meu co-orientador.

Agradeço em especial a Prof. Dra. Marcília Andrade Campos pela atenção e ajuda.

Agradeço aos meus amigos Kamienski e Dênio por esclarecerem minhas dúvidas.

Agradeço a todos os amigos de Recife, por terem me acolhido muito bem e ajudado no possível.

Agradeço a ajuda de todos os funcionários da UFPE que diretamente e indiretamente apoiaram na realização deste trabalho e nessa conquista.

Índice

Capítulo 1.0 Introdução 1

1.1 Motivação 2
1.2 Caracterização do Problema 3
1.3 Trabalhos Relacionados 4
1.4 Estrutura da Dissertação 4

Capítulo 2.0 Qualidade de Serviço na Internet 6

2.1 Qualidade de Serviço – (*Quality of Service*) 7
2.2 Parâmetros de QoS 8
2.3 Arquiteturas para prover QoS na Internet 9
2.3.1 Serviços Integrados – (IntServ) 9
2.3.1.1 Serviço Garantido 10
2.3.1.2 Serviço de Carga Controlada 10
2.3.1.3 Componentes do IntServ 11
2.3.1.3.1 Controle de Admissão 13
2.3.1.3.2 Classificador 13
2.3.1.3.3 Escalonador de Pacotes 13
2.3.2 O Protocolo de Reserva de Recursos – (RSVP) 13
2.3.2.1 Problemas com a arquitetura IntServ/RSVP 18
2.3.3 Arquitetura DiffServ 19
2.3.3.1 Definição do Campo DS 20
2.3.3.2 SLA – *Service Level Agreements* 20
2.3.3.3 Condicionamento de Tráfego 21
2.3.3.4 PHB 23
2.3.3.4.1 PHB EF 23
2.3.3.4.2 PHB AF 25
2.3.5 Engenharia de Tráfego (TE) 26
2.3.6 MPLS 28
2.3.6.1 Componentes do MPLS 29
2.3.6.1.1 Rótulo 29
2.3.6.1.2 LSR 30
2.3.6.1.3 LER 30
2.3.6.1.4 LSP 31
2.3.6.1.5 FEC 31
2.3.6.2 Vantagens do MPLS 32
2.3.6.3 Aplicações do MPLS 32
2.3.7 Roteamento Baseado em QoS (QoSR) 33
2.3.8 Roteamento Baseado em Restrições 33
2.3.9 Considerações Finais 34

Capítulo 3.0 Os Protocolos TCP e UDP 35

3.1 Histórico do Protocolo TCP 36
3.2 Definição do Protocolo TCP 36
3.3 Modelo de Serviço do TCP 37
3.4 O Cabeçalho do Segmento TCP 37
3.5 Gerenciamento de Conexão 40
3.5.1 Estabelecimento de Conexão 40

3.5.2 Mantendo uma Conexão no TCP 41
3.5.3 Encerrando uma Conexão 41
3.6 Controle de Congestionamento do TCP 42
3.7 Algoritmos para Controle de Congestionamento do TCP 42
3.7.1 Algoritmo *Slow Start e Congestion Avoidance* 43
3.7.2 Algoritmo *Fast Retransmit* 45
3.7.3 Algoritmo *Fast Recovery* 46
3.8 O Protocolo UDP 47
3.8.1 O Cabeçalho do Segmento UDP 48
3.9 Considerações Finais 48

Capítulo 4.0 O Protocolo SCTP 50

4.1 Introdução 51
4.2 Arquitetura do SCTP 51
4.3 Formato do Segmento SCTP 52
4.3.1 O Cabeçalho do Segmento SCTP 53
4.3.2 Campos do *Chunk* 53
4.4 Descrição do *Chunks* SCTP 53
4.4.1 Dados (*DATA*) (*Type* = 0) 53
4.4.2 Início (*INIT*) (*Type* = 1) 55
4.4.3 Reconhecimento do Início (*INIT ACK*) (*Type* = 2) 56
4.4.4 Reconhecimento Seletivo (*SACK*) (*Type* = 3) 57
4.4.5 Pedido do *Heartbeat* (*HEARTBEAT*) (*Type* = 4) 59
4.4.6 Reconhecimento do *Heartbeat* (*HEARTBEAT ACK*) (*Type* = 5) 59
4.4.7 Abortar uma Associação (*ABORT*) (*Type* = 6) 60
4.4.8 Início do *Shutdown* (*SHUTDOWN*) (*Type* = 7) 61
4.4.9 Reconhecimento do *Shutdown* (*SHUTDOWN ACK*) (*Type* = 8) 61
4.4.10 Operação de Erro (*ERROR*) (*Type* = 9) 61
4.4.11 *Cookie Echo* (*COOKIE ECHO*) (*Type* = 10) 62
4.4.12 Reconhecimento do *Cookie* (*COOKIE ACK*) (*Type* = 11) 63
4.4.13 Finalização do *Shutdown* (*SHUTDOWN COMPLETE*) (*Type* = 14) 63
4.5 Gerenciamento de uma associação 64
4.5.1 Iniciação de uma associação 64
4.5.2 Transmissão de Dados no SCTP 66
4.5.3 Término da Associação no SCTP 67
4.6 Características do Protocolo SCTP 69
4.6.1 SCTP *Multi-Homing* 69
4.6.2 *Multi-Streaming* na Associação 70
4.6.3 Serviço com Rígida Ordem de Entrega das Mensagens 71
4.6.4 Serviço de Entrega de Mensagens Fora de Ordem 71
4.6.5 Fragmentação e Remontagem 71
4.6.6 Empacotamento 72
4.8 Controle de Congestionamento no protocolo SCTP 72
4.8.1 Diferenças do Controle de Congestionamento do SCTP em relação ao TCP 73
4.9 Algoritmos *Slow-Start* and *Congestion Avoidance* no SCTP 74
4.9.1 *Slow-Start* 74
4.9.2 *Congestion Avoidance* 75
4.9.3 Algoritmo *Fast Retransmit* no Protocolo SCTP 75
4.10 Gerenciamento de Falhas no Protocolo SCTP 76
4.10.1 Monitoramento de Falha no *Endpoint* 76

4.10.2 Monitoramento de Falha no *Path* 76
4.11 Segurança no protocolo SCTP 77
4.12 Considerações Finais 77

Capítulo 5.0 Análise de Desempenho do SCTP 79

5.1 Técnicas de Avaliação de Desempenho 80
5.2 O simulador de redes *Network Simulator* (NS) 80
5.3 Ambiente de Simulação 81
5.4 Métricas 81
5.5 Topologia da Simulação 82
5.6 Parâmetros e Fatores da Simulação 82
5.7 Estudo de Caso 84
5.8 Resultados da Simulação 85
5.9 Considerações Finais 86

Capítulo 6.0 Conclusão 87

6.1 Considerações Finais 88
6.2 Contribuições 89
6.3 Trabalhos Futuros 89
6.4 Referências Bibliográficas 90

Anexos 95

Lista de Acrônimos

ATM	*Asynchronous Transfer Mode*. Arquitetura que transmite dados em forma de pacotes, utilizando célula de 53 *Bytes*.
BA	*Behavior Aggregate*. Corresponde a uma coleção de pacotes com o mesmo DSCP
BE	*Best Effort*. Modelo de serviço utilizado na internet
CBR	*Constraint-based routing*. Roteamento baseado em restrições
CBR	*Constant Bit Rate*. Utilizado para simular tráfego constante e voz
CoS	*Class of Service*. Refere-se as classes de serviço implementadas em QoS
CWND	*Congestion Window*. A janela de congestionamento é o limite que o transmissor tem para transferir os dados quando inicia a transmissão ou depois de receber uma confirmação
DARPA	*Department Advanced Research Project Agency*. Nome formal da ARPA
DNS	*Domain Name System*. Protocolo da Internet que converte nome de domínios em seus endereços IP correspondentes
DOD	*Department of Defense*. Departamento de Defesa Americano
DS	*Differentiated Service*. Corresponde ao domínio que implementa Serviços Diferenciados
DSCP	*Differentiated Service Code Point*. Corresponde ao código utilizado pelo Serviço Diferenciado
FEC	*Forwarding Equivalence Classes*. Classe de equivalência de encaminhamento de pacotes presente na arquitetura MPLS
FF	*Fixed Filter*. Estilo de reserva fixa utilizada pelo protocolo RSVP
FTP	*File Transfer Protocol*. Protocolo da camada de aplicação que permite a transferência de arquivos
GPRS	*General Packet Radio Service***.** Solução de dados sem fio com eficiência de espectro que oferece às operadoras um primeiro passo para serviços 3G com vantagens em termos de custo
HTTP	*Hyper Text Transfer Protocol*. Linguagem de diagramação interpretada pelos *browsers* na Internet
IAB	*Internet Architecture Board*. Grupo para definição de políticas e tendências para TCP/IP e a Internet

ICMP	*Internet Control Message Protocol*. Protocolo de controle da camada IP que trata erros e mensagens da internet
IETF	*Internet Engineering Task Force*. Grupo associado a IAB para pesquisas em TCP/IP e Internet
IGMP	*Internet Group Management Protocol*. Protocolo de gerenciamento usado por *hosts* para definição de grupos *multicast*
IP	*Internet Protocol*. Protocolo de roteamento, não orientado a conexão, utilizado na Internet
IPV4	*Internet Protocol version* 4. Protocolo IP da camada de rede versão 4
IPV6	*Internet Protocol version* 6. Protocolo IP versão 6
ISI	*Information Sciences Institute*. Instituto de Ciências da Informação da Universidade do Sul da Califórnia.
ISP	*Internet Service Provider*. Corresponde ao provedor de serviço de Internet.
IW	*Initial Window*. Variável na qual a janela de congestionamento do TCP é inicializada
LER	*Label Edge Routers*. São os roteadores de núcleos de bordas do domínio MPLS.
LSP	*Label Switch Path*. Corresponde aos caminhos percorridos pelos pacotes no domínio MPLS.
LSR	*Label Switch Routers*. São os roteadores de comutação de rótulos presentes no domínio MPLS.
MAC	*Message Authentication Code*. Mensagem de código de autenticação.
MAXTH	*Maximum Threshold*. Limite máximo da fila no mecanismo RED
MF	*Multi Field*. Tipo de classificador
MINTH	*Minimum Threshold*. Limite mínimo da fila no mecanismo RED
MPLS	*Multiprotocol Label Switching*. Comutação por rótulos que integra a funcionalidade e flexibilidade da camada de rede permitindo maior agilidade no encaminhamento de pacotes
MSS	*Maximum Segment Size*. Tamanho máximo do segmento suportado pelo receptor de dados
NS	*Network Simulator*. Simulador de redes
OSPF	*Open Shortest Path First Protocol*. Protocolo de roteamento da Internet que roteia pacotes através do menor caminho possível

OTCL	*Object Tool Control Language*. Linguagem não qual são escritos os scripts a serem simulados pelo NS
PHB	*Per Hop Behavior*. Tratamento específico de encaminhamento de pacotes, que corresponde à alocação de recursos
PHB-AF	*Per Hop Behavior Assured Forwarding*. Tipo de PHB, onde os pacotes são marcados para cada uma das classes com três níveis de precedência
PHB-EF	*Per Hop Behavior Expedited Forwarding*. PHB utilizado para prover um serviço fim a fim, com baixos níveis de perdas de pacotes, de atraso e de *jitter*
PHOP	*Previous Hop*. Campo nas mensagens PATH que determina o proximo ponto
PRR	*Priority Round Robin*. Escalonador de filas
PSTN	*Public Switched Telephonic Network*. Rede Pública de Telefonia Comutada.
QoS	*Quality of Service*. Qualidade de Serviço
QoSR	*QoS Routing*. Roteamento baseado em QoS , utilizado na engenharia de tráfego
RED	*Random Explicit Dinamic*. Mecanismo que permite o gerenciamento ativo de filas
RFC	*Request for Comments*. Nome dado a uma série de notas, que podem ser idéias, técnicas e observações, assim como a especificação de um protocolo proposto e aceito. Os documentos RFC são disponíveis *on-line*
RIP	*Routing Information Protocol*. Protocolo de roteamento da Internet
RSVP	*Resource Reservation Protocol*. Protocolo de reserva de recursos
RTO	*Retransmit Time Out*. Tempo de retransmissão do pacote
RTT	*Round Trip Time*. Corresponde ao tempo que um pacote necessita para ir e voltar da fonte de dados ao destino
RWND	*Receiver Window*. Essa janela informa o limite que o receptor tem para receber os dados, ela é transmitida no cabeçalho dos segmentos de reconhecimento e/ou dados
SCTP	*Stream Control Transmission Protocol*. Protocolo da camada de transporte
SE	*Shared Explicit*. Estilo de reserva compartilhada no protocolo RSVP
SFQ	*Stochastic Fair Queuing*. O algoritmo divide o fluxo das diversas sessões em filas, usando um algoritmo *round robin*. Isto impede que uma sessão tome conta, unitariamente, da fila de transmissão.

SLA	*Service Level Agreement*. Contrato de serviço entre o provedor e o cliente para fornecimento de Serviços Diferenciados
SMSS	*Sender Maximum Segment Size*. Tamanho máximo do segmento do transmissor
SNMP	*Simple Network Management Protocol*. Protocolo utilizado no gerenciamento de redes, para monitorar e configurar os dispositivos da rede
SSN	*Stream Sequence Number*. Número de seqüência do *stream*
SSTHRESH	*Slow Start Threshold*. Corresponde ao limite do crescimento da janela de congestionamento
TCA	*Traffic Conditioning Agreement*. Refere-se ao acordo de condicionamento de tráfego, junto com as regras de classificação adotadas pelo domínio DS
TCB	*Transmission Control Block*. Estrutura de dados interna criada pelo SCTP *endpoint* que contém todas as informações operacionais para o *endpoint*
TCL	*Tool Control Language*. É uma linguagem script gratuita e roda em várias plataformas
TCP	*Transmission Control Protocol*. Protocolo da camada de transporte
TOS	*Type of Service*. Campo do IPV4 que deu origem ao campo DS
TSN	*Transmission Sequence Number*. Número de seqüência de transmissão dos dados no protocolo SCTP.
UDP	*User Datagram Protocol*. Protocolo de transporte não orientado a conexão
VBR	*Variable Bit Rate*.Utilizado para tráfego com taxa de dados variável
VoIP	*Voice over IP*. Aplicação de voz sobre o protocolo IP
VPN	*Virtual Private Network*. Rede privada virtual.
WAN	*Wide Area Network*. Rede de grande alcance
WF	*Wildcard Filter*. Estilo de reserva coringa no protocolo RSVP
WFQ	*Weighted Fair Queuing*. É uma disciplina de filas que atende às aplicações que precisam de tempo de resposta consistente.Além de minimizar o atraso, favorece a vazão, pois se não houver tráfego nas filas de alta prioridade, a vazão a elas destinada é usada pelas filas de baixa prioridade.
WRR	*Weighted Round Robin*. Escalonador de filas

Índice de Figuras

Figura 1: Equipamentos envolvidos na Qualidade de Serviço 8
Figura 2: Modelo de Referência para Roteadores 12
Figura 3: Camada de atuação do protocolo RSVP 14
Figura 4: Reserva de recursos através do protocolo RSVP 17
Figura 5: Domínio dos Serviços Diferenciados 19
Figura 6: Estrutura do Campo DS 20
Figura 7: Etapas de um condicionador de tráfego 22
Figura 8: Encaminhamento PHB EF 24
Figura 9: Implementação do encaminhamento assegurado (PHB AF) 26
Figura 10: Encaminhamento de pacotes sem engenharia de tráfego 27
Figura 11: Encaminhamento de pacotes com engenharia de tráfego 28
Figura 12: Roteamento e a comutação associadas as rotas de tráfego 29
Figura 13: Cabeçalho MPLS 30
Figura 14: Ambiente MPLS 31
Figura 15: Layout do segmento TCP 38
Figura 16: Estabelecimento de uma conexão no TCP 40
Figura 17: Encerramento de uma conexão TCP 42
Figura 18: Funcionamento do *Fast Retransmit* 46
Figura 19: Cabeçalho do UDP 48
Figura 20:Uma associação SCTP 52
Figura 21: Formato do segmento SCTP com vários *chunks* 52
Figura 22: Formato do *DATA Chunk* 54
Figura 23: Formato do *INIT chunk* 55
Figura 24: Formato do *INIT ACK Chunk* 56
Figura 25: Formato do *SACK chunk* 58
Figura 26: Formato do *HEARTBEAT REQUEST chunk* 59
Figura 27: Formato do parâmetro *Heartbeat Info* 59
Figura 28: Formato do *HEARTBEAT ACK* 60
Figura 29 : Formato do *ABORT chunk* 60
Figura 30: Formato do *Shutdown chunk* 61
Figura 31: Formato do *SHUTDOWN ACK chunk* 61
Figura 32: Formato do *ERROR chunk* 62
Figura 33: Formato do *COOKIE ECHO chunk* 63
Figura 34: Formato do *Cookie ACK* 63
Figura 35: Formato do *SHUTDOWN COMPLETE chunk* 63
Figura 36:*For-way-Handshake* na iniciação de uma associação 64
Figura 37:Transmissão de dados no SCTP 66
Figura 38: Término de uma associação no SCTP 68
Figura 39 : *Endpoint Multi-Homing* 69
Figura 40 : Topologia adotada na simulação 82

Índice de Gráficos

Gráfico 1 : Funcionamento do Slow Start e Congestion Avoidance 45

Índice de Tabelas

Tabela 1: Atraso de propagação numa rede WAN 8
Tabela 2: Vazão típica de algumas aplicações 9
Tabela 3 : Estilo de reserva suportado e escopo 16
Tabela 4: Proposta de Codificação AF 25
Tabela 5: Serviços do TCP com o número da porta 37
Tabela 6: Causas de erro presentes no *ERROR chunk* 62
Tabela 7: Fontes de Tráfego utilizada na simulação para uma rede sem QoS 83
Tabela 8: Fontes de Tráfego utilizada em uma rede com QoS 84
Tabela 9: Indicadores Estatísticos 85

Resumo

A Internet de hoje usa o modelo do melhor-esforço (*Best-Effort* - BE), em que não há nenhum suporte para QoS (*Quality of Service*). Os pacotes são encaminhados sem quaisquer requisitos de qualidade de serviço, ou seja, quando há congestionamento os pacotes são descartados aleatoriamente.

Em virtude das novas aplicações que surgiram para integrar o uso de dados, voz e vídeo na Internet, foi necessário, então, a utilização de QoS para prover um serviço diferenciado em relação ao modelo do melhor esforço, mesmo que seja necessário pagar um custo mais alto. Como conseqüência deste fato ocorreu o desenvolvimento de serviços adicionais a arquitetura TCP/IP (*Transfer Control Protocol / Internet Protocol*) para o provimento de QoS, dentre esses se destaca o DiffServ (*Differentiated Services*) que agrega os diversos fluxos em classes de serviços.

Este trabalho aborda as questões referentes ao uso do protocolo SCTP no transporte de dados, explicita os conceitos de Qualidade de Serviço e os mecanismos necessários para sua implementação. Apresenta também as vantagens do protocolo SCTP como *Multi-Streaming*, *Multi-Homing* que o diferencia em relação aos protocolos atuais da Internet (TCP e UDP). Analiza-se o comportamento dos pacotes em relação ao atraso e *jitter* em redes que utilizam o protocolo SCTP e TCP.

Palavras chaves: SCTP, TCP, DiffServ, Qualidade de Serviço

Abstract

Nowadays, the Internet uses the model of the best-effort (BE), in which there is no support for QoS (Quality of Service). Packets are referred without any requirements of quality of service. In others words, when it has congestion, the packets are discarded randomly.

Due to the news applications that arise to integrate the use of data, voice, and video in the Internet, it's necessary the utilization of QoS to cater the differentiated service beside the best-effort, even though you have to pay more. As consequence of this fact happened the development of extra services, one of them is the TCP/ IP (Transfer Control Protocol/ Internet Protocol) to offer QoS, between these terms the DiffServ (Differentiated Service) that collect the diverse fluxes in classes of service.

This work accosts questions about to the use of SCTP protocol in data transportation , show the concepts of Quality of Service and the necessary mechanisms to this installation. Also introduce the vantages of SCTP protocols as Multi-Streaming, Multi-Homing that differentiate in relation to present protocols from Internet (TCP and UDP). Analyzes the behavior of the packages in relation to the delay and to jitter in nets that use protocol SCTP and TCP.

Keywords: SCTP, TCP, DiffServ,Qualidade de Serviço

Capítulo 1.0
Introdução

Este capítulo apresenta o modelo da Internet atual, a motivação para esse trabalho, seus objetivos e o escopo. Apresenta também a estrutura geral desta dissertação.

1.1 Motivação

A Internet atual oferece um único tipo de serviço, chamado melhor-esforço, onde todo pacote recebe o mesmo tipo de tratamento. Isto significa que não é oferecido ao usuário nenhuma garantia de perda de pacotes, largura de banda, atraso fim a fim e variação do atraso para as suas aplicações.

O grande crescimento da Internet nos últimos anos se deu devido principalmente à possibilidade de trabalhar com outras mídias, além do texto, à utilização também de imagens e de áudio, por exemplo. Existe atualmente uma necessidade crescente de aplicações como vídeo sob demanda, transmissão de TV ao vivo, vídeo-conferência, tele-medicina, entre outras. Com o surgimento dessas aplicações multimídias e com aumento da capacidade de processamento dos computadores pessoais, a Internet tende a receber uma maior quantidade desse tipo de tráfego. O tipo de tráfego da Internet BE (*Best-Effort*) não atende a nova demanda por Qualidade de Serviço (QoS), e não oferece garantias de desempenho a determinados usuários e aplicações.

As aplicações multimídias caracterizam-se pela dependência em relação ao tempo de entrega dos pacotes de dados aos destinatários. Os atrasos sofridos pelos pacotes, ao longo do caminho desde a origem, devem ser de alguma maneira compensados pela aplicação no destino, de forma que a reprodução da informação transmitida seja realizada com a menor distorção possível. Porém, para que esta compensação seja possível, a infra-estrutura de comunicação deve prover meios de garantir que estes referidos atrasos mantenham-se dentro de limites restritos, especificados pelas aplicações. Com isso, as aplicações multimídias demandam controle de QoS e uma crescente capacidade de tráfego. Isto implica em consumo excessivo de largura de banda, recurso considerado escasso que deve ser compartilhado por todas essa aplicações.

Com a introdução dos parâmetros de QoS na rede e o uso das novas aplicações multimídias, com requisitos diferentes das aplicações tradicionais, houve uma tarifação diferente para este tipo de serviço diferenciado, em relação ao utilizado pela Internet atual. Aliado a esta tarifação e com a utilização dos parâmetros de QoS pode-se reservar largura de banda para diferentes tipos de fluxos, onde os pacotes não serão descartados e a largura de banda não excederá os valores predefinidos.

Pode-se observar um crescimento na convergência das redes de voz e dados para o oferecimento de serviços de voz e multimídia, e para que os serviços de voz baseados em pacotes tenham aceitação no mercado, é essencial que a qualidade do serviço oferecido ao usuário final seja pelo menos tão boa quanto a oferecida pelas atuais redes de comutação de circuitos. O protocolo TCP tem apresentado limitações para estes serviços, por não suportar *hosts Multi-Homing* (várias interfaces IP). Uma solução para este problema foi o surgimento do protocolo SCTP (*Stream Control Transmission Protocol*) [Ste00] comportando-se de uma forma mais eficiente no transporte de mensagens de sinalização SS7 (Sistema de Sinalização 7) sobre redes IP.

1.2 Caracterização do Problema

As redes TCP/IP (*Transmission Control Protocol/Internet Protocol*), foram projetadas para o envio de informações usando o modelo de melhor esforço sem quaisquer requisitos de qualidade de serviço. Essas redes têm uma imensa base instalada com milhões de computadores que continuam crescendo em praticamente todo o mundo. O forte aumento e a aceitabilidade das redes IP, ocorreu em função do crescimento explosivo da Internet por volta dos anos 90, e que ainda continua evoluindo no novo milênio. Este é um contraste em relação aos anos 70 e 80, onde TCP/IP era um assunto, praticamente, restrito as comunidades de pesquisa. Hoje, o protocolo TCP é responsável por mais de 90 % do tráfego da Internet [Tho97].

A definição original do protocolo TCP [Pos81c], o qual foi concebido e incubado principalmente dentro de ambiente de pesquisa, tem começado a mostrar certas limitações diante das novas necessidades comerciais. Principalmente com o surgimento de um grande número de aplicações multimídias e serviços tais como a Telefonia sobre o IP, algumas limitações do TCP são:

- O fluxo orientado do TCP é freqüentemente um inconveniente, porque as aplicações devem adicionar um registro marcado para delimitar as suas mensagens, e deve fazer o uso explícito do comando "*push*" para assegurar que a mensagem completa seja transferida dentro de um tempo razoável; e
- O TCP fornece transferência confiável e ordem rígida de transmissão de dados. Algumas aplicações precisam de confiabilidade na transferência sem a necessidade de manter a seqüência;

- Uma conexão TCP é diretamente identificada por um par de endereços de transporte (endereço IP e número da porta). Isto impede o suporte de *hosts Multi-Homing* (várias interfaces IP).

Diante dessas limitações, o bloqueio do cabeçalho oferecido pelo TCP causa um atraso desnecessário, com isso, o protocolo SCTP [Ste00] apresenta melhor confiabilidade e segurança.

Com o surgimento das novas aplicações comerciais e outras classes de serviços com diferentes prioridades e certas garantias de disponibilidade da rede, observou-se a necessidade que houvesse garantia e qualidade no serviço, por isso é necessário o uso de QoS. Dentre os vários mecanismos de implementação de QoS utilizou-se o DiffServ (Serviços Diferenciados) por controlar os fluxos de tráfego em uma ou mais redes atendendo as necessidades das aplicações.

1.3 Trabalhos Relacionados

Alguns trabalhos considerados relevantes para está dissertação são:

Em [Jun00] ocorre uma descrição das características do protocolo SCTP. Contém uma comparação do protocolo TCP e SCTP em um cenário que utiliza uma conexão de satélite com *Backup*. O [Mag01] analisa um ambiente de Serviços Diferenciados com tráfego de vídeo MPEG-4, mostrando a necessidade do uso de QoS (Qualidade de Serviço).[Cam02b] realiza uma análise estatística de descarte de pacotes em redes TCP.

1.4 Estrutura da Dissertação

Os capítulos desta dissertação estão organizados da seguinte forma:

Capítulo 1 – Apresenta a motivação do trabalho, caracterização do problema, além dos trabalhos relacionados com o desenvolvimento do mesmo e a estrutura da dissertação.

Capítulo 2 - Apresenta qualidade de serviço na internet, com a descrição detalhada das arquiteturas para o provimento de QoS;

Capítulo 3 - Será discutido o protocolo TCP e UDP (*User Datagram Protocol*), mostrando as principais características desses protocolos de transporte;

Capítulo 4 - Será especificado o funcionamento do protocolo SCTP;

Capítulo 5 - Avalia, através de simulação com a utilização do NS (*Network Simulator*), o desempenho do protocolo SCTP e será feita uma comparação com o protocolo TCP;

Capítulo 6 - Serão apresentadas as conclusões desta dissertação, contribuições, algumas sugestões para futuros trabalhos e são relacionadas às referências bibliográficas utilizadas na elaboração deste trabalho;

Capítulo 2.0
Qualidade de Serviço na Internet

Este capítulo descreve algumas propostas para implementação de Qualidade de Serviço na Internet, e apresenta as arquiteturas utilizadas para o provimento de QoS.

2.1 Qualidade de Serviço – (*Quality of Service*)

Encontrar uma definição padrão para QoS (*Quality of Service*) que contemple todos os serviços é bastante difícil. Cada serviço tem sua própria definição para QoS e pode ser descrito por suas características. Para exemplificar, têm-se algumas definições:

- Qualidade de Serviço é um requisito da aplicação para a qual exige-se que determinados parâmetros (atraso, vazão, etc.) estejam dentro de limites bem definidos (valor mínimo, valor máximo) [Job99];
- Qualidade de Serviço é utilizado para definir o desempenho de uma rede relativa às necessidades das aplicações, como também o conjunto de tecnologias que possibilita às redes oferecer garantias de desempenho [Tei98].

Há duas expressões que são freqüentemente utilizadas na literatura [Cro99, Fer98a]: (CoS) Classe de Serviço Diferenciados e (QoS) Qualidade de Serviço. Embora os dois termos sejam usados como sinônimos há diferença entre eles. CoS significa que serviços podem ser divididos em classes, onde existe um tratamento diferenciado de encaminhamento dos pacotes. No CoS o principal conceito é a diferenciação. QoS, por sua vez, é utilizado num sentido mais específico, para designar serviços que oferecem garantias restritas com relação a determinados parâmetros (como largura de banda e atraso) a seus usuários.

A Qualidade de Serviço é garantida pelos equipamentos da rede e camadas de protocolo de forma cooperada. A Figura 1 [Mar99] ilustra uma situação onde na trajetória fim-a-fim dos pacotes tem-se equipamentos tipo *LAN Switch*, roteadores, *Firewalls*. Utiliza-se uma rede pública de comutação de pacotes e, obviamente, tem-se os próprios *hosts* dos usuários finais.

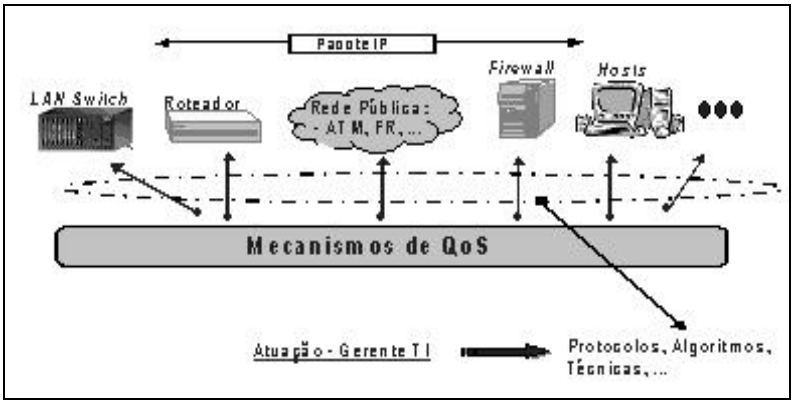

Figura 1: Equipamentos envolvidos na Qualidade de Serviço

2.2 Parâmetros de QoS

Para utilizar QoS na transmissão de dados é importante o planejamento dos seguintes parâmetros [Fer98b]:

Atraso – (*Latency*) – É o tempo necessário para que um pacote possa ser encaminhado na rede desde o transmissor até o receptor. Os principais fatores que influenciam no atraso de uma rede são [Mar99]:

1. Atraso de propagação;
2. Velocidade de transmissão;
3. Tempo de processamento nos equipamentos.

A Tabela 1 [Mar99] ilustra o atraso de propagação entre cidades numa rede WAN (*Wide Area Network*) que utiliza fibras ópticas como meio físico de comunicação.

Trecho (*Round Trip Delay*)	**Atraso de Propagação**
Miami a São Paulo	100 mseg
New York a Los Angeles	50 mseg
Los Angeles a Hong Kong	170 mseg

Tabela 1: Atraso de propagação numa rede WAN

Variação de Atraso – (*Jitter*) - É a variação do atraso verificada durante uma transmissão fim-a-fim. Mesmo com o retardo dentro dos limites aceitáveis, variações acentuadas no retardo podem ter efeitos negativos na qualidade de serviço oferecida à algumas aplicações (como um vídeo sob demanda).

Largura de Banda – (*Bandwidth*) - Indica a capacidade máxima de transmissão de dados em uma conexão entre um transmissor e receptor. Para transferir mais dados o usuário deve aumentar a largura de banda. Em termos práticos as aplicações geram vazões que devem ser atendidas pela rede. A Tabela 2 [Mar99] ilustra a vazão típica de algumas aplicações:

Aplicação	Vazão (Típica)
Aplicações Transacionais	1 Kbps a 50 Kbps
Quadro Branco (*Whiteboard*)	10 Kbps a 100 Kbps
Voz	10 Kbps a 120 Kbps
Aplicações Web (WWW)	10 Kbps a 500 Kbps
Transferência de Arquivos (Grandes)	10 Kbps a 1 Mbps
Aplicação de vídeo-conferência	500 Kbps a 1 Mbps
Vídeo MPEG	1 Mbps a 10 Mbps
Aplicação com Imagens Médicas	10 Mbps a 100 Mbps
Aplicação para Realidade Virtual	80 Mbps a 150 Mbps

Tabela 2: Vazão típica de algumas aplicações

Confiabilidade – (*Confiability*) – É a taxa de ocorrência de erros no meio físico, que compreende a duplicação de pacotes, descarte e reordenação de pacotes na rede.

Na Internet, entretanto, para prover QoS é necessário à combinação desses parâmetros com as várias propostas da IETF (*Internet Engineering Task Force*) que serão vistas na próxima seção.

2.3 Arquiteturas para prover QoS na Internet

Atualmente, no âmbito da IETF, estão sendo discutidas várias abordagens para o oferecimento de QoS na Internet [Xia99]. Dentre as quais se destacam: a arquitetura de Serviços Integrados (IntServ) que utiliza o protocolo RSVP (*Resource Reservation Protocol*) [Bra94] para reserva de recursos, a arquitetura de serviços diferenciados (Diffserv) [Bla98], MPLS (*Multiprotocol Label Switching*) [Ros99], Engenharia de Tráfego (*Traffic Engineering*) [Awd99] e Roteamento Baseado em QoS (*QoS Routing*) [Cra98].

2.3.1 Serviços Integrados – (IntServ)

Os Serviços Integrados [Bra94] surgiram devido a Internet atual oferecer um único modelo de serviço, ou seja, o de melhor esforço, onde não há garantia rígidas para: largura de banda, atraso e perda de pacotes. Com isso a IETF criou o grupo de trabalho de serviços integrados (IntServ) com a finalidade de viabilizar o surgimento de uma rede de Serviços

Integrados, em que fosse possível ter aplicações de tempo real na Internet satisfatórias do ponto de vista do usuário final.

O termo Serviços Integrados é usado para definir um modelo para a Internet que inclui os seguintes serviços: o de melhor esforço, o de compartilhamento controlado do link e o de tempo real.

A arquitetura IntServ [Bla98] adiciona duas classes de serviços para a existência do serviço de melhor esforço, elas são: serviço garantido [She97] e o serviço de carga controlada [Wro97].

2.3.1.1 Serviço Garantido

É uma classe de QoS [She97] proporcionada pelo modelo de serviços integrados que oferece um nível assegurado de largura de banda, um limite de atraso fim a fim na fila e uma proteção contra perda de pacotes nas filas, para os pacotes que estiverem obedecendo ao perfil de tráfego contratado. Esse serviço não oferece garantia mínima da variação do atraso, ele simplesmente garante um atraso máximo gerado pelas filas.

A obtenção de um limite máximo para o atraso exige que todos os roteadores no caminho suportem o serviço garantido. O comportamento fim a fim oferecido por uma série de roteadores, que implementam o serviço garantido é um nível assegurado de largura de banda para um determinado fluxo que, quando utilizado por um fluxo que está sendo policiado, produz um serviço com atraso limitado para todos os pacotes que estejam dentro do perfil. Esse serviço é voltado para aplicações com requisitos rígido de tempo real, como certas aplicações multimídias, que precisam garantir (garantia rígida) que o pacote não irá chegar no destino após um tempo maior que o contratado.

2.3.1.2 Serviço de Carga Controlada

O comportamento fim a fim oferecido para uma aplicação por uma série de roteadores que implementa esse serviço se assemelha ao comportamento visto por aplicações que estão recebendo o serviço de melhor esforço em uma rede apenas levemente carregada, ou seja, sem nenhuma situação grave de congestionamento [Wro97]. As garantias que as aplicações têm são:

- Um percentual grande dos pacotes transmitidos chega com sucesso no receptor (deve se aproximar da taxa básica de erros do meio de transmissão, ou seja, pouquíssimos descartes em filas são permitidos); e
- O atraso sofrido por um alto percentual dos pacotes não deverá exceder em muito, o atraso mínimo sofrido por um pacote dentro de um fluxo. A maior parte dos pacotes deve ter um atraso muito próximo do atraso mínimo.

Para assegurar que essas condições sejam válidas, aplicações que requisitam o serviço de carga controlada devem fornecer aos roteadores uma estimativa de tráfego de dados que elas irão gerar, chamada de TSpec (especificador de tráfego), que é baseada em balde de fichas[1] (*Token Bucket*). Como resposta, o serviço assegura que a aplicação terá a sua disposição recursos suficientes dos roteadores, para processar adequadamente todos os pacotes que estiverem de acordo com a especificação contida no TSpec. Por outro lado, os pacotes inseridos na rede fora dessas especificações, poderão ser descartados, ou enfrentar um atraso mais significativo.

O objetivo do serviço de carga controlada é suportar uma ampla classe de aplicações que tem sido desenvolvida para a Internet atual, mas que não funcionam em situações de carga alta (congestionamento) na rede.

As aplicações podem solicitar o serviço de carga controlada antes de iniciar as transmissões, ou então, somente quando elas detectam que o serviço de melhor esforço não está oferecendo um desempenho aceitável. A primeira estratégia oferece uma maior garantia, ou seja, que o nível de QoS não irá mudar enquanto durar a sessão. A segunda estratégia é mais flexível e barata, pois o serviço de tarifação mais cara não será utilizado durante todo o tempo de duração da sessão.

2.3.1.3 Componentes do IntServ

Nos Serviços Integrados o roteador deve implementar QoS para cada fluxo. A Figura 2 [Pag00] ilustra o modelo de referência para roteadores que implementam o IntServ.

[1]É um mecanismo de implementação de QoS que contém uma determinada quantidade máxima de fichas que são inseridas regularmente no balde. As fichas correspondem a permissão para transmitir uma quantidade de *bits*. Quando chega um pacote, o seu tamanho é comparado com a quantidade de fichas no balde. Se houver quantidade de fichas disponíveis o pacote é enviado, senão, é inserido numa fila e aguarda haver fichas disponíveis no balde.

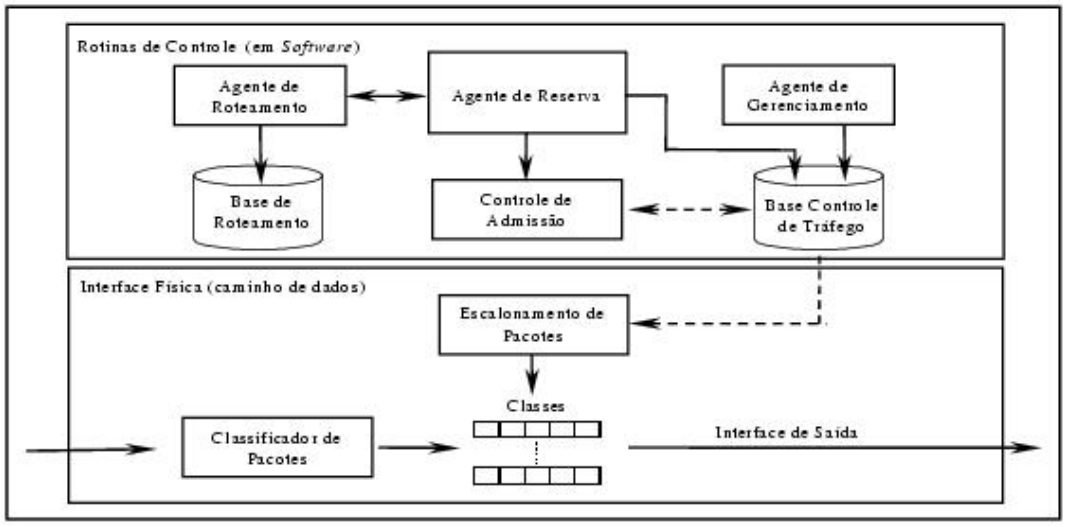

Figura 2 : Modelo de Referência para Roteadores

O esquema no modelo do roteador tem duas divisões funcionais:

- O caminho de dados (interface física);
- As rotinas de controle (em *software*).

A primeira recebe e envia os dados de acordo com a política definida localmente em cada roteador e a última, executada em segundo plano, modifica as bases de dados do modelo e controla o caminho de dados. De acordo com as classes de serviços, um roteador classifica localmente os pacotes dividindo-os em filas onde podem ser reordenados e reenviados de maneira que a QoS seja mantida para os fluxos majoritários.

As rotinas de controle são formadas por três agentes:

1. Roteamento – Faz a interface com o protocolo de roteamento para obtenção de endereços;
2. Reserva – Implementa o protocolo de configuração de reservas;
3. Gerenciamento – De forma local, implementa a política de compartilhamento do canal e modifica os parâmetros da política de controle de admissão.

Os Serviços Integrados são implementados por quatro componentes [Cla94]: o controle de admissão (*admission contro*l), o classificador (*classifier*), o escalonador de pacotes (*packet scheduler)* e o RSVP (*Resource Reservation Protocol*) que será visto na próxima seção em detalhes, os demais serão apresentados a seguir.

2.3.1.3.1 Controle de Admissão

O controle de admissão [She97, Sug99] implementa um algoritmo, que o roteador ou *host* usa para determinar se um novo fluxo pode ter seu pedido de QoS atendido, sem interferir nas garantias concedidas anteriormente. Como cada fluxo precisa certamente de uma quantidade de recursos (largura de banda e espaço em *buffer* no roteador), então o controle de admissão é usado para controlar a alocação de recursos da rede. O algoritmo do controle de admissão deve ser consistente com o modelo de serviço, e é logicamente parte do controle de tráfego [Bra94].

As principais considerações levadas para aceitação do pedido são: carga de tráfego atual, QoS contratada, perfil de tráfego solicitado entre outras. O controle de admissão tem um papel importante na contabilidade e no relatório administrativo [Jus00].

2.3.1.3.2 Classificador

O classificador é responsável pela classificação dos pacotes que chegam nas várias classes existentes, onde pacotes pertencentes a uma mesma classe recebem o mesmo tratamento. Uma classe é uma abstração que pode ser local para um roteador, ou seja, o mesmo pacote pode ser classificado de forma diferente por roteadores diferentes ao longo do trajeto [Bra94]. A classe geralmente corresponde a um fluxo específico.

2.3.1.3.3 Escalonador de Pacotes

O escalonador de pacotes é responsável pelo encaminhamento dos diferentes fluxos de pacotes, utilizando políticas de fila e temporizadores. O escalonador deve ser implementado, no local, onde os pacotes são enfileirados e deve haver uma comunicação com a interface da camada de enlace de dados para controlar a alocação da largura de banda entre os fluxos.

2.3.2 O Protocolo de Reserva de Recursos – (RSVP)

O protocolo RSVP (*Resource Reservation Protocol*) foi originalmente desenvolvido pelos pesquisadores da Universidade do Sul da Califórnia e pela *Xerox Palo Alto Reaserch Center*. A IETF está no momento trabalhando na padronização do RSVP. O RSVP é um protocolo de sinalização para as aplicações reservar recursos da rede. Ele é utilizado por

sistemas finais para requisitar a rede níveis específicos de QoS para as aplicações. Para sua utilização, dois pré-requisitos devem ser adotados:

1. Elementos de redes, tais como roteadores, devem adequar-se aos mecanismos de controle de qualidade de serviço para garantir a entrega dos pacotes de dados;
2. A aplicação deve estar apta a fornecer os parâmetros ideais de QoS.

O protocolo RSVP foi feito de forma a garantir que as arquiteturas mais antigas sejam compatíveis com o novo sistema, através do encapsulamento de seus pacotes de controle. O protocolo atua tanto em máquinas do usuário quanto em roteadores, responsabilizando-se, nesse caso, a estabelecer e manter as condições para o serviço requisitado. As principais características do RSVP apresentadas de forma resumida são:

1. O RSVP não realiza transporte de dados, ou seja, não é um protocolo de roteamento, sendo apenas um protocolo de controle e atua no mesmo nível dos outros protocolos como o ICMP (*Internet Control Message Protocol*), e o IGMP (*Internet Group Management Protocol*). A Figura 3 ilustra a camada de atuação do protocolo RSVP [Sch00].

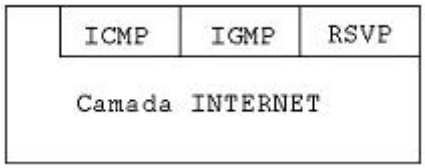

Figura 3: Camada de atuação do protocolo RSVP

2. O protocolo RSVP opera em modo *simplex*, isto é, as reservas de recursos são estabelecidas apenas em uma direção, possibilitando a transmissão de dados em apenas um sentido. Essas reservas podem ser tanto para aplicações *unicast* como para *multicast*, ou seja, tanto para aplicações que são configuradas para um único receptor, quanto para as que têm a capacidade de transmitir para mais de um receptor sem precisar enviar um *broadcast* para a rede inteira.
3. O protocolo RSVP utiliza o conceito de sessão [Bra97] como sendo um fluxo de dados que possui um identificador composto pelo endereço de destino, a identificação do protocolo de transporte e, opcionalmente, a porta de destino na

camada de transporte. O campo de identificação de sessão permite a escolha na recepção dos fluxos de uma sessão. Uma sessão RSVP pode possuir diversos fluxos caracterizados como uma fonte enviando dados a um conjunto de receptores.

4. O RSVP introduz a noção de *soft-state* (atualizações periódicas no estado de controle que são: estado de caminho e estado de reserva) inicialmente proposto por [Cla88], que define o estado que um determinado elemento pertencente ao percurso de dados de um determinado par fonte-destino. Os estados de reserva no RSVP têm um tempo máximo de validade. O receptor automaticamente atualiza os estados de reserva, ou seja, o RSVP delega aos receptores o trabalho de reenviar periodicamente suas requisições de serviços. Isso permite que ele se adapte, automaticamente, as alterações no roteamento. Caso uma falha ocorra, somente uma nova requisição do serviço restabelecerá o *soft-state* nos roteadores.
5. Os roteadores que não implementam RSVP podem estar presentes no caminho, que suas mensagens serão encaminhadas de forma transparente.
6. As reservas de recursos são iniciadas e mantidas pelos receptores.
7. O RSVP suporta vários estilos de reserva (ver Tabela 3), para se adaptar a uma grande variedade de aplicações e uso.
8. O RSVP suporta tanto o IPV4 (*Internet Protocol Version* 4) como o IPV6 (*Internet Protocol Version* 6).

O protocolo RSVP envolve uma série de opções em uma reserva de recursos, que agrupados, definem um estilo de reserva. O estilo de reserva pode ser: **distinto**, a existência de cada fluxo RSVP reserva exatamente para um único transmissor; ou **compartilhado** com múltiplos transmissores usando o mesmo fluxo RSVP.

Um estilo de reserva pode ser também por **seleção explícita de transmissor,** onde recursos são dedicados somente para transmissores explícitos dentro da mensagem de reserva, ou por **seleção de transmissor coringa,** onde o tráfego de algum transmissor é selecionado. A Tabela 3 ilustra cada combinação de estilo de reserva suportado e o seu escopo.

Escopo	Reserva	
	Distinto	**Compartilhado**
Explícito	Estilo Filtro Fixo (FF)	Estilo Explícito-Compartilhado (SE)
Coringa	Não definido	Estilo Filtro Coringa (WF)

Tabela 3 : Estilo de reserva suportado e escopo

Estilo de Filtro Fixo (FF – *Fixed Filter*): As reservas **FF** são requisitadas por diferentes receptores, que selecionam o mesmo transmissor, em todo caso, precisam se juntar para compartilhar uma simples reserva em um dado nó. Esse tipo de reserva é utilizado por aplicações de vídeo, onde cada fluxo de vídeo pode ser dedicado para um particular fluxo RSVP.

Estilo de Filtro Coringa (WF – *Wildcard Filter*): O estilo de reserva **WF**, é projetado para sessões na qual todas as fontes requerem garantias de serviços similares e as fontes são capazes de limitar suas saídas, como numa conferência de áudio. No estilo **WF**, uma reserva é feita e compartilhada para todas as fontes. Esse estilo deve ser particularmente usado em caso de sessões *multicast*.

Estilo de Compartilhamento Explícito (SE – *Shared Explicit*): O **SE** especifica um ambiente de reserva compartilhada com um escopo explícito de reserva. O estilo **SE** cria uma reserva simples no qual todos os fluxos dos servidores superiores são misturados. Como no caso de uma reserva **FF**, na qual um conjunto de servidores é especificado implicitamente pelo receptor que está fazendo a reserva. Assim como o estilo **WF**, o **SE** permite aplicações apropriadas para *multicast*.

O protocolo RSVP suporta quatro tipos de mensagens básicas:

- **Mensagem de requisição de reserva** – A mensagem *RESV* enviada por cada receptor ao transmissor, percorre o caminho indicado pela mensagem *PATH* e solicita a reserva dos recursos necessários.
- **Mensagem de caminho** – A mensagem *PATH* enviada pelo transmissor ao receptor, percorre todos os roteadores indicados como o melhor caminho pelo protocolo de roteamento. Esta mensagem contém informações sobre as características do caminho, a fim de subsidiar o receptor na solicitação de recursos.

- **Mensagem de erro e confirmação** – Existem três tipos diferentes de mensagens.

  path-error – Quando ocorre um erro no registro de caminho de uma mensagem *PATH*, esta mensagem é retornada ao transmissor;

  reservation-request error – Informa ao receptor falha de admissão, banda indisponível, serviço não suportado ou caminho ambíguo, em retorno a uma mensagem de requisição de reserva;

  reservation-request acknowledgment – Enviada ao receptor como confirmação de recebimento da mensagem de requisição de reserva;

- **Mensagens de teardown** – Essa mensagem desfaz o caminho de reserva de recursos sem esperar por um *timeout*, ou seja, apaga uma reserva que existia.

A Figura 4 [Xia99] ilustra a reserva de recursos através do protocolo RSVP.

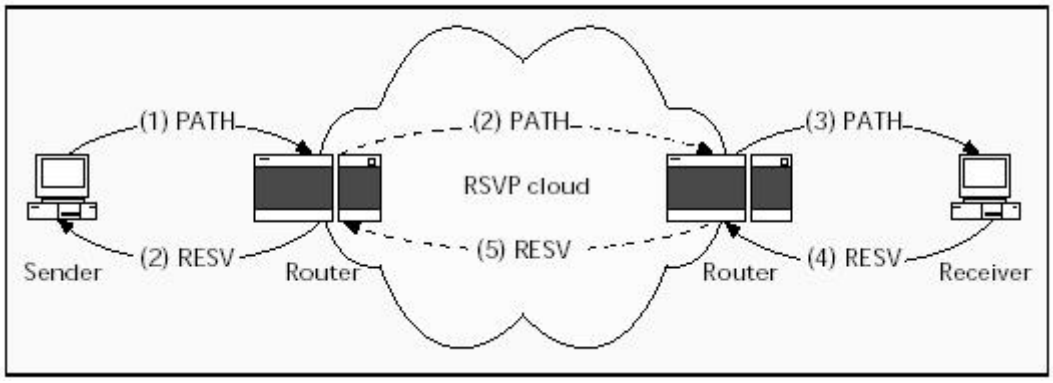

Figura 4: Reserva de recursos através do protocolo RSVP

O RSVP opera na troca de dois tipos de mensagens entre transmissor(es) e receptor(es): *PATH* (mensagem de caminho) e *RESV* (mensagem de reserva). O transmissor envia uma mensagem PATH para o receptor especificando o tráfego gerado pela fonte e os níveis de QoS que a rede pode oferecer aos receptores, além de estabelecer o caminho. Esse caminho é construído através do campo PHOP (*previous-hop*) nas mensagens *PATH* [Whi97]. Este campo contém o endereço da máquina adjacente que envia a mensagem *PATH*.

Todo roteador intermediário junto ao caminho transfere a mensagem *PATH* para o próximo *hop* que é determinado pelo protocolo de roteamento. Além disso, a mensagem

PATH instala informações de roteamento reverso em todos os nós por onde passa, para que a mensagem *RESV* possa percorrer o mesmo caminho. A mensagem *PATH* não faz a reserva, ela é feita pela mensagem *RESV*, enviada pelo receptor. Essa mensagem informa a caracterização do tráfego e da QoS requisitada pelos receptores, e define estilos de reserva que especificam quais fontes devem ser selecionadas para que os respectivos tráfegos recebam um tratamento com QoS.

As informações de roteamento reverso são necessárias porque é comum que a comunicação nos dois sentidos siga caminhos distintos. Sem essas informações, as reservas de recursos poderiam ser feitas em roteadores diferentes por onde os dados irão passar. Quando o receptor recebe uma mensagem *PATH,* o receptor responde com uma mensagem *RESV* para requisitar recursos para o fluxo. Todo roteador intermediário do caminho pode rejeitar ou aceitar a requisição da mensagem *RESV*. Se a requisição é rejeitada, o roteador enviará uma mensagem de erro para o receptor, e o processo de sinalização será terminado. Se a requisição é aceita, a largura de banda e o espaço em *buffer* são alocados para o fluxo, e as informações de estado relacionadas ao fluxo são instaladas no roteador.

2.3.2.1 Problemas com a arquitetura IntServ/RSVP

Apesar de oferecer garantia de recursos para todos os fluxos a arquitetura IntServ e sua componente RSVP apresentam os seguintes problemas:

- Todos os roteadores são obrigados a implementar RSVP, controle de admissão, classificação e escalonamento de pacotes, com isso as exigências nos roteadores são altas.
- A quantidade de informações de estado aumenta proporcionalmente com o número de fluxos que um roteador tem de tratar. Esta situação gera uma grande sobrecarga aos roteadores, em termos de capacidade de processamento e armazenamento, considerando que roteadores de núcleo da Internet tratam simultaneamente milhares de fluxos. Por esta razão, a arquitetura não é escalável para o núcleo da Internet.
- Para cada fluxo deve haver sinalização em cada nó (sistema final ou roteador). A troca de mensagens de sinalização é grande, inclusive porque o RSVP trabalha com estado leve, prejudicando a escalabilidade.

2.3.3 Arquitetura DiffServ

Para solucionar o problema da falta de escalabilidade encontrada no IntServ surgiu a arquitetura DiffServ [Bla98] para a implementação de QoS na rede. Apesar de ser altamente escalável, o Diffserv não oferece a garantia de recursos para todos os fluxos, como o IntServ. As reservas de recursos são feitas para agregações, ou seja, grandes conjuntos de fluxo chamado de BA (*Behavior Aggregate*).

Um BA é definido formalmente como uma coleção de pacotes com o mesmo DSCP (*Differentiated Service Code Point*) que está cruzando um enlace em uma determinada direção dentro de um domínio DS (*Differentiated Service*). As redes DiffServ são chamadas de domínio DS, que corresponde a um conjunto de nós DS trabalhando com uma mesma política de serviços. Os roteadores de borda (*Edge Routers*) são aqueles que se comunicam com roteadores de outros domínios, recebendo ainda a denominação de roteadores de ingresso ou de egresso, dependendo do sentido de tráfego contratado, com a função de rotear "fluxos agregados". Estes fluxos são formados através da marcação de um novo campo no pacote, chamado DS (*Differentiated Service*). Os roteadores de núcleo (*Core Routers*) somente se comunicam internamente, ou seja, com os roteadores do próprio domínio, encaminhando os fluxos de acordo com a "classe de serviço" estabelecida [Bla98]. A Figura 5 [Mag01] ilustra o domínio dos serviços diferenciados.

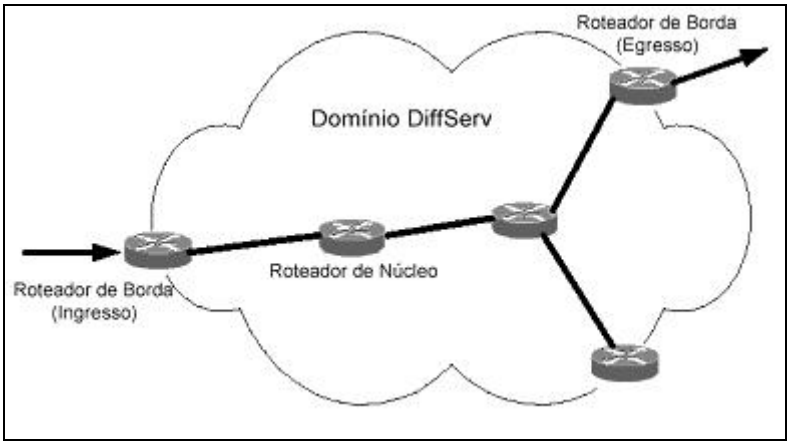

Figura 5: Domínio dos Serviços Diferenciados

O princípio básico de implementação de QoS na arquitetura de Serviços Diferenciados, é garantido através de mecanismos de priorização de pacotes, o que reduz o

nível de processamento nos roteadores para fluxos de dados. Isto é realizado com a definição de algumas Classes de Serviços (CoS) numa estrutura de rede. O objetivo da agregação dentro de um domínio DS é aumentar a escalabilidade pela manutenção de um menor número de estados.

2.3.3.1 Definição do Campo DS

O campo DS é obtido pela mudança de nome do campo TOS (*Type of Service*), no caso do IPV4 ou do campo *Traffic Class,* no caso do IPV6. A Figura 6 [Ziv99] ilustra a estrutura do campo DS, onde os seis primeiros *bits* do campo DS formam o sub-campo do DSCP que identifica o fluxo agregado, ou seja, seleciona o tipo de tratamento a ser oferecido pelos roteadores que suportam este modelo. Os dois outros *bits* estão reservados para ser utilizado em futuras funcionalidades.

Figura 6: Estrutura do Campo DS

2.3.3.2 SLA – *Service Level Agreements*

Um SLA são contratos de serviço que os domínios DS negociam entre si, visando o oferecimento de garantias mínimas de QoS para as aplicações dos usuários [Mcc98]. Para um cliente receber Serviços Diferenciados a partir de seu provedor de serviço Internet, ele deve possuir um SLA (*Service Level Agreement*) associado ao seu ISP (*Internet Service Provider*).

O SLA basicamente especifica as classes de serviços suportadas e a quantidade de tráfego permitida em cada classe. Um exemplo típico de SLA, para uma aplicação de voz sobre IP (VoIP - *Voice over IP*) com algumas centenas de canais de voz simultâneos numa rede IP WAN poderia ser ilustrado por:

- Vazão ≥ 2 *Mbps*;
- Atraso ≤ 250 mseg;

- Disponibilidade ≥ 99,5%.

Uma vez que a rede garanta este SLA, tem-se como resultado que a aplicação VoIP em questão poderá ser executada, garantindo a qualidade de voz prevista para os seus usuários em comunicação simultânea através da rede IP. Um SLA pode ser classificado como:

SLA Estático - são negociados de forma regular, por exemplo, mensalmente ou anualmente.

SLA Dinâmico - clientes com SLAs dinâmicos devem usar um protocolo de sinalização, por exemplo RSVP, para requisitar serviços sob demanda.

As regras de classificação dos pacotes, policiamento e condicionamento usadas nos roteadores de borda são derivadas a partir dos SLAs. A quantidade de espaço em *buffer* necessária para essas operações, é derivada a partir dos SLAs.

Todos os pacotes que seguem de um domínio para outro são fiscalizados (policiados) nos roteadores de borda para verificar sua conformidade com os contratos. Quando um pacote entra em um domínio a partir de outro domínio, seu campo DS pode ser marcado se o domínio anterior não possuir DiffServ, como especificado pelo SLA entre os dois domínios [Bla98]. Dentro do núcleo da rede, os roteadores ordenam os pacotes de entrada em diferentes classes de encaminhamento, de acordo com os valores do campo DS.

2.3.3.3 Condicionamento de Tráfego

A política de condicionamento de tráfego integra o acordo de condicionamento de tráfego TCA (*Traffic Conditioning Agreement*), junto com as regras de classificação adotadas pelo domínio DS.

O TCA se refere às atividades realizadas pelos roteadores de borda, que envolve o policiamento dos pacotes para averiguar a sua adequação ao perfil de tráfego contratado, prazos, cobranças e multas em caso de descumprimento das garantias oferecidas. A ação a ser adotada dependerá da política escolhida pelos administradores do domínio, levando-se em consideração os medidores, marcadores, descartadores, classificadores e suavizadores na aderência dos pacotes ao perfil de tráfego contratado. A Figura 7 [Bla98] ilustra as etapas de um condicionador de tráfego.

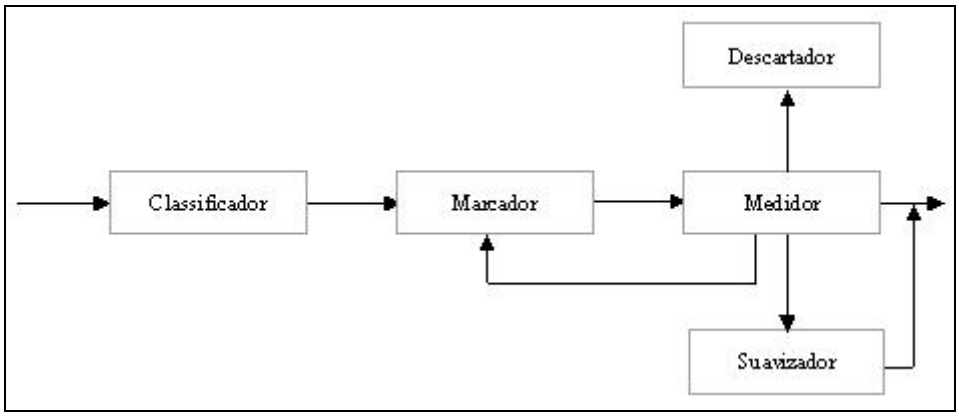

Figura 7: Etapas de um condicionador de tráfego

Classificador (*Classifier*) - classifica os pacotes recebidos com base no conteúdo de alguma parte do cabeçalho. Os dois tipos principais de classificadores são o BA (*Behavior Aggregate*) e o MF (*Multi-Field*). O BA classifica os pacotes somente no conteúdo do campo DSCP, ou seja, vem de um domínio DiffServ. Enquanto que o MF ocorre quando o domínio anterior não é habilitado com o campo DSCP previamente marcado, podendo o classificador avaliar vários campos dos pacotes, como por exemplo: endereço origem/destino, protocolos, portas.

Marcador (*Marker*) - dispositivo que marca o DSCP dos pacotes. Ele é também utilizado quando os dois domínios utilizam valores de DSCP diferentes para o mesmo BA. É de fundamental importância o seu uso quando os pacotes são classificados por um classificador MF, para que os roteadores de núcleo posteriores possam classificar os pacotes com um classificador BA.

Medidor (*Metter*) - responsável por verificar a conformidade do tráfego ao perfil de tráfego contratado. Um medidor passa informação de estado para outras funções de condicionamento que podem ser suavização, descarte ou marcação, para disparar uma ação particular a cada pacote, o pacote pode estar dentro ou fora do perfil dependendo do SLA. Embora vários mecanismos de medição possam ser utilizados, o balde de fichas é o mais apropriado.

Suavizador (*Shaper*) - introduz um atraso nos pacotes fora do perfil contratado, até que a balde tenha fichas suficientes para encaminhá-lo, ou seja, suaviza a passagem dos pacotes para manter o fluxo dentro do perfil contratado pelo usuário.

Descartador (*Dropper*) – descarta os pacotes considerados fora do perfil pelo medidor.

2.3.3.4 PHB

Pacotes diferentes podem ter tratamento distintos nos roteadores, para sua aderência aos requisitos de QoS. Esse tratamento específico de encaminhamento é chamado de PHB (*Per Hop Behavior*), indispensável ao fornecimento de um Serviço Diferenciado. Para identificar pacotes como pertencente a determinado PHB é utilizado o sub-campo DSCP (*Differentiated Service Code Point*) do campo DS.

A implementação de um BA requer que todos os pacotes recebam o mesmo tratamento de encaminhamento PHB (comportamento por enlace) nos roteadores onde passam. O PHB é a maneira como um roteador aloca recursos para os BA's, e é em cima desse mecanismo local que os serviços diferenciados são construídos. Por isso, os PHB's também são referenciados como classes de serviços. A maneira mais simples de implementar um PHB é destinar a ele um determinado percentual de utilização da largura de banda dentro de um enlace de saída.

Há várias propostas para tipos de PHB's na arquitetura de Serviços Diferenciados [Dob99]. Porém, há basicamente dois tipos sendo padronizados: Encaminhamento Expresso (EF) - (*Expedited Forwarding*) e Encaminhamento Assegurado (AF) - (*Assured Forwarding*). Além desses dois, há o PHB BE (*Best Effort*) para o comportamento de encaminhamento de tráfego de melhor esforço, pois Serviços Diferenciados devem ser compatíveis com as implementações já definidas e em funcionamento. De acordo com [Hei99] o sub-campo DSCP do campo DS possui o valor "000000" para o PHB BE.

2.3.3.4.1 PHB EF

Em um domínio DS podemos utilizar o PHB EF (*Per Hop Behavior Expedited Forwarding*) [Jac99] que define garantias mais rígidas de QoS para aplicações muito sensíveis as variações de características temporais da rede. Desta maneira, o PHB EF pode ser usado para prover um serviço fim-a-fim, com baixo níveis de perda de pacotes, atraso e baixo *jitter*, bem como largura de banda assegurada. Esse serviço aparece nos pontos finais como uma linha dedicada virtual, sendo conhecido como *Premium*. Os pacotes em conformidade com este tipo de serviço são colocados em uma fila de maior prioridade que a fila *best-effort,* e são os primeiros a serem encaminhados.

A existência de filas nos roteadores ocorre devido a perda de pacotes, atraso e *jitter*. A banda passante da rede é reservada para o atendimento de um fluxo agregado cuja taxa de

serviço em cada nó é dada por um valor mínimo bem definido, independente da intensidade dos outros tráfegos que transitam neste nó. Os fluxos que compõem esse tráfego devem ser limitados por uma taxa de pico. A agregação de fluxos ocorre na fronteira de um domínio DS que implementa este modelo. Um mecanismo de policiamento atua neste ponto, marcando os pacotes que receberam o tratamento EF, no interior do domínio DS, de acordo com a taxa de pico contratada por cada usuário.

Os roteadores internos do domínio policiam a agregação de fluxos, descartando os pacotes que não estiverem de acordo com as características do tráfego agregado. O inconveniente desta proposta é justamente alocar os recursos de forma radical, onde a capacidade da rede como um todo, é drasticamente reduzida pelo que foi alocado aos fluxos que utilizam o serviço EF. Apesar, disso o mecanismo EF é o mais adequado para o encaminhamento de pacotes que fazem parte de fluxos multimídia (como voz e vídeo-conferência), uma vez que certas garantias podem ser dadas o que não ocorre com o mecanismo AF. O PHB EF possui o valor "101110" no sub-campo DSCP do campo DS.

A Figura 8 [Mag01] ilustra uma configuração de uma rede Diffserv que utiliza o PHB EF no transporte de vídeo MPEG-4. Nesta configuração, utiliza-se um *buffer* onde os pacotes esperam pela disponibilidade de permissões no balde de fichas (*Token Bucket)*. Os pacotes são descartados quando o tamanho do *buffer* for insuficiente. Para que os pacotes possuam um tempo de permanência mínimo na fila, foi utilizado o escalonador de filas PRR (*Priority Round Robin*) ou WRR (*Weighted Round Robin*), os quais podem garantir uma vazão ao tráfego EF, utilizando-se múltiplas filas com diferentes prioridades. O tamanho da fila EF é controlado pelo mecanismo de descarte *Drop Tail,* o qual faz o descarte de pacotes que chegam, se a capacidade de armazenamento da fila for ultrapassada.

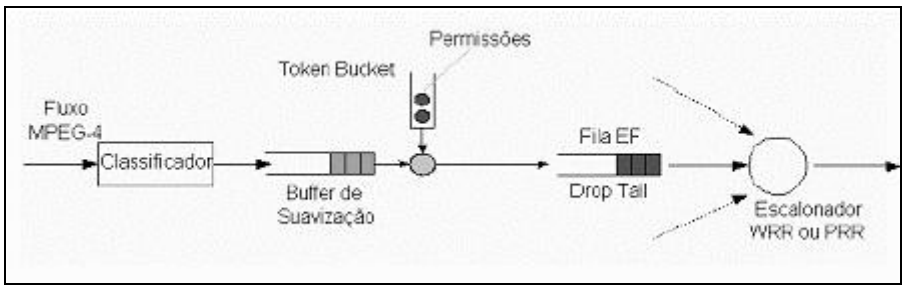

Figura 8: Encaminhamento PHB EF

2.3.3.4.2 PHB AF

O PHB AF (*Per Hop Behavior Assured Forwarding*) é utilizado em um domínio DS que oferece diferentes níveis de garantias de encaminhamento para pacotes IP recebidos de um cliente DS. Os pacotes que desejam usar os serviços providos pelo PHB's AF são marcados pelo usuário ou provedor DS dentro de uma ou mais classes AF de acordo com os serviços que o receptor contratar.

[Hei99] definiu uma proposta de codificação do AF que possui quatro classes, cada uma com três níveis de precedência para descarte de pacotes, ocupando doze códigos DSCP diferentes para implementação do PHB AF. A Tabela 4 identifica essas classes e também as prioridades de descarte dos pacotes. Os pacotes são marcados para cada uma das classes de acordo com o tratamento de encaminhamento desejado pelo cliente.

O PHB AF garante que os pacotes são entregues enquanto o tráfego agregado não exceder a taxa contratada definida no perfil de serviço. No entanto, é permitido ao usuário exceder as taxas contratadas. Ao fazê-lo, contudo, o usuário deve estar consciente de que o tráfego excedente não terá a mesma probabilidade alta de entrega. Em caso de congestionamento nos roteadores ou no nó, o descarte será feito considerando-se a classe e a probabilidade de descarte escolhido.

DESCARTE	CLASSE			
	Serviço Olímpico			Classe 4
	Classe 1/Ouro	Classe 2/Prata	Classe 3/Bronze	
Baixo	AF11=001010	AF21=010010	AF31=011010	AF41=001010
Médio	AF12=001100	AF22=010100	AF32=011100	AF42=100100
Alto	AF13=001110	AF23=010110	AF33=011110	AF43=100110

Tabela 4: Proposta de Codificação AF

De acordo com [Hei99], o serviço olímpico trabalha apenas com três classes: Ouro, Prata e Bronze e com três níveis de descarte: baixo, médio e alto. O serviço ouro trabalha com baixa prioridade de descarte (AF11), pode ser usado para transmissão de algumas aplicações multimídia, desde que sejam elásticas, ou seja, que essas aplicações possam se adaptar a

situações de congestionamento. A Figura 9 [ZIV99] ilustra a implementação do encaminhamento assegurado (PHB AF).

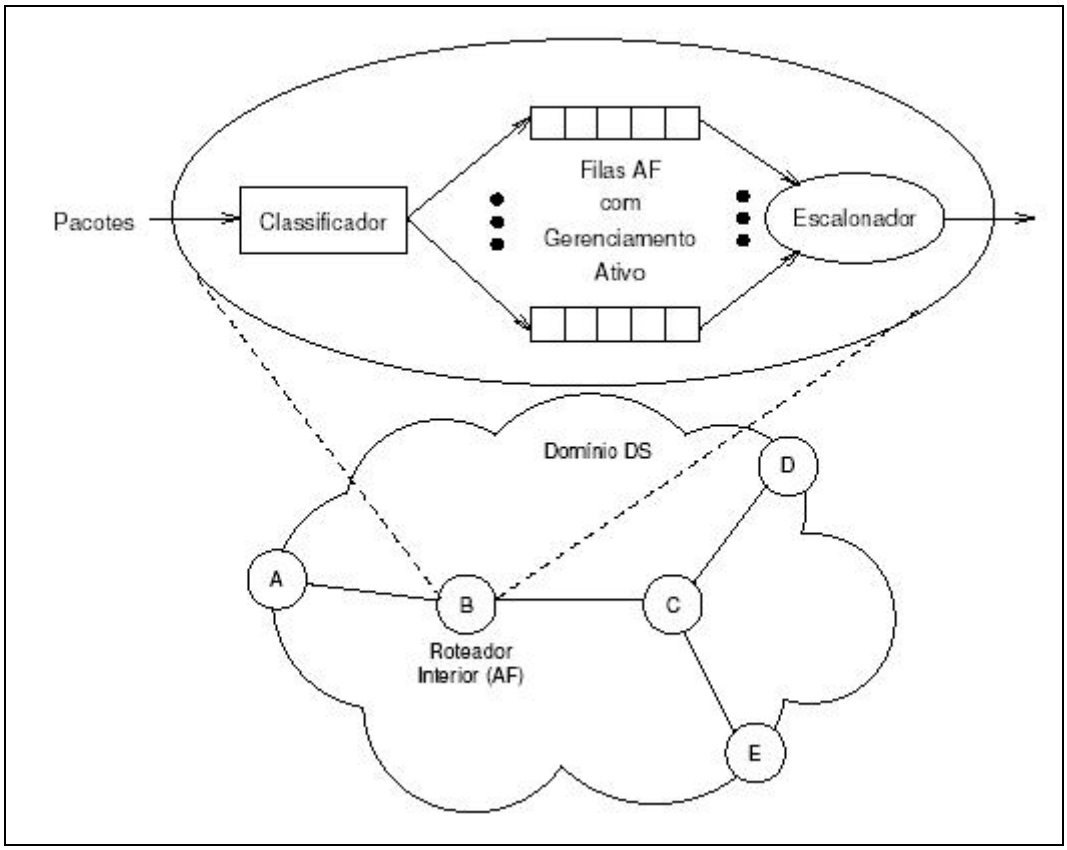

Figura 9: Implementação do encaminhamento assegurado (PHB AF)

Em um roteador ou nó DS, o nível de garantia de encaminhamento de um pacote depende dos seguintes fatores:

- Da quantidade de recursos que foram alocados para a classe AF, o qual o pacote pertence;
- Da carga atual da classe AF;
- Da probabilidade ou nível de descarte dos pacotes.

2.3.5 Engenharia de Tráfego (TE)

A engenharia de tráfego é responsável pela organização do tráfego através da rede. O principal objetivo é otimizar os recursos da rede para alcançar um desempenho satisfatório. As seguintes técnicas são utilizadas para implementar engenharia de tráfego:

- MPLS (*Multiprotocol Label Switching*)**;**

- Roteamento Baseado em QoS - QoSR (*QoS Routing*);
- Roteamento Baseado em Restrições -CBR (*Constraint-Based Routing*).

O congestionamento de uma rede pode ser causado por dois motivos:

1. Falta de recursos da rede ou;
2. Distribuição irregular de tráfego.

No primeiro motivo, todos roteadores e *links* estão sobrecarregados e a solução é aumentar a capacidade da rede, ou seja, prover mais recursos para atualização da infra-estrutura. O segundo motivo, é a distribuição irregular do tráfego, algumas partes da rede estão sobrecarregadas enquanto outras estão levemente carregadas, ou seja, deve ser feito o balanceamento de carga.

Na Internet a distribuição irregular de tráfego pode ser causada pelos atuais protocolos de roteamento, como o RIP (*Routing Information Protocol*), e o OSPF (*Open Shortest Path First*), porque eles sempre utilizam o caminho mais curto para transferir os pacotes. Com isso, roteadores e *links* junto ao caminho mais curto entre os dois nós podem ficar congestionados, enquanto roteadores e *links* junto ao caminho mais longo estarão ociosos.

A Figura 10 [Kam00] ilustra, uma situação em que os pacotes são encaminhados pelo caminho mais curto, enquanto que os outros caminhos estão ociosos, ou seja, há uma degradação do desempenho da rede apesar de haver recursos disponíveis, o que causa congestionamento.

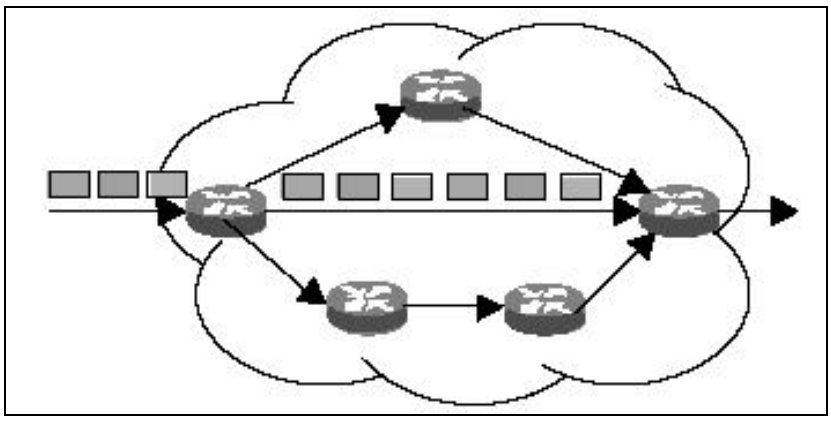

Figura 10: Encaminhamento de pacotes sem engenharia de tráfego

A Figura 11 [Kam00] ilustra uma rede utilizando a engenharia de tráfego, onde os diversos tipos de tráfego seguem por caminhos distintos. Nesse caso, os recursos da rede são utilizados de forma racional, evitando as situações de congestionamento.

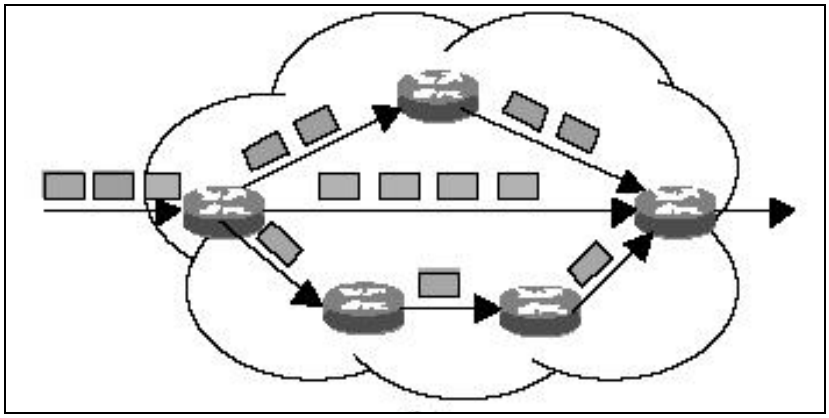

Figura 11: Encaminhamento de pacotes com engenharia de tráfego

2.3.6 MPLS

Um pacote em uma rede é encaminhado de um roteador ao outro.Cada roteador faz uma busca na sua tabela de roteamento e de acordo com o endereço IP do cabeçalho decide para onde encaminhá-lo, ou seja, o *next hop* (próximo salto), isso é feito na camada de rede. Essa consulta na tabela de roteamento é complexa[2] e pode demorar bastante tempo, dependo do tamanho da tabela de roteamento. Para evitar esse tipo de consulta surgiu o MPLS (comutação por rótulos) que utiliza um rótulo de tamanho fixo, onde o roteador decide por onde enviar os pacotes [Odl99]. O MPLS (*Multiprotocol Label Switching*) tem como principal objetivo integrar a funcionalidade e a flexibilidade da camada de rede (utilizando informações de roteamento) com a agilidade e a capacidade de tráfego da camada de enlace. A Figura 12 ilustra o roteamento e a comutação associadas as rotas de tráfego.

[2] Quando o prefixo dos endereços são longos, as consultas envolvem comparações que consomem mais tempo na análise dos pacotes , retardando o envio dos pacotes.

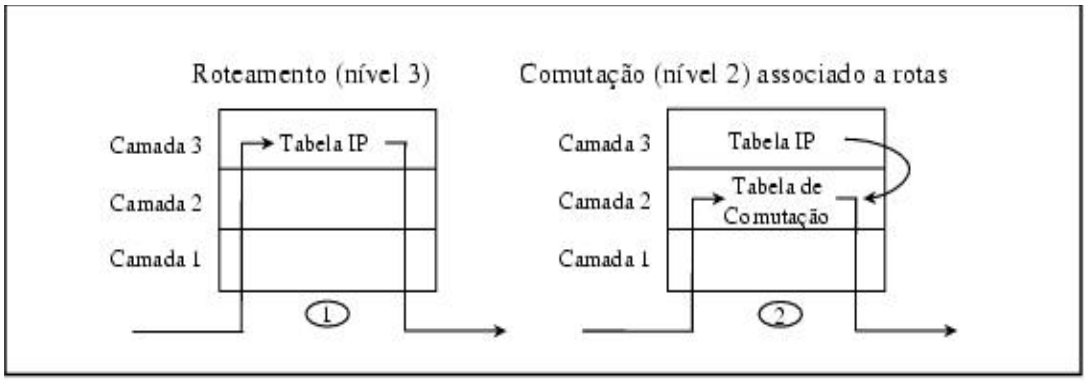

Figura 12: Roteamento e a comutação associadas as rotas de tráfego

O MPLS é uma tecnologia utilizada em *backbones*, e consiste em uma solução para os problemas atuais de rede de computadores como: velocidade, escalabilidade, gerenciamento de qualidade de serviço e a necessidade de engenharia de tráfego. O MPLS é chamado de multiprotocolo porque qualquer protocolo de rede pode ser utilizado, embora só esteja padronizado, por enquanto, para o protocolo IP.

2.3.6.1 Componentes do MPLS

Para implementação do MPLS é necessária a utilização dos seguintes componentes: Rótulo (*Label*), LSR (*Label Switch Routers*), LER (*Label Edge Routers*), LSP (*Label Switch Path*), FEC (*Forwarding Equivalence Classes*) que serão detalhados a seguir.

2.3.6.1.1 Rótulo

Os rótulos são identificadores de fluxos representados por números inteiros e de tamanho fixo. Possui significado local (específico para o domínio em que está atuando) e é usado para identificar uma FEC (*Forwarding Equivalence Classes*). O cabeçalho MPLS deve ser posicionado depois de qualquer cabeçalho da camada 2 e antes de um cabeçalho da camada 3, seu tamanho é definido em 4 *octetos*. Os rótulos comutam diretamente as células para as suas rotas definidas pela camada de rede. Os pacotes podem ser enviados diretamente no nível de enlace através da rede sem sofrer processamento na camada de rede. Assim, o encaminhamento de pacotes fica completamente independente da função de determinação de rotas da camada de rede.

Um rótulo pode ser pensado como uma forma abreviada para o cabeçalho do pacote, de forma a indicar ao pacote a decisão de remessa que um roteador faria. Neste contexto, o

rótulo é uma abreviação para um fluxo agregado de dados de usuário. A Figura 13 ilustra o cabeçalho MPLS.

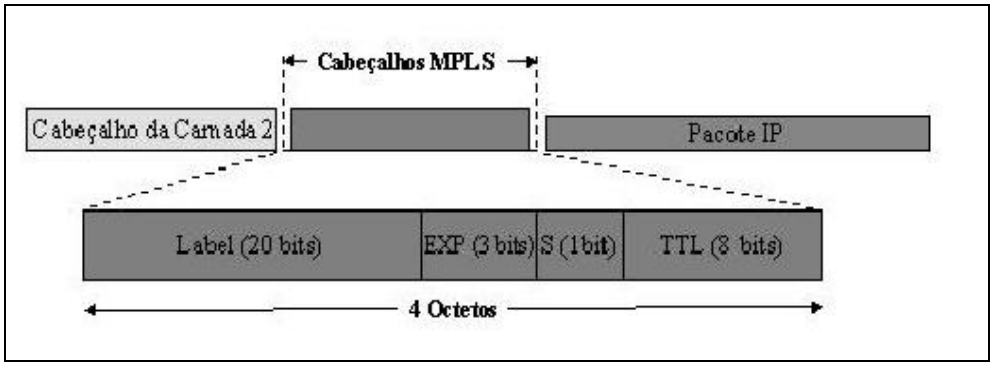

Figura 13: Cabeçalho MPLS

2.3.6.1.2 LSR

Os LSRs (*Label Switch Routers*) são os roteadores de comutação por rótulos. Trata-se de equipamentos de comutação (por exemplo: roteadores IP, *switches* ATM – *Asyncronus Transfer Mode*) habilitados para MPLS. São equipamentos situados no núcleo da rede MPLS, e sua função é encaminhar pacotes baseados apenas no rótulo. Ao receber um pacote, cada LSR troca o rótulo existente por outro, passando o pacote para o próximo roteador e assim por diante.

2.3.6.1.3 LER

Os LERs (*Label Edge Routers*) são LSRs situados na periferia da rede MPLS como ilustrado na Figura 14 [Mes01]. Eles ligam diversas sub-redes (*Ethernet*, *FrameRelay*, ATM) à rede MPLS e são responsáveis pela designação e retirada de rótulos para o tráfego que entra ou sai da rede MPLS.

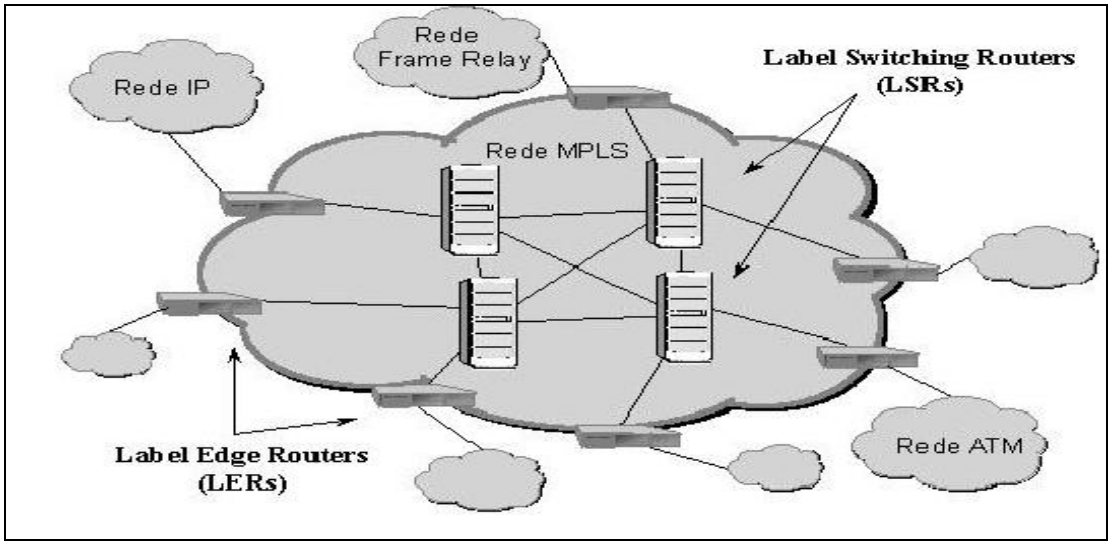

Figura 14: Ambiente MPLS

2.3.6.1.4 LSP

O LSP (*Label Switch Path*), trata-se de um caminho através de uma seqüência ordenada de LSRs, estabelecido entre uma origem e um destino. Um LSP é unidirecional, portanto é preciso ter dois LSPs para uma comunicação entre duas entidades.

Num LSP (caminho comutado por rótulo) transitarão pacotes de uma mesma classe, que compartilham o mesmo destino. Assim, uma rota deve ser estabelecida inicialmente, isto é feito através de protocolos de roteamento convencionais ou roteamento com restrições, então o caminho será definido e os pacotes pertencentes a ele não precisarão mais ser roteados, serão apenas comutados com base nos seus rótulos. Estes rótulos são distribuídos entre LSRs no momento do estabelecimento de LSPs.

2.3.6.1.5 FEC

Para decidir o próximo ponto da rede são necessárias duas funções: a primeira divide o conjunto de pacotes inteiros possíveis em uma FEC (*Forwarding Equivalence Classes*) classe de equivalência de encaminhamento. A segunda mapeia cada FEC para algum roteador que é o próximo ponto. Cada FEC pode ser vista com uma entrada na tabela de roteamento, onde todos os pacotes mapeados para uma FEC são iguais, com relação à decisão de encaminhamento, ou seja, todos os pacotes que pertencem a uma determinada FEC e que trafegam pelo roteador seguirão o mesmo caminho.

O MPLS desvincula essas duas funções, atribuindo um pacote a uma FEC específica apenas uma vez, quando o pacote entra na rede. A informação sobre a FEC à qual um pacote

pertence é codificada como um rótulo que é inserido no pacote. Quando o pacote é enviado ao seu ponto seguinte, o rótulo será emitido junto com ele. Os pacotes serão rotulados antes de serem enviados.

Nos roteadores subseqüentes, o cabeçalho do pacote não é mais analisado e não há mais busca na tabela de roteamento, com isso evita-se o retardo de análise dos cabeçalhos dos pacotes na camada de rede. O rótulo é utilizado como um índice em uma tabela que especifica o próximo ponto e também o novo rótulo. A distribuição de rótulos não é global em um domínio, ela é local a um determinado par de roteadores. O roteador troca o rótulo antigo pelo novo e encaminha o pacote para o próximo ponto da rede.

2.3.6.2 Vantagens do MPLS

As vantagens com o uso do MPLS em relação ao encaminhamento IP tradicional são as seguintes:

- Melhor desempenho no encaminhamento de pacotes;
- O algoritmo que faz o mapeamento FEC/rótulo pode ser cada vez mais complexo, sem que isso tenha nenhum impacto no desempenho dos roteadores de núcleo;
- Possibilidade de associar requisitos de QoS baseados nos rótulos carregados pelos pacotes.

2.3.6.3 Aplicações do MPLS

Uma das principais aplicações para MPLS hoje é a Engenharia de Tráfego. O MPLS consegue forçar os pacotes a seguirem certas rotas pré-estabelecidas, o que seria impossível no esquema tradicional de encaminhamento IP.

O MPLS também permite facilmente a configuração de Redes Privadas Virtuais (VPNs- *Virtual Private Network*). As Empresas usam VPNs para criar túneis seguros entre matrizes e filiais ou entre os seus parceiros comerciais. Normalmente, esse tipo de comunicação utiliza linhas privadas, porque a Internet é considerada insegura para transportar informações confidenciais. Com MPLS pode-se criar uma VPN usando o serviço de emulação de *Frame Relay*, ou então, configurando roteadores MPLS nas redes dos usuários para serem

o início e o fim de LSPs. Dentro dos LSPs a comunicação apresenta um alto nível de segurança.

2.3.7 Roteamento Baseado em QoS (QoSR)

Os protocolos atuais de roteamento RIP (*Routing Information Protocol*) e OSPF (*Open Shortest Path First*) utilizados na Internet utilizam como parâmetro o menor caminho, mesmo que esteja congestionado.

O roteamento baseado em QoS (QoS *Routing*) [Cra98], surgiu para evitar o desperdício de recursos. O QoSR é um mecanismo de roteamento que seleciona o caminho percorrido pelos pacotes de um fluxo baseado no conhecimento da disponibilidade de recursos da rede, bem como nos requisitos de QoS dos fluxos, como largura de banda e atraso. Desta forma, o roteamento baseado em QoS pode encontrar um caminho mais longo, mas muito menos sobrecarregado que o caminho mais curto que geralmente é o mais congestionado.

Vale a pena ressaltar que QoSR, não é uma técnica de oferecimento de QoS, ela apenas informa qual a rota mais apropriada para que os níveis de QoS possam ser atendidos. É necessária a utilização de alguma técnica de QoS, como o IntServ ou DiffServ. Além disso, o roteamento baseado em QoS não é um mecanismo de encaminhamento, ou seja, pode descobrir novas rotas, mas não tem como forçar pacotes de certos fluxos a seguirem obrigatoriamente essas rotas.

2.3.8 Roteamento Baseado em Restrições

O roteamento baseado em restrições CBR (*Constraint-Based Routing*) desenvolveu -se a partir do roteamento baseado em QoS. O CBR é o processo de computar rotas que são sujeitas a múltiplas restrições, onde as restrições são requisitos de QoS. Os objetivos do roteamento baseado em restrições são:

- Aumentar a utilização da rede, otimizando os recursos usados baseado no esquema de roteamento, podendo utiliza-se de forma mais eficiente à vazão total da rede;
- Determinação dinâmica dos caminhos possíveis, ou seja, selecionar rotas que possam encontrar os requisitos de QoS do fluxo.

Enquanto determina uma rota, C*onstraint-Based Routing* considera não só a topologia da rede, mas também a solicitação de fluxos, disponibilidade de recursos como largura de banda, e possivelmente outras políticas especificadas pelo administrador da rede [Xia99].

2.3.9 Considerações Finais

Neste capítulo foram apresentadas propostas para implementação de Qualidade de Serviço na Internet, como as arquiteturas utilizadas para o provimento de QoS. Definiu-se os parâmetros e conceitos de QoS. Discutiu-se a utilização do IntServ através da arquitetura RSVP para as aplicações reservar recursos da rede, foi exposto as desvantagens dessa arquitetura mais especificadamente do RSVP. Falou-se do funcionamento da arquitetura DiffServ que trabalha com a priorização de pacotes e surgiu para resolver o problema da falta de escalabilidade encontrada no IntServ. Estudou –se os tipos de PHB's como é o caso do BE, EF, AF utilizados na implementação de Serviços Diferenciados.

Por fim, falou-se sobre o mecanismo de engenharia de tráfego e as técnicas utilizadas para sua implementação como: MPLS (*Multiprotocol Label Switching*)**,** Roteamento Baseado em QoS - QoSR (*QoS Routing*).

No capítulo seguinte, será apresentada uma visão geral dos protocolos de transporte TCP e UDP.

Capítulo 3.0
Os Protocolos TCP e UDP

Este capítulo descreve o funcionamento dos protocolos TCP e UDP, mostrando também as principais características desses protocolos de transporte.

3.1 Histórico do Protocolo TCP

O protocolo TCP (*Transmission Control Protocol*) foi formalmente padronizado e definido pelo DOD (*Department of Defense*) Departamento de Defesa Americano e foi publicado pelo DARPA (*Departament Advanced Research Projetct Agency*) na RFC 793 (*Request for Comments*) [Pos81].

A criação desse protocolo teve como principal objetivo a sua utilização nas aplicações militares. Era interesse do DOD permitir a comunicação entre os diversos sistemas espalhados pelo mundo, no caso de ocorrer, por exemplo, um desastre nuclear. Os sistemas poderiam ser de diferentes fabricantes, possuir diferentes sistemas operacionais, como também diferentes topologias e protocolos. Hoje em dia, o TCP é o protocolo de transporte mais utilizado em todas as redes do mundo.

3.2 Definição do Protocolo TCP

O TCP é um protocolo da camada de transporte orientado a conexão, pois antes de transferir dados é necessário que seja estabelecida uma conexão com o *host* destino, depois ocorre à transferência dos dados e por fim o encerramento da conexão. O protocolo TCP oferece um serviço confiável tendo em vista que os dados são transmitidos integralmente para os destinos corretos. De acordo com [Tan96] o TCP foi projetado para trabalhar com uma "inter-rede" não confiável. Como o TCP trabalha com o protocolo imediatamente inferior a sua camada o IP, que não oferece nenhuma garantia de envio de seus pacotes, cabe então ao TCP cuidar para que as mensagens do IP sejam encaminhadas de forma confiável através da rede, tratando das retransmissões, do ordenamento e da integridade de cada segmento.

O TCP é responsável pela segmentação, ou seja, divide os dados a serem transmitidos em pequenos blocos - os segmentos - que são identificados para, no *host* destino, serem agrupados novamente. O tamanho dos segmentos são definidos de acordo com a capacidade da rede.

Para controlar os erros, é realizada uma identificação nos datagramas antes da sua transmissão. Além disso, é adicionado o *checksum*, um campo numérico utilizado para verificar erros na transmissão, ver maiores detalhes em [Tan96] e [Com94]. No controle de fluxo, o TCP é capaz de adaptar a transmissão dos segmentos às condições de transmissões (velocidade, tráfego etc.) entre os sistemas.

3.3 Modelo de Serviço do TCP

As informações transmitidas entre os computadores podem ser originadas de diversas aplicações diferentes, como um E-Mail, um acesso a HTTP (*Hyper Text Transfer Protocol*) ou uma transferência de arquivos FTP (*File Trasfer Protocol*). Para que seja possível identificar a que serviço um determinado datagrama pertence, o TCP utiliza o conceito de *socket,* que pode ser definido como a representação numérica de um serviço na Internet. O *socket* é identificado pelo endereço IP da máquina mais um número de 16 *bits* de identificação TCP chamado "porta".

Um *socket* pode ser utilizado por várias conexões ao mesmo tempo, mas uma combinação entre *socket* de origem e *socket* de destino nunca se repete, simultaneamente. Determinado o *socket*, toda a comunicação com a aplicação é realizada e endereçada através dele. A Tabela 5 [Tan96] ilustra os tipos de serviços do TCP com o respectivo número da porta:

Serviço	Porta
HTTP (WWW)	80
HTTPS (WWW seguro)	443
IRC (Chat, ou bate-papo)	6667
FTP (Transferência de arquivos)	21
TELNET (Emulação de terminal)	23
SMTP (Transferência de E-Mail)	25

Tabela 5: Serviços do TCP com o número da porta

As conexões que o TCP estabelece são *full-duplex,* ou seja, a transferência dos dados pode ocorrer em ambos os sentidos, possibilitando que os dois *hosts* conversem durante a transmissão. Cada conexão TCP é, também, ponto-a-ponto porque estabelece a comunicação entre dois pontos.

3.4 O Cabeçalho do Segmento TCP

O segmento TCP é formado por duas partes: a primeira parte é o cabeçalho, de onde são originadas as informações de controle necessárias para a transmissão, e a outra parte é composta dos dados a serem transportados. O cabeçalho do TCP possui um tamanho fixo de 20 *bytes* e mais um campo opcional de tamanho variável. O tamanho do cabeçalho é

calculado em múltiplos de 32 *bits*. Os segmentos sem dados são válidos e em geral são utilizados para confirmações e mensagens de controle. A Figura 15 ilustra o segmento TCP segundo [Pos81].

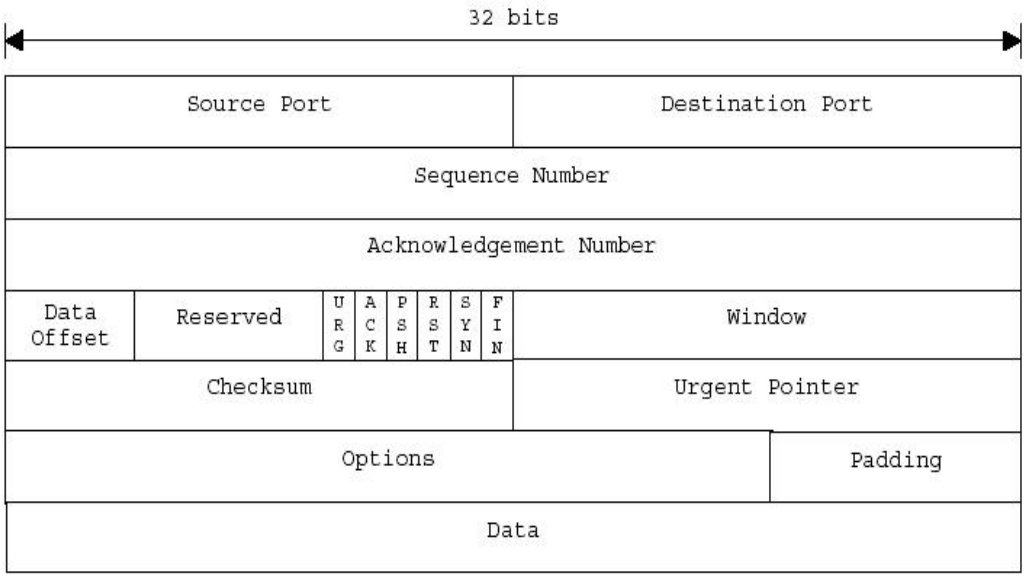

Figura 15: Layout do segmento TCP

O campo *Source Port* (16 *bits*-porta de origem) e *Destination Port* (16 *bits*-porta de destino) identificam a conexão da máquina de origem e de destino. Cada *host* decide como alocar suas próprias portas começando com a porta de número 256. Uma porta e o endereço IP de seu *host* formam um TSAP (*Transport Service Access Point*) único de 48 *bits*. Os números dos *sockets* de origem e destino identificam a conexão.

O campo *Sequence Number* (32 *bits*) indica o número de seqüência do segmento enviado. O número de seqüência inicial é definido no momento em que a conexão estiver sendo estabelecida.

O campo *Acknowledgement Number* (32 *bits*) é o número de reconhecimento que indica o próximo *byte* aguardado e informa que o último *byte* foi recebido corretamente.

O campo *Data Offset* (4 *bits*) é chamado também de *Header Length* (comprimento do cabeçalho), e indica onde os dados iniciam dentro do segmento, o seu tamanho é múltiplo de 32 *bits*, zeros devem ser preenchidos no final para atender ao requisito do tamanho.

O segmento do TCP tem seis *bits* não utilizados seguidos de seis *flags*, são:

URG = Indica que o campo *Urgent Pointer* é utilizado se o valor é “1”;

ACK = Possui o valor "1" se o campo *Acknowledgement Number* estiver sendo usado para indicar o recebimento de um segmento. Caso *ACK* seja "0", significa que este segmento não carrega informação de confirmação de recebimento de outro segmento;

PSH = Indica dados com a *flag PUSH*, ou seja, informa ao TCP destino que os dados contidos no segmento devem ser encaminhados imediatamente à aplicação;

RST = Serve para reiniciar ou rejeitar uma conexão, é utilizado quando há algum problema na conexão;

SYN = Utilizado para solicitar o estabelecimento de uma conexão. Este *bit* é sempre "1" no primeiro segmento enviado, no início da conexão. A resposta a esse pedido é confirmada em outro segmento com as *flags SYN* e *ACK* iguais a "1" (denominados: *Connection Request e Connection Accepted*, respectivamente);

FIN = Utilizado para encerrar uma conexão. É uma indicação de que o transmissor não tem mais dados a enviar, porém, não indica uma imediata quebra da conexão para o receptor, permitindo que dados ainda em trânsito na rede possam ser recebidos.

O campo *Window* (16 *bits*) informa o tamanho da janela oferecido pelo receptor *RWND* (*Receiver Adversited Window)*. Indica quantos *bytes* podem ser enviados a partir do *byte* confirmado (*Acknowledgement Number*), maiores detalhes estão ilustrados na seção 3.7.1.

O campo *Checksum* (16 *bits*) é utilizado para aumentar a confiabilidade. Ele faz a checagem da integridade dos dados e do cabeçalho. A soma de todos os *bytes* mais o *Checksum* deve ser igual a zero.

O campo *Urgent Point* (16 *bits*) permite a um extremo (da conexão) dizer ao outro para saltar o processamento para um *octeto* particular.

O campo *Options* foi projetado como forma de oferecer recursos extras, ou seja, recursos que não foram previstos pelo cabeçalho comum. Uma de suas funções é informar o tamanho máximo dos segmentos MSS (*Maximum Segment Size*) que pode ser recebido pelo *host*, a opção *MSS* contém 16 *bits*.

O campo *Padding* é um campo variável, preenchido com zeros, usado para especificar o final do cabeçalho e o início dos dados.

O campo *Data* é a segunda parte do segmento. Logo após o final do cabeçalho, onde iniciam os dados que serão transportados.

3.5 Gerenciamento de Conexão

No protocolo TCP, o gerenciamento de uma conexão segue três fases que são: estabelecimento de conexão, manutenção da conexão e encerramento de uma conexão.

3.5.1 Estabelecimento de Conexão

Para estabelecer uma conexão o TCP utiliza o *Three Way Handshake* [Tan96], que significa, literalmente, "um aperto de mão em três vias", onde são necessários apenas três segmentos para que a conexão seja estabelecida. O *Three Way Handshake* funciona da seguinte forma: o *host* que deseja fazer um pedido de conexão envia um segmento com a *flag SYN* ativo, assim que o outro *host* recebe o pedido de conexão, ele envia um segmento contendo as *flags* (*ACK+SYN*) confirmando o recebimento, caso esteja interessado e preparado para estabelecer a conexão.

Para finalizar o estabelecimento da conexão, o primeiro *host* envia um segmento com a *flag ACK*, indicando ter aceitado a conexão com o *host,* e a partir deste momento, a conexão é considerada estabelecida pelo primeiro *host*. Somente quando o terceiro segmento é recebido pelo segundo *host*, é que este considera a conexão estabelecida.

A partir do momento em que um *host* considera uma conexão estabelecida, ele pode iniciar a transmissão de dados. A Figura 16 ilustra o estabelecimento de uma conexão no TCP.

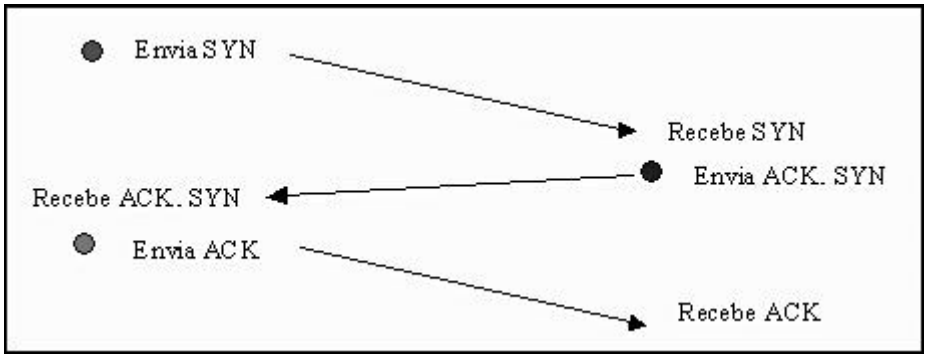

Figura 16: Estabelecimento de uma conexão no TCP

3.5.2 Mantendo uma Conexão no TCP

O mecanismo de "*keep-alive*" serve para controlar o estado de uma conexão, ou seja, para checar se uma conexão está ociosa ou se ela continua ativa.

O "*keep-alive*" consiste em enviar um segmento, sem dados, para a outra ponta da conexão quando ocorreu um longo intervalo de tempo sem o recebimento de um segmento de dados ou de um *ACK*, segundo [Bra89] este intervalo não deve ser menor que 2 horas. A outra ponta deve enviar um *ACK* para o segmento do "*keep-alive*" com o objetivo de garantir que a conexão ainda está ativa, caso contrário, a conexão pode ser encerrada. Mas, de acordo com [Bra89], o "*keep-alive*" não deve interpretar a falha na confirmação de qualquer requisição como uma conexão ociosa. O autor, também afirma que não existe mecanismo especial para temporizar uma conexão que não responde as requisições de um "*keep-alive*", e se o mecanismo não permitir um amplo espaço de tempo em uma rede congestionada ou com atraso, conexões serão encerradas desnecessariamente. Por isto, alguns autores não são a favor da implementação do "*keep-alive*". O "*keep-alive*" consome banda desnecessariamente e tem alto custo para uma rota cheia de pacotes.

3.5.3 Encerrando uma Conexão

O método utilizado para encerrar uma conexão é similar ao *Three Way Handshake*, com duas diferenças, a primeira é o tipo de utilização da *flag*, para estabelecer uma conexão utiliza-se a *flag SYN*, para finalizar a conexão utiliza-se a *flag FIN*. A segunda diferença é que o segundo segmento não inclui a *flag FIN* junto com o *ACK*. Primeiro, vai um segmento com a *flag FIN*, indicando que um dos *host*s não tem mais dados para enviar e quer encerrar a conexão. Em resposta ao pedido de encerramento de conexão, o outro *host* envia um segmento com a *flag ACK*, avisando que recebeu o pedido. Este deve então avisar a aplicação que não existem mais dados para receber e perguntar se pode encerrar a conexão. Caso a aplicação permita o encerramento da conexão, o TCP envia o segmento com a *flag FIN* para o outro, que por sua vez, envia um *ACK* finalizando a conexão. A Figura 17 ilustra o encerramento de uma conexão.

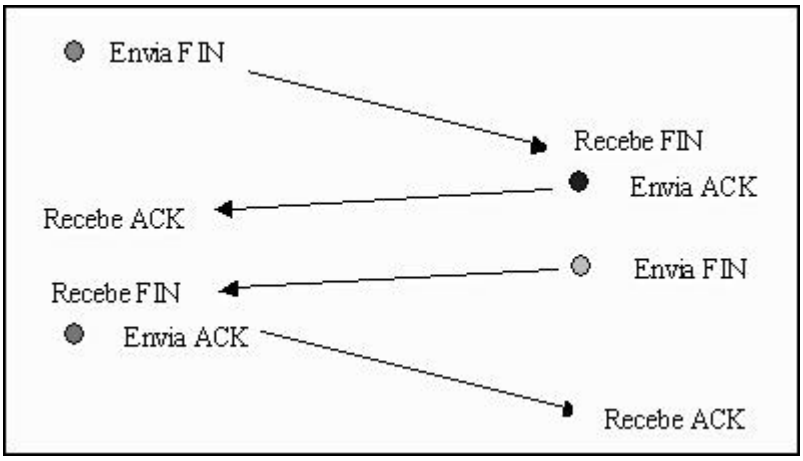

Figura 17: Encerramento de uma conexão TCP

3.6 Controle de Congestionamento do TCP

Na Internet, a ocorrência de congestionamento é devida ao serviço do melhor esforço não ter nenhum tipo de controle de admissão nem reserva de recursos. Pode-se definir congestionamento como uma situação em que a demanda pelos recursos da rede ultrapassa a sua capacidade. O controle de congestionamento tem como principal função alocar recursos de modo que a rede opere da forma mais eficiente possível nesse tipo de situação. Entre os recursos de uma rede, podem ser citados: a largura de banda dos enlaces, o espaço em *buffers* e o tempo de processamento nos nós intermediários da rede.

Há duas atividades distintas relacionadas ao controle de congestionamento [Jai92] [Lef96] [Jac98], a primeira é a prevenção do congestionamento que tenta detectar possíveis condições que levem a congestionamentos futuros e executar procedimentos para impedi-los. A segunda atividade é a recuperação da rede para que ela volte ao seu estado normal quando ocorre um congestionamento.

Na Internet o controle de congestionamento fica a cargo do protocolo de transporte TCP. A taxa de transmissão de dados do protocolo TCP é controlada pelas condições da rede e por isso os fluxos TCP são chamados de “compreensivos” ou “responsáveis”, porque respondem positivamente as notificações de congestionamentos [Bra98].

3.7 Algoritmos para Controle de Congestionamento do TCP

Os algoritmos de controle de congestionamento do protocolo TCP foram criados por [Jac88] e padronizados por [Bra99], e são quatro: *Slow Start* (inicialização lenta), *Congestion*

Avoidance (evitar congestionamento), *Fast Retransmit* (retransmissão imediata), *Fast Recovery* (recuperação rápida). Apesar de serem independentes, esses algoritmos são geralmente implementados de forma conjunta. A seguir, explicaremos de forma detalhada o funcionamento de cada um dos algoritmos de controle de congestionamento do protocolo TCP.

3.7.1 Algoritmo *Slow Start e Congestion Avoidance*

O algoritmo S*low Start* é baseado na seguinte regra: a taxa de envio dos novos pacotes na rede deve ser a mesma utilizada pelas confirmações *ACK* (confirmação do recebimento de um ou mais segmentos enviados) que foram enviadas pelo destino. A taxa inicial de transmissão dos pacotes é geralmente, igual a uma ou duas vezes o tamanho máximo de um segmento suportado pelo transmissor *MSS* (*Maximum Segment Size*).

À medida que uma transmissão é bem sucedida, ou seja, o receptor envia um *ACK* confirmando que recebeu o segmento esperado, a quantidade de dados é aumentada. O protocolo TCP armazena o tamanho do pacote em uma variável denominada "Janela de Congestionamento" (*CWND - Congestion Window*). O algoritmo *Slow Start* sempre utiliza a janela de congestionamento (*CWND*) que é o limite que o transmissor tem para transferir os dados quando inicia a transmissão, ou depois de receber uma confirmação. O TCP, por sua vez também, restringe a taxa de transmissão através da janela oferecida pelo receptor *RWND* (*Receiver Window*). Essa janela informa o limite que o receptor tem para receber os dados, ela é transmitida no cabeçalho de seus segmentos de reconhecimento e/ou de dados. O tamanho da janela de transmissão será definido entre o valor da janela informada pelo receptor *RWND*[3] e o valor da janela do transmissor *CWND*[4] .

A janela de congestionamento do TCP tem como valor uma variável mantida pelo protocolo e é definida como *IW* (*Initial Window*) segmento, embora tradicionalmente seja ajustado para o tamanho de um *MSS*. Existem algumas propostas de utilização de tamanhos maiores para *IW* (valores de até 4**MSS*), e podem ser vistas em [All98] e [Pod98].

A *CWND* tem o seu tamanho especificado em *bytes*, mas o *Slow Start* sempre incrementa em segmentos. O emissor começa transmitindo um segmento e aguarda pelo *ACK* correspondente, quando este *ACK* é recebido, a janela de congestionamento - *CWND* é

[3] A *Receiver Window* refere-se ao controle de fluxo imposto receptor
[4] A *Congestion Window* refere-se ao controle de fluxo imposto pelo transmissor

incrementada em uma vez o tamanho do *MSS* transmissor (*SMSS – Sender Maxium Segment Size)* e dois segmentos podem ser enviados. Assim que os dois segmentos forem confirmados através do *ACK*, a janela de congestionamento - *CWND* é incrementada de quatro vezes e assim por diante. Isso acontece enquanto *CWND* tiver um tamanho inferior a um limite pré-estabelecido denominado de *SSTHRESH* (*Slow Start Threshold* – o limite do *Slow Start*). O valor inicial de *SSTHRESH* deve ser igual ao valor do *RWND*, para que a taxa de transmissão cresça de forma exponencial durante o período em que *CWND* seja menor que *SSTHRESH*, o que implica em um rápido crescimento inicial.

Quando o valor de *CWND* ultrapassar *SSTHRESH*, começará a ocorrer o descarte dos pacotes. Este fato simboliza que a janela de congestionamento ficou muito grande, então o crescimento passa a ser linear num processo chamado de *Congestion Avoidance* (evitar congestionamento).

O Gráfico 1 [Nla02] ilustra o funcionamento do *Slow Start* operando juntamente com o *Congestion Avoidance*. Durante o quinto *Round Trip Time*[5] (representa o tempo que um pacote necessita para ir e voltar da fonte de dados ao destino), quando o *CWND*[6] (janela de congestionamento) é ajustado a 32 *KBytes*, observa-se que um *ACK* duplicado indica que os dados foram perdidos, ou seja, a janela ultrapassou a capacidade da rede. Assim a variável *SSTHRESH*, é ajustada a metade do valor da janela no instante da detecção da perda, ou seja, 16 *KBytes*. Em seguida a janela é reinicializada em um segmento e seu crescimento se dá de forma exponencial, até que se atinja o limite de *SSTHRESH*, de acordo com o *Slow Start*. Quando o tamanho de *CWND* atinge o valor de *SSTHRESH* o protocolo passa a operar em *Congestion Avoidance* (entre o RTT 10 e 11) realizando um aumento linear da janela. Se uma perda já for notada na fase de *Congestion Avoidance*, o comportamento do TCP será o mesmo.

[5] Corresponde ao eixo X do gráfico 1
[6] Corresponde ao eixo y do gráfico 1 e informa o tamanho da janela de congestionamento em *KBytes*.

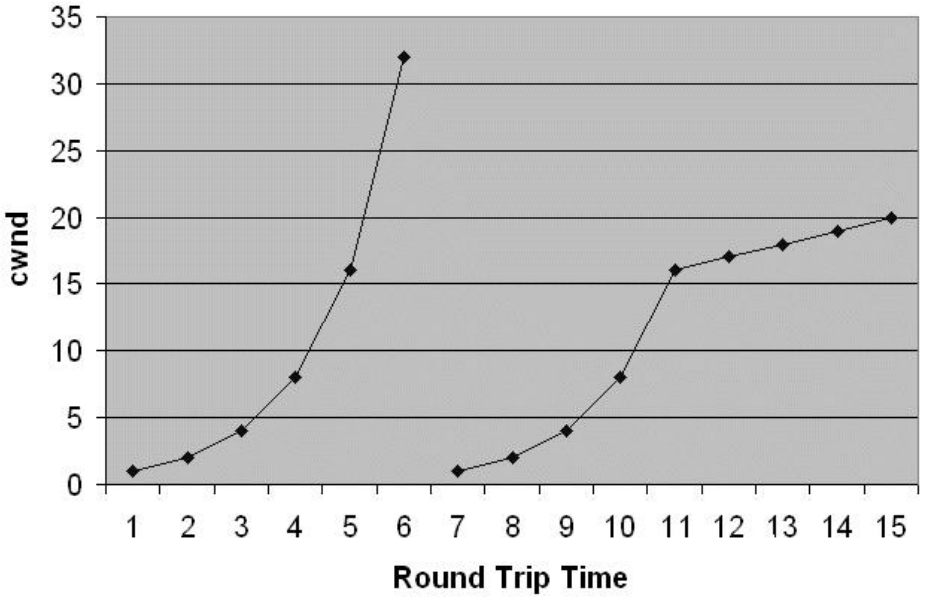

Gráfico 1 : Funcionamento do Slow Start e Congestion Avoidance

3.7.2 Algoritmo *Fast Retransmit*

De acordo com [Ste97] o algoritmo *Fast Retransmit* (retransmissão imediata) surgiu de algumas modificações no algoritmo *Congestion Avoidance* propostas por [Jac98], que visa a agilizar a retransmissão de segmentos quando a rede se encontra em uma situação de congestionamento moderado. Quando um segmento chega fora de ordem, o TCP receptor gera um *ACK* duplicado em um segmento de reconhecimento de dados, para avisar ao transmissor que um segmento fora de ordem foi recebido e informa o número de seqüência do *octeto* esperado. O TCP receptor deve enviar um *ACK* duplicado toda vez que receber um segmento com *octetos* com o número de identificação acima do esperado. O transmissor, por sua vez, não sabe se o segmento está realmente fora de ordem ou se o segmento foi perdido.

Normalmente, o TCP transmissor aguarda a chegada de outros *ACKs* duplicados para saber o que fazer, pois se forem segmentos fora de ordem, com um ou dois *ACKs* duplicados o receptor consegue ordenar. O algoritmo *Fast Retransmit* é usado *s*e três ou mais *ACKs* duplicados forem recebidos seguidamente, este fato é um forte indício que o segmento foi perdido. A Figura 18 [Mor02] ilustra o funcionamento do *Fast Retransmit*.

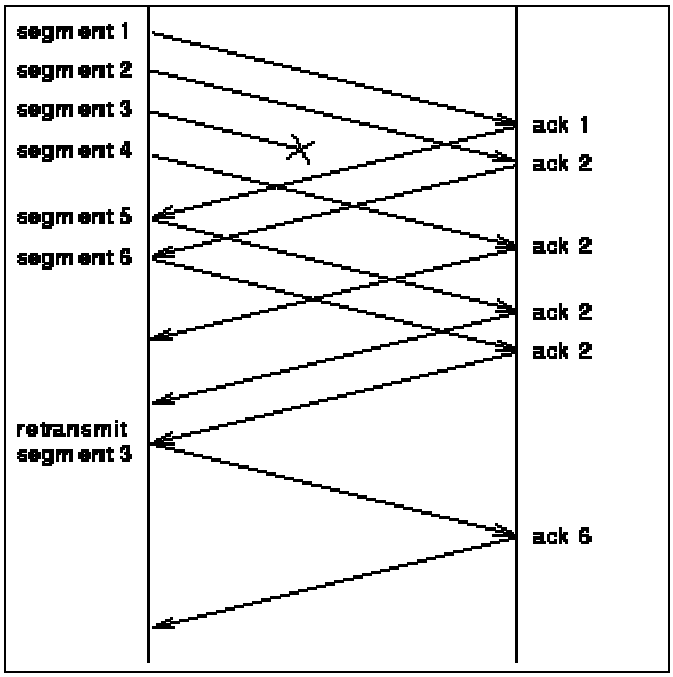

Figura 18: Funcionamento do *Fast Retransmit*

Então o TCP realiza a retransmissão imediata do segmento que foi perdido sem esperar o (RTO - *Retransmit Time Out*) [All99] expirar e aciona o algoritmo *Fast Recovery* que será visto na próxima seção.

3.7.3 Algoritmo *Fast Recovery*

O algoritmo *Fast Recovery* é uma modificação no método *Fast Retransmit*, implementada na versão Reno 1990, uma outra alteração foi realizada para a versão *New Reno* (ver maiores detalhes em [Flo99a]). De acordo com [All99], o transmissor, depois de receber três *ACKs* duplicados e retransmitir o segmento que foi perdido, não executa o *Slow Start*. A razão para não executar o *Slow Start* é que talvez o receptor queira dizer que há mais de um segmento perdido, e que o receptor somente pode enviar um *ACK* duplicado após ter recebido um segmento de dados. Então se, por caso, não houver segmentos em trânsito, o transmissor não deve abruptamente parar de enviar segmentos, como é feito no *Slow Start*.

O *Fast Recovery* implementado na versão Reno faz com que, quando o terceiro *ACK* duplicado chegue, a janela de congestionamento seja reduzida pela metade do seu valor atual (*Multiplicative Decrease*), mas ela não deve ficar menor que dois segmentos. Depois de retransmitir o segmento perdido, a janela de congestionamento *CWND* deve ser incrementada em (*SSTHRESH* + 3**SMSS*) o valor de um segmento de dados. Essa soma adicional de três

tamanhos de segmento é chamada "*Inflação Artificial*" da janela de congestionamento e corresponde à compensação dos três segmentos que dispararam os *ACKs* duplicados, e que por conseqüência já deixaram a rede chegando íntegros ao receptor.

Durante o *Fast Recovery*, cada *ACK* duplicado adicional recebido incrementa *CWND* em (1**SMSS*), continuando o processo de "*Inflação Artificial*" da janela, já que outro segmento saiu da rede, chegando ao receptor. Neste momento, caso o novo valor de *CWND* e também de *RWND* permitam, um novo segmento é enviado.

Ao receber um *ACK* que confirme o recebimento do segmento retransmitido no início do *Fast Recovery*, ou seja, quando o receptor indicar que espera um *octeto* de numeração superior à maior numeração de segmentos enviada até o momento da retransmissão, o TCP transmissor ajusta *CWND* para o valor atual de *SSTHRESH* (no processo de "*Deflação da Janela*") e sai do *Fast Recovery*, voltando ao seu processo normal de controle de congestionamento (*Slow-Start* ou *Congestion Avoidance*). No entanto, caso algum segmento intermediário dentro da mesma janela de congestionamento daquele retransmitido também seja perdido, o problema do congestionamento ainda não terá sido resolvido. Neste caso, as primeiras implementações do TCP que já continham *Fast Recovery* (chamadas "*TCP-Reno*") permaneciam neste algoritmo até que fosse detectada a perda dos segmentos seguintes por expiração do tempo RTO (*Retransmit Time Out*), com conseqüente retorno ao *Slow-Start*.

3.8 O Protocolo UDP

O protocolo UDP (*User Datagram Protocol*) é um protocolo da camada de transporte não orientado a conexão e não confiável. Além disso, o UDP não faz identificação de número de seqüência para ordenamento de dados, não possui controle de fluxo e não divide os dados em datagramas. O protocolo UDP fornece o número das portas usadas de maneira similar ao TCP, ou seja, vários programas podem utilizar o protocolo UDP ao mesmo tempo. Existem várias aplicações na Internet que utilizam o protocolo UDP, entre elas: o DNS (*Domain Name System*), e o SNMP (*Simple Network Management Protocol*).

O protocolo UDP é amplamente utilizado, em aplicações onde à perda ou o re-ordenamento de um segmento de mensagem seja pouco relevante ou quando a retransmissão de um pacote seja mais problemática (devido ao atraso causado) do que sua perda, como no caso da transmissão de dados de voz ou de vídeo. Isso é bastante comum em aplicações de tempo real, como um vídeo animado em que é preferível deixar de mostrar um quadro da

animação e continuar mostrando os próximos quadros, para manter o sincronismo com o som relacionado. Uma parada para espera de algum dado atrasado (por re-ordenamento na rede ou por perda e retransmissão) faria com que toda a apresentação a partir deste ponto ficasse sem sincronismo, tornando-a muito mais indesejável do que uma rápida falha de visualização.

É importante frisar, que devido à simplicidade do protocolo UDP, ele é bastante utilizado em aplicações com maior requisito de desempenho do que de confiabilidade. O *overhead* causado pelo UDP é bem inferior aos que existem nos protocolos orientados a conexão como o TCP e o SCTP, pois o UDP dispensa a troca de informações de estabelecimento e encerramento da conexão. Um pacote no protocolo UDP possui um cabeçalho e dados que serão vistos a seguir.

3.8.1 O Cabeçalho do Segmento UDP

O cabeçalho UDP é composto de oito *bytes*. A Figura 19 ilustra o mesmo. Os campos *Source Port* e *Destination Port* contém os números das portas fonte e destino, respectivamente do protocolo UDP. A porta fonte é opcional, quando usada ela especifica a porta na qual uma resposta poderá ser enviada, se não é usada deve ser preenchida com zeros.

O campo UDP *Length* informa o tamanho, em *octetos*, do segmento UDP, incluindo o próprio cabeçalho. Já o campo *Checksum* apresenta a mesma função de *checksum* do TCP e é utilizado para conferência da integridade dos dados recebidos. Seu cálculo é feito de forma similar, ou seja, utilizando-se o *pseudocabeçalho* montado de acordo com a descrição contida em [Pos81c]. Este campo também é opcional e deve ser preenchido com zeros no caso de uma não utilização.

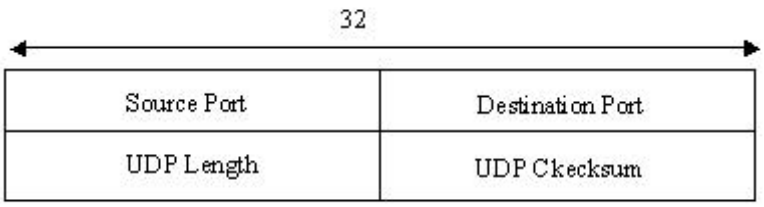

Figura 19: Cabeçalho do UDP

3.9 Considerações Finais

Neste capítulo foi descrito o funcionamento do protocolo TCP. Destacou-se o formato do pacote TCP, as fases do gerenciamento de uma conexão, os algoritmos utilizados durante o

congestionamento que são: *Slow Start*, *Congestion Avoidance*, *Fast Retransmit*, *Fast Recovery*. Foram mostrados os algoritmos RED e RIO que são utilizados no gerenciamento ativo de filas nos roteadores.

Apresentou se as principais características do protocolo UDP e o cabeçalho do segmento.

No próximo capítulo será estudado o funcionamento do protocolo SCTP. Mostrar-se á as principais característica que justificam a superioridade do protocolo SCTP frente ao TCP.

Capítulo 4.0
O Protocolo SCTP

Este capítulo descreve o funcionamento do protocolo de transporte SCTP.

4.1 Introdução

O SCTP [Ste00] foi definido pela IETF no grupo *Signalling Transport* (SIGTRAN) formado em 1999. É um protocolo de transporte orientado a conexão, que transporta datagramas e opera no topo de uma rede não confiável como o IP. Entretanto, o protocolo SCTP foi projetado originalmente para transportar mensagens de sinalização PSTN (*Public Switched Telefonic Network*) sobre redes IP. Resumidamente, apresentam-se algumas das características do protocolo SCTP:

- O SCTP permite empacotamento de múltiplas mensagens do usuário dentro de um único pacote SCTP;
- O nível de rede é tolerante a falhas através do suporte *multi-homing* (múltiplos endereços IP) em um ou em ambos pontos de uma associação. Com isso, há possibilidade de se manter uma sessão mesmo com falhas na rede; e
- Entrega seqüencial de mensagens do usuário dentro de múltiplos *streams*. O termo *stream* é usado no SCTP para referenciar a seqüência de mensagens do usuário que são entregues para a camada superior do protocolo em ordem. No protocolo TCP o termo *stream* refere-se a uma seqüência de *bytes* [Ste00] com a opção de entrega fora de ordem das mensagens individuais do usuário.

4.2 Arquitetura do SCTP

O protocolo SCTP transmite dados através de uma associação, termo semelhante a conexão no TCP que será visto com detalhes na Seção 4.5.1. A Figura 20 [Ste00] ilustra uma associação no SCTP. O protocolo SCTP localiza-se entre a camada de rede IP e a camada de aplicação do usuário SCTP. Observa-se que o serviço do SCTP está na mesma camada que o serviço TCP e UDP. O “Nó A” e o “Nó B” possuem um ou mais endereços IP que serão utilizados em combinação com uma porta na associação SCTP.

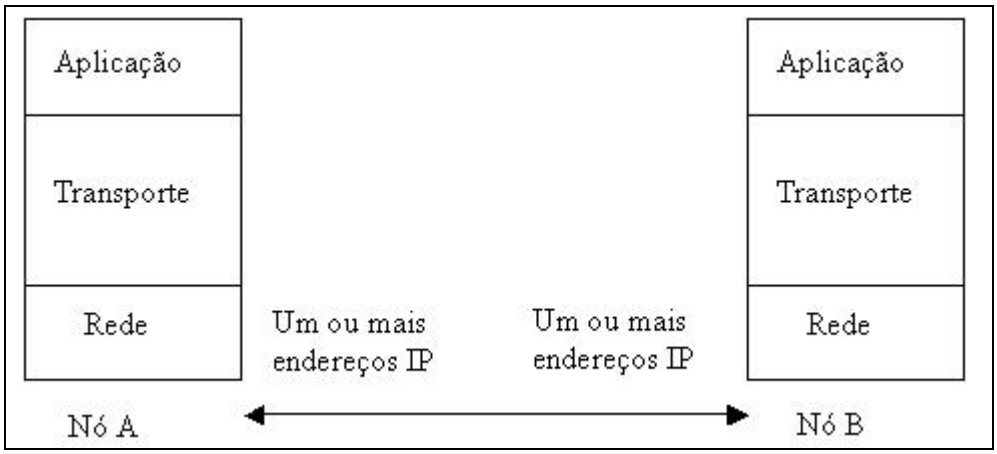

Figura 20:Uma associação SCTP

4.3 Formato do Segmento SCTP

Um segmento no protocolo SCTP é composto de um cabeçalho comum e *chunks* (blocos). Um bloco contém outras informações de controle e dados do usuário. A Figura 21 [Ste00] ilustra o formato do segmento SCTP.

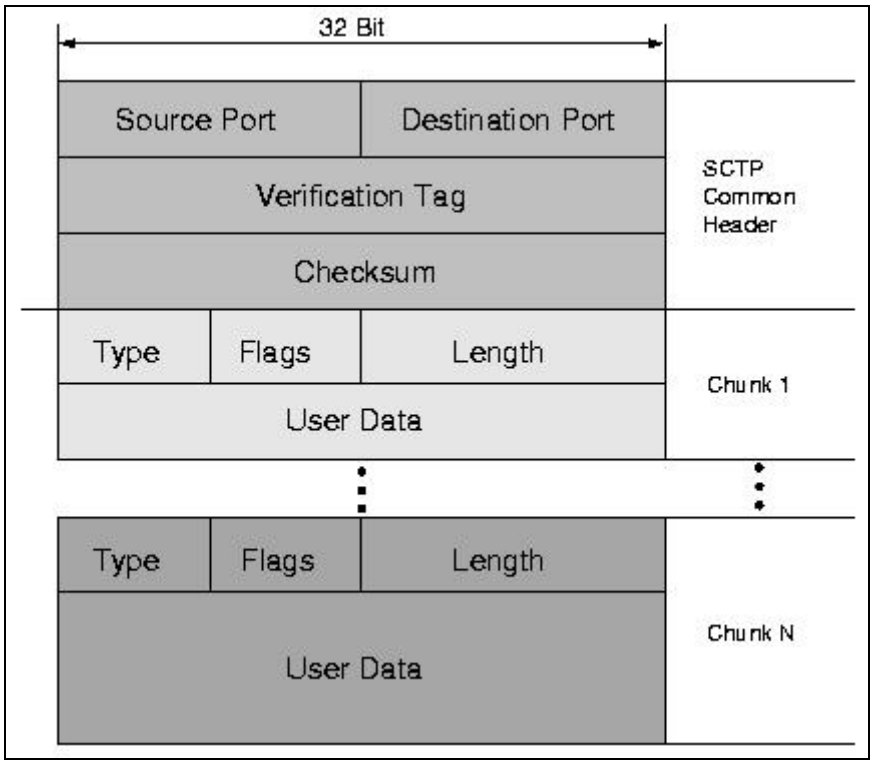

Figura 21 : Formato do segmento SCTP com vários *chunks*

4.3.1 O Cabeçalho do Segmento SCTP

De acordo com a Figura 21, o cabeçalho do SCTP é formado pelos seguintes campos: *Source Port*, *Destination Port* , *Verification Tag* e *Checksum* . O campo *Source Port* (16 *bits*) indica o número da porta do SCTP remetente e pode ser usado pelo receptor em combinação com o endereço IP da fonte. A porta de destino do SCTP e o endereço IP identificam a associação a que este pacote pertence.O campo *Destination Port* (16 *bits*) indica o número da porta SCTP, a qual o pacote será enviado.

O campo *Verification Tag* (32 *bits*), é usado para validar o remetente do pacote O campo *Ckecksum* (32 *bits*) é utilizado na detecção de erros. O *Ckecksum* do pacote SCTP utiliza o algoritmo *CRC*-32 [Sto00] que é mais robusto que o *Checksum* de 16 *bits* do TCP ou UDP.

4.3.2 Campos do *Chunk*

A Figura 21 ilustra o formato dos *chunk* (blocos) que podem ser transmitidos no pacote SCTP. O campo *Type* (8 *bits*) indica o tipo de informação contida no *chunk* . O uso do campo *Flag* (8 *bits*) depende do tipo que é especificado no campo *Type*. O campo *Length* (16 *bits*) informa o tamanho em *bytes* incluindo os campos *Type Flags*, *Length* e *Chunk Value*.

Como ilustrado na Figura 21 [Ste00], são N *Chunks* dentro de uma única associação SCTP. O número N é determinado pelo número máximo de transmissões (MTU), ou seja, o tamanho de um caminho de transmissão. O protocolo SCTP permite que *chunks* sejam empacotados com exceção do *INIT chunk* (início da associação) e do *INIT ACK* (início de reconhecimento). Existem quatorze tipos de *chunks*: um *chunk* de dados e treze c*hunks* de controle.

4.4 Descrição do *Chunks* SCTP

Mostramos nesta seção o formato dos diferentes tipos de *chunks* do SCTP.

4.4.1 Dados (*DATA*) (*Type* = 0)

A Figura 22 [Ste00] ilustra o formato do *DATA chunk* seguido da descrição dos seus campos.

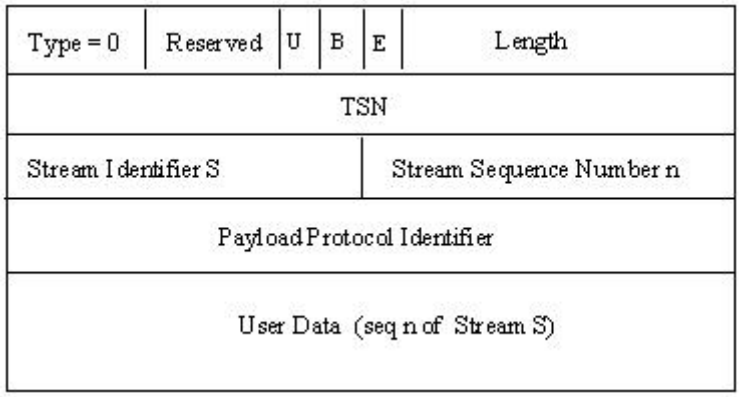

Figura 22: Formato do *DATA Chunk*

O campo *Reserved* (5 *bits*) deve estar com todos os *bits* preenchidos com 0 e ser ignorado pelo receptor. Os três *bits* que são utilizados nos blocos de dados são:

- U (1 *bit*) – Com o valor 1, indica que os *DATA chunk* estão desordenados e não há número de seqüência do *stream* associado a este *DATA chunk*;
- B (1 *bit*) – Com o valor 1, indica o primeiro fragmento da mensagem do usuário; e
- E (1 *bit*) – Indica o último fragmento da mensagem do usuário caso o valor seja 1.

O campo *Length* (16 *bits*) indica o tamanho do *DATA chunk* em *bytes*. O campo TSN (32 *bits*) representa o número de seqüência de transmissão do *DATA chunk*. O campo *Stream Identifier* S (16 *bits*) identifica o fluxo ao qual a mensagem do usuário está associado. O campo *Stream Sequence Number* (16 *bits*) indica o número de seqüência do fluxo dos dados do usuário dentro do fluxo S. Quando a mensagem do usuário é fragmentada pelo protocolo SCTP, o mesmo número de seqüência do fluxo deve ser carregado em cada fragmento da mensagem.

O campo *Payload Protocol Identifier* (32 *bits*) representa um identificador especificado do protocolo da aplicação. O campo *User Data* (tamanho variável) indica os dados do usuário que pertencem ao pacote.

4.4.2 Início (*INIT*) (*Type* = 1)

O *INIT chunk* é usado para iniciar uma associação SCTP entre dois pontos. O formato do *INIT chunk* é ilustrado na Figura 23 [Ste00].

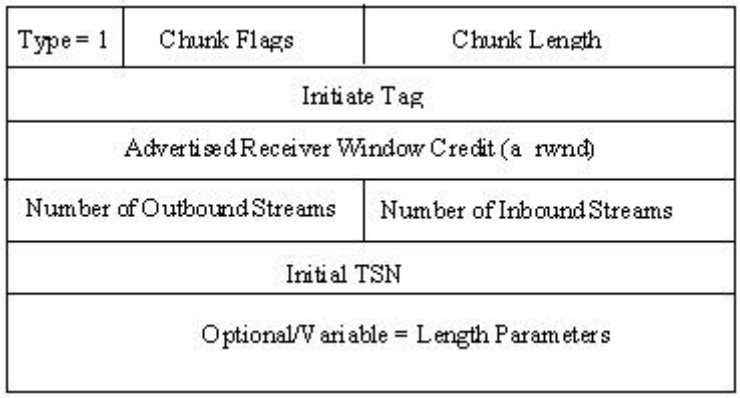

Figura 23: Formato do *INIT chunk*

O campo *Chunk Flags* (8 *bits*) é reservado e todos os *bits* devem ser ajustados para 0 pelo remetente e ignorado pelo receptor. O campo *Chunk Length* (16 *bits*) representa o tamanho total do *INIT chunk*.

O campo *Initiate tag* (32 *bits*) possui um valor que é gravado pelo receptor do *INIT*. Este valor deve ser colocado no campo *Verification Tag* do cabeçalho do pacote SCTP que o receptor do *INIT* transmitir dentro desta associação. Se o valor recebido do *INIT chunk* for zero, o receptor deve tratá-lo como um erro e fechar a associação pela transmissão de um *ABORT*.

O campo *Advertised Receiver Window Credit* (32 *bits*) indica o valor do tamanho do espaço em *buffer* (em *bytes*), que o remetente do INIT reserva na associação.

O campo *Number of Outbound Streams* (16 *bits*) define o número máximo de fluxos de saída do remetente do *INIT chunk* solicitado na criação da associação. Se o receptor do *INIT* receber o valor zero nesse campo, a associação deverá ser abortada.

O campo *Number of Inbound Streams* (16 *bits*) define o número máximo de fluxos de entrada do remetente do *INIT chunk* permitido pelo ponto final quando da criação da associação. Se o receptor do *INIT* receber o valor de campo igual a zero, a associação deverá ser abortada.

O campo *Initial TSN* (32 *bits*) define o número de seqüência inicial que o remetente usa.

No campo *Optional/Variable Length Parameters*, os seguintes parâmetros são opcionais no *INIT*: IP*V4 Address Parameter* (5), *IPV6 Adress Parameter* (6), *Cookie Preservative* (9), *Host Name Address* (11), *Support Address Types* (12), maiores detalhes ver [Ste00].

4.4.3 Reconhecimento do Início (*INIT ACK*) (*Type* = 2)

O *INIT ACK chunk* é usado para reconhecer a iniciação de uma associação no SCTP. É similar ao *INIT chunk*, só que ele usa dois parâmetros adicionais no campo *Optional/Variable*: o *Statie Cookie* e o *Unrecognized Parameter*. A Figura 24 [Ste00] ilustra o formato do *INIT ACK*, e em seguida serão apresentadas as descrições dos campos.

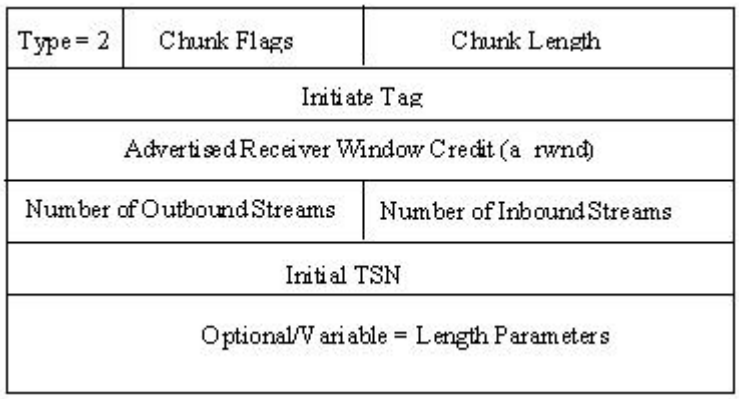

Figura 24: Formato do *INIT ACK Chunk*

O campo *Chunk Flags* (8 *bits*) é reservado a todos os *bits* e deve ser igual a 0 no remetente e ignorado pelo receptor. O campo *Chunk Length* (16 *bits*) representa o tamanho total do *INIT ACK chunk*.

O campo *Initiate Tag* (32 *bits*) possui um valor que é gravado pelo receptor do *INIT ACK chunk*. Este valor deve ser colocado no campo *Verification Tag* de todo pacote SCTP que o receptor do *INIT* transmitir dentro desta associação. Se o valor recebido do *INIT chunk* for zero, o receptor deve tratar como um erro e fechar a associação pela transmissão de um *ABORT*.

O campo *Advertised Receiver Window Credit* (a_rwnd) (32 *bits*) indica o valor do tamanho espaço em *buffer* (em *bytes*) que o remetente do *INIT ACK* reserva na associação.

O campo *Number of Outbound Streams* (16 *bits*) define o número máximo de fluxos de saída do remetente do *INIT chunk* pedido para criar na associação. Se o receptor do *INIT ACK* receber o valor desse campo ajustado para zero, a associação deve ser abortada.

O campo *Number of Inbound Streams* (16 *bits*) define o número máximo de fluxos de entrada do *INIT ACK chunk* permitido pelo ponto final para criar na associação. Se o receptor do *INIT ACK* receber o valor desse campo ajustado para zero a associação deve ser abortada.

O campo *Initial TSN* (32 *bits*) define o número inicial de seqüência de transmissão que o remetente utilizará.

No campo *Optional/Variable Length Parameters* – Os dois parâmetros mais importantes são:

- *State Cookie* (*Parameter Type* = 7) - contém todos os estados necessários e informações de parâmetros requeridas pelo remetente do *INIT ACK* criado na associação, juntamente com o MAC (*Message Authentication Code*). Os estados são: *CLOSED*, *COOKIE ECHOED*, *COOKIE WAIT*, *COOKIE ECHOED, ESTABLISHED*, *SHUTDOWN PENDING*, *SHUTDOWN SENT*, *SHUTDOWN RECEIVED*, *SHUTDOWN ACK SENT*.
- *Unrecognized Parameters* (*Parameter Type* = 8) - é retornado ao remetente do *INIT ACK chunk* quando o *INIT ACK* contém um parâmetro irreconhecível, ou seja, um valor que indique que se deve reportar ao remetente.

4.4.4 Reconhecimento Seletivo (*SACK*) (*Type* = 3)

O *SACK chunk* é enviado para o par *endpoint* a fim de reconhecer o recebimento de dados e também para informar ao *endpoint* lacunas no recebimento de dados subseqüentes, como representado pelos seus TSNs. A Figura 25 [Ste00] ilustra o *SACK chunk*.

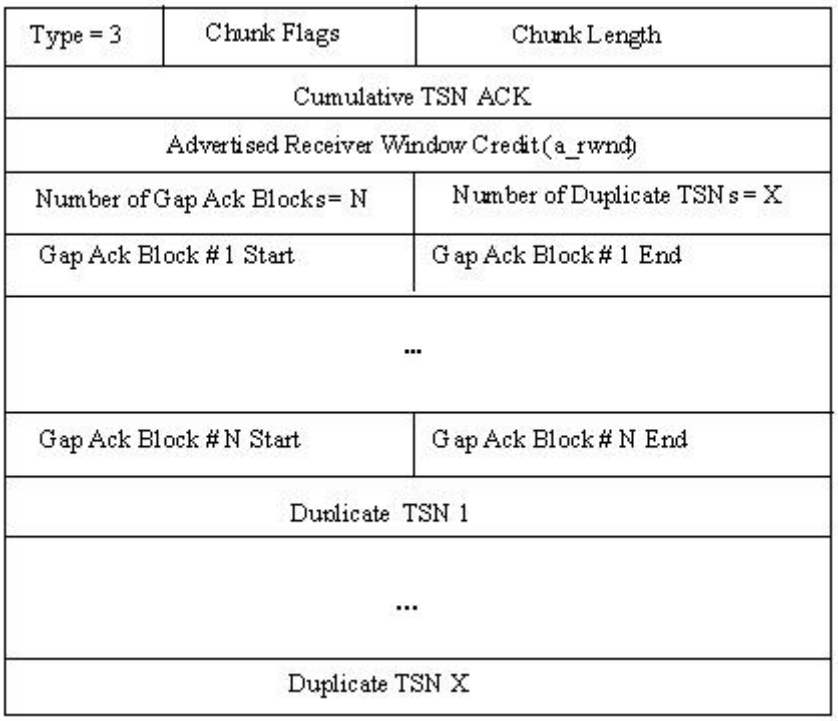

Figura 25: Formato do *SACK chunk*

O campo *Chunks Flags* (8 *bits*) contém todos os *bits* com o valor zero na transmissão e será ignorado na recepção.

O campo *Cumulative TSN Ack* (32 *bits*) contém o número de seqüência de transmissão do último *DATA chunk* recebido, na seqüência antes da perda de algum segmento de dados.

O campo *Advertised Receiver Window Credit* (32 *bits*) indica a atualização do espaço em *buffer*, em *bytes*, do remetente do *SACK*.

O campo *Number of Gap Ack Blocks* (16 *bits*) indica o número de segmento de dados incluído no *SACK* que chegaram com "buracos".

O campo *Number of Duplicate TSNs* (16 *bits*) contém o número de TSN duplicados que o *endpoint* recebeu. Cada TSN duplicado é listado seguindo a lista do *Gap Ack Block*.

O campo *Gap Ack Blocks Start* (16 *bits*) indica o início do *offset* TSN para este *Gap Ack Block*. Já o campo *Gap Ack Block End* (16 *bits*) indica o final do *offset* TSN para este *Gap Ack Block*.

O campo *Duplicate TSN* (32 *bits*) informa o número de vezes que um TSN foi recebido e duplicado desde o último *SACK* que foi enviado.

4.4.5 Pedido do *Heartbeat* (*HEARTBEAT*) (*Type* = 4)

Um *endpoint* deve enviar o *Heartbeat Request chunk* para testar a conectividade de um endereço destino específico dentro de uma associação. A Figura 26 [Ste00] ilustra o *Heartbeat Request chunk*.

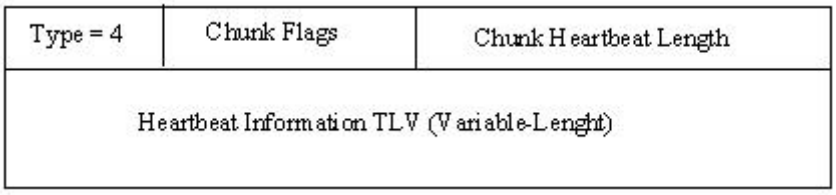

Figura 26: Formato do *HEARTBEAT REQUEST chunk*

O campo *Chunk Flags* (8 *bits*) possui valor zero na transmissão sendo ignorado na recepção. O campo *Chunk Heartbeat Length* (16 *bits*) indica o tamanho do *chun*k em *bytes*.

O campo *Heartbeat Information* TLV (tamanho variável) utiliza o parâmetro *Hearbeat Info* ilustrado na Figura 27 [Ste00]. O campo *Sender-specific Heartbeat Info* possui o tempo atual de envio do pacote e o endereço para o qual o *HEARBEAT chunk* foi enviado.

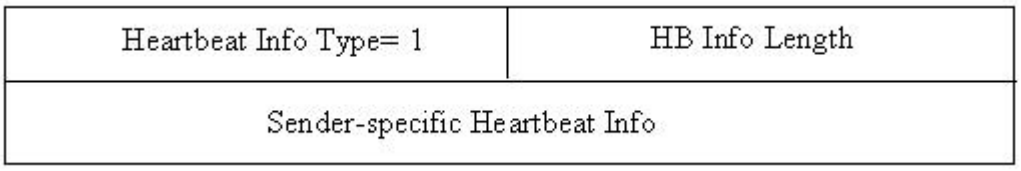

Figura 27: Formato do parâmetro *Heartbeat Info*

4.4.6 Reconhecimento do *Heartbeat* (*HEARTBEAT ACK*) (*Type* = 5)

Um *endpoint* deve enviar este *chunk* para o par *endpoint* como resposta ao *HEARTBEAT chunk*. Um *HEARTBEAT ACK* é sempre enviado para o endereço IP de origem que originou o *HEARTBEAT chunk*. A Figura 28 [Ste00] ilustra o *HEARTBEAT ACK chunk*, e a seguir a descrição dos campos.

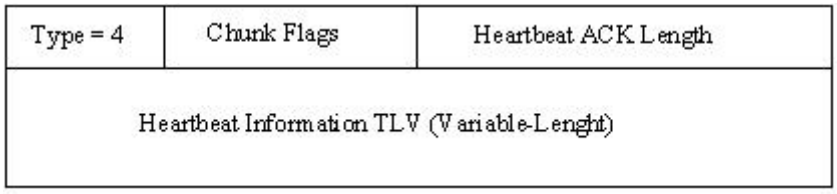

Figura 28: Formato do *HEARTBEAT ACK*

O campo *Chunk Flags* (8 *bits*) possui valor zero na transmissão sendo ignorado na recepção. Já o campo *Hearbeat ACK Length* (16 *bits*) indica o tamanho do *chunk,* em *bytes*.

O campo *Hearbeat Information* TLV (tamanho variável) contém o parâmetro *Heartbeat Request* para o qual *Heartbeat Acknowledgement* deverá retornar.

4.4.7 Abortar uma Associação (*ABORT*) (*Type* = 6)

O *ABORT chunk* é enviado pelo *endpoint* de uma associação para fechar uma associação. O *ABORT chunk* contém os parâmetros da causa do erro e são usados para informar o receptor o motivo do *abort*. A Figura 29 [Ste00] ilustra o *ABORT chunk*, e em seguida tem-se a descrição dos campos.

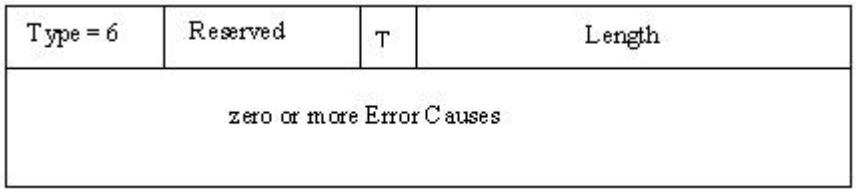

Figura 29 : Formato do *ABORT chunk*

O campo *Reserved* (7 *bits*) possui o valor zero na transmissão, sendo ignorado na recepção.

- **T** (1 bit) – O T *bit* é ajustado para zero pelo remetente se tiver um TCB (*Transmission Control Block*) a ser destruído, caso contrário o *bit* deve ser um. Maiores detalhes sobre o TCB na Seção 4.5.1.

O campo *Lenght* (16 *bits*) indica o tamanho do *chunk* em *bytes*. Na Seção 4.4.10 serão ilustradas as definições das causas de erro.

4.4.8 Início do *Shutdown* (*SHUTDOWN*) (*Type* = 7)

Um nó em uma associação deve usar este *chunk* para iniciar o fechamento da associação com o par. O *SHUTDOWN chunk* é ilustrado de acordo com a Figura 30 [Ste00] em seguida encontra-se a descrição do campos.

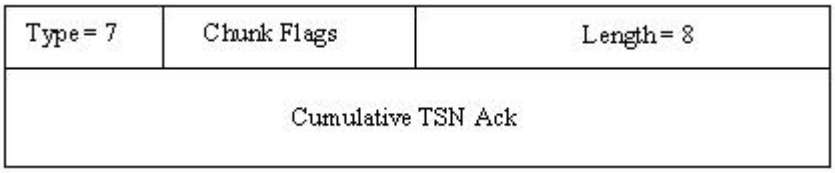

Figura 30: Formato do *Shutdown chunk*

O campo *Chunk Flags* (8 *bits*) possui o valor zero na transmissão, sendo ignorado na recepção. O campo *Cumulative TSN Ack* (32 *bits*) contém o TSN do último *chunk* recebido na seqüência antes do intervalo.

4.4.9 Reconhecimento do *Shutdown* (*SHUTDOWN ACK*) (*Type* = 8)

O *SHUTDOWN ACK chunk* deve ser usado no reconhecimento da recepção de um *SHUTDOWN chunk* com a finalidade de completar o processo de *shutdown*. A Figura 31 [Ste00] ilustra o formato do *SHUTDOWN ACK chunk*.

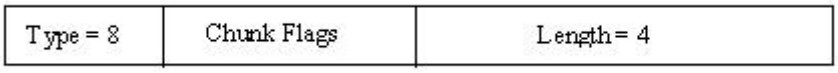

Figura 31: Formato do *SHUTDOWN ACK chunk*

Chunk Flags (8 *bits*) – Possui o valor zero na transmissão, sendo ignorado na recepção.

4.4.10 Operação de Erro (*ERROR*) (*Type* = 9)

Um *endpoint* envia o *ERROR chunk* para o par *endpoint* para notificar sobre as condições de erro. O *ERROR chunk* pode conter uma ou mais causas de erros. A Figura 32 [Ste00] ilustra o formato do *ERROR chunk*.

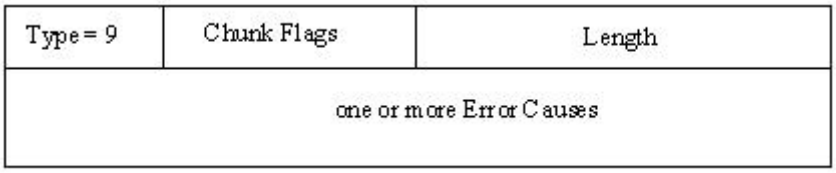

Figura 32: Formato do *ERROR chunk*

O campo *Chunk Flags* (8 *bits*) possui o valor zero na transmissão, sendo ignorado na recepção. Já o campo *Lenght* (16 *bits*) indica o tamanho do *chunk,* em *bytes*, incluindo o *chunk header* e todas as causas de erro presente no *ERROR chunk*.

As causas de erro são mostradas na Tabela 6 e para maiores detalhes ver em [Ste00] [Ste02].

Cause Code Value	Cause Code
1	Invalid Stream Identifier
2	Missing Mandatory Parameter
3	Stale Cookie Error
4	Out of Resource
5	Unresolvable Address
6	Unrecognized Chunk Type
7	Invalid Mandatory Parameter
8	Unrecognized Parameters
9	No User Data
10	Cookie Received While Shutting Down
11	Restart of an with new Address
12	User Initiated Abort

Tabela 6: Causas de erro presentes no *ERROR chunk*

4.4.11 *Cookie Echo* (*COOKIE ECHO*) (*Type* = 10)

O *COOKIE ECHO chunk* é usado somente durante o estabelecimento de uma associação. Ele é enviado pelo iniciador da associação para seu par com a finalidade de completar o processo de iniciação. A Figura 33 [Ste00] ilustra o *COOKIE ECHO chunk*, e a seguir tem-se a descrição dos campos.

Figura 33: Formato do *COOKIE ECHO chunk*

O campo *Chunk Flags* (8 *bits*) possui o valor zero na transmissão, sendo ignorado na recepção. Já o campo *Lenght* (16 *bits*) indica o tamanho do *chunk* em *bytes*, e inclui os quatro *bytes* do *chunk header* e também o tamanho do *cookie*.

O campo *Cookie* (tamanho variável) possui o *cookie* recebido no parâmetro *State Cookie* do *INIT ACK* anterior.

4.4.12 Reconhecimento do *Cookie* (*COOKIE ACK*) (*Type* = 11)

O *COOKIE ACK* é usado durante uma associação e sua função é reconhecer o recebimento de um *COOKIE ECHO chunk*. A Figura 34 [Ste00] ilustra o *COOKIE ACK chunk*.

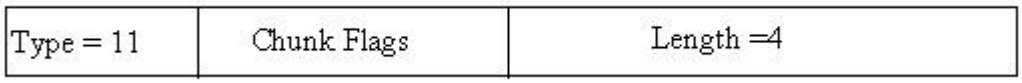

Figura 34: Formato do *Cookie ACK*

O campo *Chunk Flags* (8 *bits*) é ajustado para zero na transmissão, sendo ignorado na recepção.

4.4.13 Finalização do *Shutdown* (*SHUTDOWN COMPLETE*) (*Type* = 14)

O *SHUTDOWN COMPLETE chunk* deve ser usado para o reconhecimento do recibo de um *SHUTDOWN ACK* e para finalizar o processo de *shutdown*. A Figura 35 [Ste00] ilustra o formato do *SHUTDOWN COMPLETE chunk*.

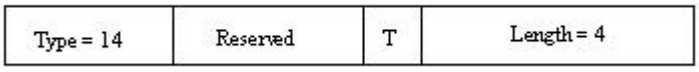

Figura 35: Formato do *SHUTDOWN COMPLETE chunk*

O campo *Reserved* (7 *bits*) é configurado para zero na transmissão, sendo ignorado na recepção.

T (1 *bit*) possuí o valor zero pelo remetente se tiver um TCB a ser destruído, caso contrário o *bit* deve possuir o valor um.

4.5 Gerenciamento de uma associação

Nesta seção mostramos as três fases de uma associação SCTP: iniciação da associação, transmissão de dados e término da associação.

4.5.1 Iniciação de uma associação

O processo de iniciação de uma associação no SCTP utiliza o mecanismo *Four- Way-Handshake* ilustrado na Figura 36 [Str02]. Ele significa literalmente um "aperto de mão em quatro vias", onde são necessários quatros segmentos para que a associação entre os *endpoints* seja estabelecida.

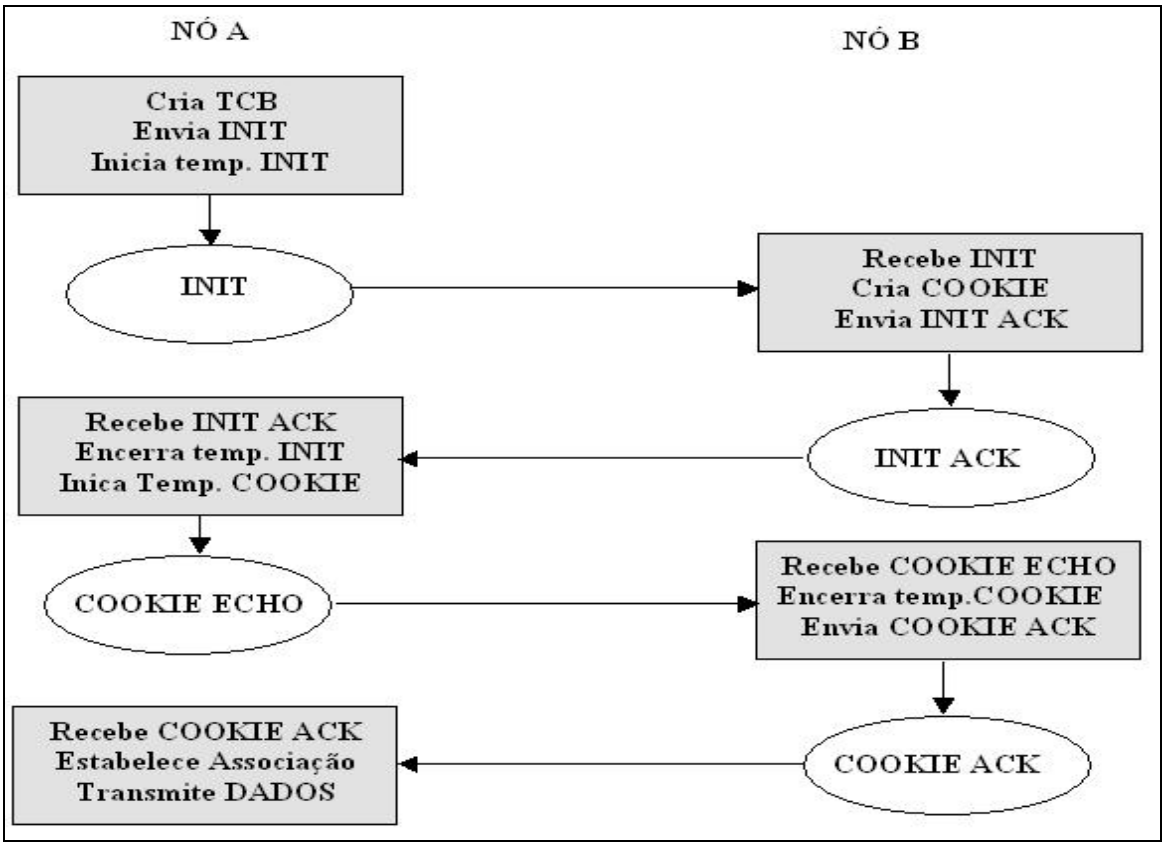

Figura 36:*For-way-Handshake* na iniciação de uma associação

O estabelecimento de uma associação que utiliza quatro segmentos funciona da seguinte maneira:

1. Inicialmente estamos nos estado *CLOSED* (indica que associação não foi criada). Na inicialização de uma associação acontece a criação do TCB (*Transmission Control Block*). O TCB é uma estrutura de dados interna criada pelo SCTP *endpoint* que contém todas as informações operacionais para o *endpoint* manter e gerenciar a associação correspondente. O "NÓ A" gera um *INIT chunk* e envia para o "NÓ B" o temporizador do *INIT* é iniciado e entra no estado *COOKIE-WAIT*. O "NÓ A" também fornece seu *Verification Tag* no campo *Initiate Tag*".

2. O "NÓ B" recebe o *INIT chunk* e responde com um *INIT ACK chunk*. O "NÓ B" deve configurar o *Verification Tag* para (Tag_Nó A) e fornecer também seu próprio *Verification Tag* (Tag_Nó B) no campo *Initiate Tag*. É gerada também uma chave secreta (por exemplo, com o algoritmo SHA-1 (*Secure Hash Algorithm*)) para maiores detalhes ver [Eas01]. Este valor da chave secreta é colocado no parâmetro *COOKIE* juntamente com o MAC (código de autenticação da mensagem) que será entregue ao "NÓ A" juntamente com o *INIT ACK chunk*.

3. No recebimento do *INIT ACK chunk do* "NÓ B", o "NÓ A" então parar o temporizador do *INIT* e deixar o estado *COOKIE-WAIT*. O "NÓ A" envia o parâmetro *COOKIE* que foi aceito do *INIT ACK chunk* no *COOKIE ECHO chunk*, nesse momento é iniciado o temporizador do *COOKIE* e o estado passa a ser o *COOKIE-ECHOED*.

4. Na recepção do *COOKIE ECHO chunk* pelo "NÓ B" o temporizador do *COOKIE ECHO* é paralisado sendo enviado para o "NÓ A" o *COOKIE ACK chunk*. O *COOKIE ECHO chunk* contêm a estrutura de dados do *COOKIE* como parâmetro, o "NÓ B", então "desempacota" os dados contidos neste *COOKIE* e utiliza novamente o MAC contido nele para verificar se ele foi o originador do *COOKIE*. Caso o valor do MAC esteja OK, o *COOKIE* será considerado válido (o "NÓ B" realmente o criou) e os dados contidos no *COOKIE* serão usados para iniciar a associação SCTP.

5. O "NÓ A" recebe o *COOKIE ACK* e muda do estado *COOKIE-ECHOED* para o estado de estabelecimento e está pronto para enviar e receber dados

É importante lembrar que não há alocação de recurso até que o terceiro segmento seja recebido e validado. Há utilização deste mecanismo é uma forma de garantir que o pedido de estabelecimento de uma associação realmente tenha sido originado no par *endpoint* correto. O protocolo TCP não disponibiliza deste mecanismo.

4.5.2 Transmissão de Dados no SCTP

No processo de transmissão de dados entre o "NÓ A" e o "NÓ B", são trocados *HEARTBEAT* e *HEARTBEAT ACK chunks* entre eles, em intervalos regulares de tempo, controlado pelo *HEARTBEAT TIMER*. Estes *chunks* testam a conectividade dos *endpoints*. A Figura 37 [Str02] ilustra a transmissão de dados no protocolo SCTP.

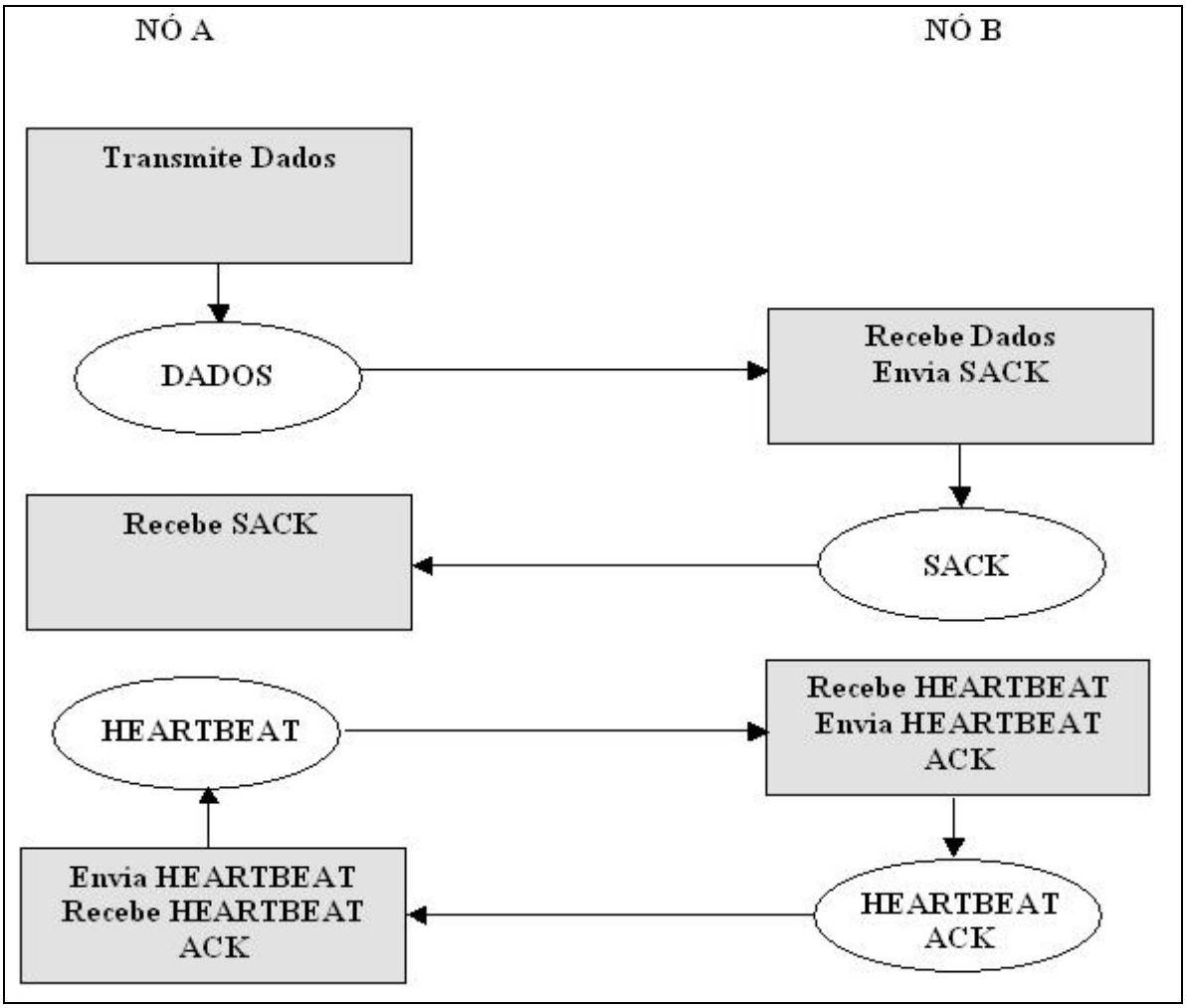

Figura 37:Transmissão de dados no SCTP

A transmissão de dados no protocolo SCTP ocorre como descrito a seguir.

1. O "NÓ A" transmite dados para o "NÓ B", de acordo com a janela *CWND*. O "NÓ B" utiliza janela *RWND* que informa a capacidade de recepção dos dados.
2. Depois que cada dado é recebido, o "NÓ B" receptor retorna um *SACK chunk* para o "NÓ A" para informar que recebeu o *DATA chunk*.
3. O "NÓ A" envia um *HEARTBEAT chunk* para o "NÓ B" para testar a conectividade. O "NÓ B" responde através do envio de um *HEARTBEAT ACK chunk* para o "NÓ A".

O processo de transmissão dos dados ao "NÓ B" ocorre até o momento que um *endpoint* decidir finalizar a associação pelo envio de um *SHUTDOWN chunk* ou *ABORT chunk*.

4.5.3 Término da Associação no SCTP

Uma associação no protocolo SCTP pode ser finalizada pelo *Abort chunk* ou *Shutdown chunk*. Um *Abort* de uma associação indica que todos os dados pendentes foram rejeitados e não entregues ao par *endpoint*. Um *shutdown* de uma associação é considerado um *graceful close* onde todos os dados na fila de um *endpoint* foram entregues aos respectivos par.

O protocolo SCTP não suporta o estado *half-open* (como o protocolo TCP) onde um lado pode continuar emitindo dados enquanto a outra extremidade está fechada. Quando outro *endpoint* executa um *shutdown*, a associação em cada *endpoint* para de receber novos dados do usuário e somente entrega os dados que estão na fila, ao mesmo tempo em que envia ou recebe o *SHUTDOWN chunk*. A Figura 38 [Str02] ilustra o término de uma associação no SCTP.

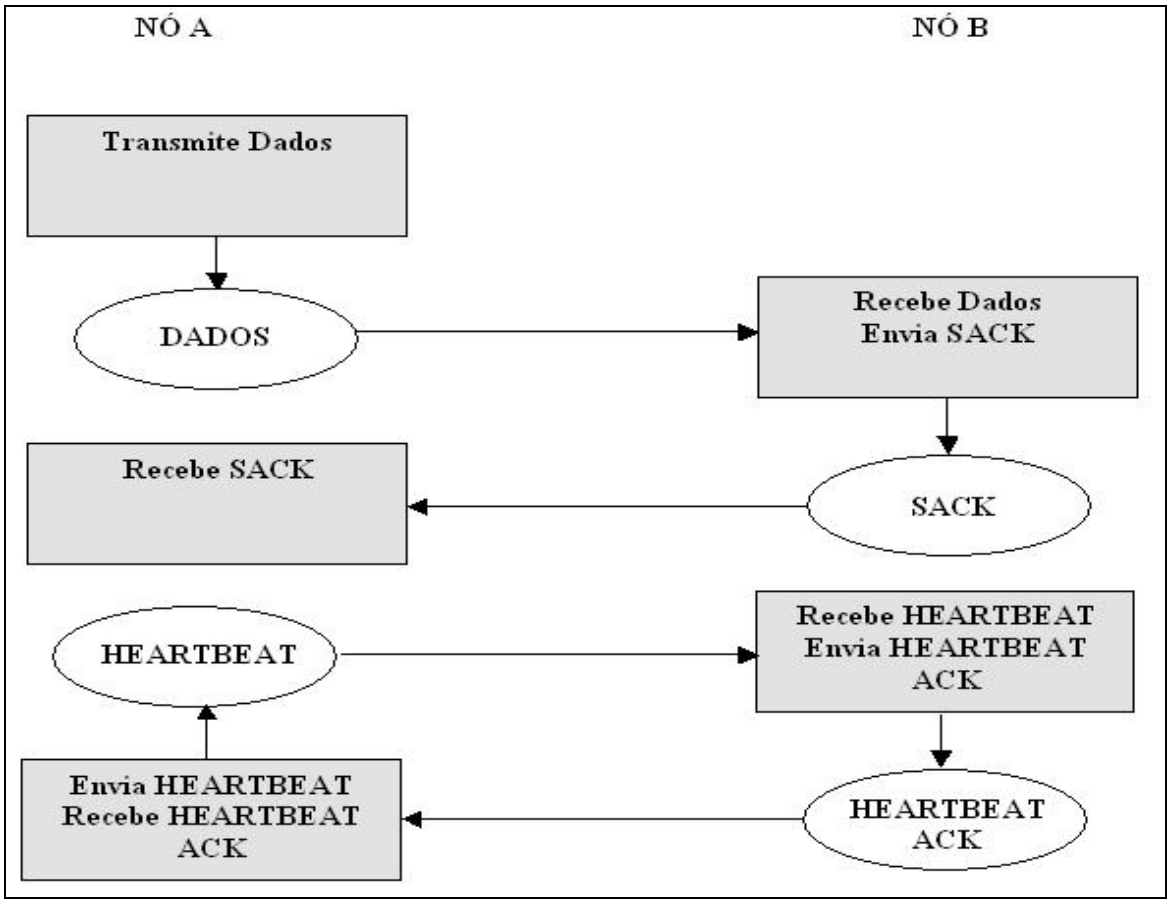

Figura 38: Término de uma associação no SCTP

O processo de finalização de uma associação no protocolo SCTP funciona da seguinte maneira.

1. O "NÓ A" envia um *SHUTDOWN chunk para o* "Nó B" e inicia o temporizador do *SHUTDOWN*. A partir desse momento ele entra no estado *SHUTDOWN-SENT* (nesse estado o *endpoint* deve receber e reconhecer dados).
2. O "NÓ B" reconhece o recibo de um *SHUTDOWN chunk* através do envio de um *SHUTDOWN ACK chunk* para o "NÓ A". Nesse momento ele entra no estado *SHUTDOWN-RECEIVED* (nesse estado para de aceitar novos dados, mas pode transmitir os dados que estiverem na fila).
3. O "NÓ A" recebe o *SHUTDOWN ACK chunk*, pára o *SHUTDOWN TIMER*, deleta o TCB e entra no estado *SHUTDOWN-ACK-SENT* (não pode enviar dados). O "NÓ A" gera um *SHUTDOWN COMPLETE chunk* e envia este *chunk* para o "NÓ B". Quando o "NÓ B" recebe o *SHUTDOWN COMPLETE chunk* a associação é finalizada.

4.6 Características do Protocolo SCTP

Nesta seção mostramos de forma detalhada as principais características do protocolo SCTP que são: SCTP *Multi-Homing*, *Multi-Streaming*, fragmentação, remontagem, empacotamento dentro de uma associação, serviço com rígida ordem de entrega das mensagens e o serviço de entrega de mensagens fora de ordem.

4.6.1 SCTP *Multi-Homing*

Um SCTP *endpoint* é considerado *Multi-Homing* por suportar múltiplos endereços IP. O *Multi-Homing* permite ao protocolo SCTP um certo nível de tolerância a falhas, caso uma das interfaces IP falhe. O *endpoint* pode usar a outra interface para receber ou enviar dados. Para suportar *Multi-Homing*, o SCTP *endpoint* troca uma lista de endereços durante a iniciação de uma associação. Cada *endpoint* deve ser capaz de receber mensagens de alguns dos endereços associados com o *endpoint* remoto. Uma única porta é utilizada pela lista de endereços em uma sessão específica [Ong02]. A Figura 39 [Rav01] ilustra um *endpoint Multi-Homing* .

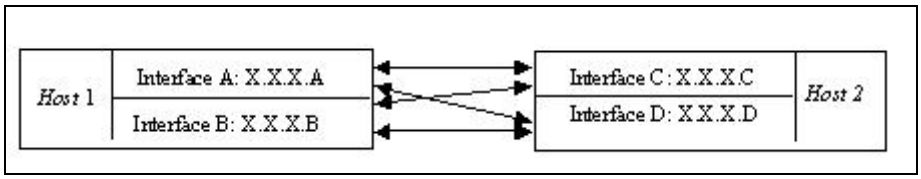

Figura 39 : *Endpoint Multi-Homing*

Um *endpoint Multi-homing* adiciona as seguintes funções básicas ao protocolo SCTP:

1. Um *endpoint* deve selecionar um dos múltiplos endereços destino de um *endpoint Multi-Homing* como o caminho principal. Por definição, um *endpoint* deve sempre transmitir pelo caminho principal, exceto se o usuário SCTP especificar explicitamente o endereço de transporte destino (e possivelmente o endereço de transporte de origem) para ser utilizado.

2. O SCTP *endpoint* deve sempre transmitir *chunks* (isto é, *SACK*, *HEARBEAT ACK*) para o mesmo endereço de transporte destino, para o qual recebeu o *chunk* de dados ou *chunk* de controle ao qual estabeleceu a associação.
3. Na retransmissão de *DATA chunk*, o *endpoint* deve considerar cada par de endereços origem-destino em sua política de seleção de retransmissão. O *endpoint*, então, dever tentar escolher o par origem-destino que seja o mais diferente possível do par original origem-destino ao qual o pacote foi transmitido, até que o *primary path* torne-se disponível novamente.
4. Na retransmissão de *DATA chunk* um *endpoint Multi-Homing*, deve retransmitir um c*hunk* a um endereço de transporte destino ativo que seja diferente do último endereço de destino a que o *DATA chunk* foi enviado.

É importante ressaltar que o *Multi-Homing* no SCTP permite o tráfego de dados em diferentes caminhos usando endereços IP distintos. Perdas de sessão devido a falhas na rede física é na prática inexistente, então conclui-se que há maior disponibilidade da associação com múltiplos endereços IP do que só com *endpoint Single-Homing*. Com isso, observa-se que as associações do SCTP são bastante tolerantes a falhas.

O protocolo TCP não dispõe dessa característica sendo vulnerável a essas falhas, com isso, pode acontecer o isolamento de pontos finais ou da rede como um todo até que os protocolos de roteamento da Internet contornem o problema.

4.6.2 *Multi-Streaming* na Associação

O *Multi-Streaming* no protocolo SCTP permite separar e transmitir os dados do usuário em múltiplos fluxos, estes fluxos são independentes. Vale a pena salientar que a mensagem que for perdida somente afetará a transmissão apenas do fluxo a que ela está associada, isto que dizer que os outros fluxos dentro da associação não serão afetados [Cam02a].

O *Multi-Streaming* permite o protocolo SCTP eliminar o bloqueio desnecessário que freqüentemente ocorre nas transmissões que utilizam o protocolo de transporte TCP. Como foi explicado anteriormente, o fluxo do TCP é uma seqüência de *bytes* que mantém uma rígida ordem de entrega que será visto na seção 4.6.3, em maiores detalhes. Essa entrega seqüenciada resulta na desvantagem conhecida como "*head-of-the-queue blocking*", onde as

mensagens dentro do fluxo não podem ser entregues antes das outras. No protocolo SCTP as mensagens de alta prioridade podem ser entregues antes das mensagens de menor importância.

4.6.3 Serviço com Rígida Ordem de Entrega das Mensagens

No serviço de entrega ordenada das mensagens o receptor recebe as mensagens na mesma ordem em que foram enviadas pelo transmissor. Para isso acontecer dentro do *stream*, o *endpoint* deve entregar os dados com a *flag* U configurado para o valor zero, de acordo com a ordem de seu SSN, isto é, o SSN é usado para o receptor determinar a ordem de entrega das mensagens.

4.6.4 Serviço de Entrega de Mensagens Fora de Ordem

No serviço de entrega de mensagens fora de ordem, o receptor recebe as mensagens sem levar em consideração a ordem em que foram enviadas. É importante ressaltar que esse tipo de serviço somente é utilizado nos *DATA chunks*, para isso a *flag* U é configurada para o valor um. Como os fluxos são independentes um dos outros, então, os dados dentro de um fluxo podem ser entregues em ordem, enquanto outros dados, no mesmo fluxo podem ser entregues fora de ordem. Esse serviço pode ser utilizado por mensagens importantes que podem ser entregues antes das outras. (para exemplificar temos as mensagens que podem abortar uma aplicação). Esta é uma virtude do protocolo SCTP que não pode ser implementada pelo protocolo TCP.

4.6.5 Fragmentação e Remontagem

Na fragmentação o *endpoint* transmissor deve fragmentar (dividir) as mensagens do usuário em uma série de *DATA chunks*, se o tamanho do pacote SCTP não for superior ao *MTU* (*Maximum Transmission Unit*), em caso contrário, será enviado um erro para a ULP (*Upper Layer Protocol*) e a mensagem não será enviada. O mecanismo de fragmentação é formado pelos seguintes passos abaixo:

1. Na transmissão o transmissor deve atribuir a seqüência dos *DATA chunks* , um TSN para cada um dos *DATA chunks* enviados, mas é atribuído o mesmo SSN a cada um dos *DATA chunks*.
2. O transmissor configura o *bit* B (indica a primeira fragmentação da mensagem do usuário) e o *bit* E (indica a última fragmentação da mensagem do usuário) da seguinte maneira: o primeiro *DATA chunk* da série possuí os seguintes valores para os *bits* B e E '10', já os *bits* B e E do último *DATA chunk* na série possui o valor '01' e os demais *DATA chunk* possui o valor '00'.
3. O receptor enfileira os *DATA chunks* fragmentados recebidos para a remontagem das mensagens do usuário. Depois que a mensagem é remontada, o protocolo SCTP a entrega ao *stream* específico.

4.6.6 Empacotamento

Para o protocolo SCTP utilizar o mecanismo de empacotamento é necessário que o *endpoint* empacote os *DATA chunks*, incluindo vários *DATA chunks* em um único pacote SCTP, com isso há um crescimento de eficiência na utilização da rede, reduzindo a possibilidade de congestionamento. É importante lembrar que o tamanho total do datagrama IP (formado pelo pacote SCTP e cabeçalho IP) não pode ultrapassar o *Path MTU*.

A técnica de empacotamento utilizada pelo protocolo SCTP permite empacotar *chunk* de controle com *chunk* de dados, com exceção do *SHUTDOWN chunk* ou *SHUTDOWN ACK chunk*. Os *DATA chunks devem* ser inseridos depois dos *chunks* de controle. O *endpoint* não pode empacotar *INIT*, *INIT ACK ou SHUTDOWN COMPLETE* com nenhum dos outros *chunks*. Para determinar o fim de um *chunk* e o início do próximo *chunk* o receptor utiliza o campo *chunk length*.

4.8 Controle de Congestionamento no protocolo SCTP

O protocolo SCTP deve operar no surgimento inesperado de tráfego e utiliza mecanismos para controlar o congestionamento. Na ausência de congestionamento na rede, os algoritmos utilizados na prevenção de congestionamento não devem influenciar o desempenho do protocolo.

Os algoritmos de controle de congestionamento usados pelo SCTP são baseados em [All99] e são adaptados ao uso do protocolo SCTP e usam sempre a mesma terminologia do protocolo TCP.

4.8.1 Diferenças do Controle de Congestionamento do SCTP em relação ao TCP

O mecanismo de controle de congestionamento do protocolo SCTP é aparentemente equivalente ao protocolo TCP com pequenas mudanças. As mudanças mais significativas são.

- No protocolo SCTP pode haver um número ilimitado de *Gap Ack Blocks*. Já o protocolo TCP tem um limite máximo de três blocos *SACK*. Isto permite o protocolo SCTP ser mais resistente à perda dos reconhecimentos, melhorando a capacidade de tratar a perda (não seqüencial) de múltiplos pacotes.
- O protocolo TCP permite a mudança para a fase *Congestion Avoidance* ou *Slow Start* quando a variável *CWND = SSTRRESH*. No protocolo SCTP permanece a fase *Slow Start*.
- A fase *Fast Retransmit* (retransmissão rápida) dos pacotes é controlada pela variável *CWND*. Isso diminui a velocidade com que o protocolo SCTP reage à perda de pacotes.

A principal diferença no controle de congestionamento é que o protocolo SCTP suporta *Multi-Homing*, então os *endpoints* podem ser alcançados por vários caminhos. Os parâmetros de controle de congestionamento são utilizados para cada um dos endereços de destino (não para cada par origem-destino). Os atuais algoritmos de controle de congestionamento que são detalhados nas próximas seções fazem as seguintes suposições.

1. O protocolo SCTP pode retransmitir utilizando um endereço de transporte diferente da transmissão original quando este for considerado “inativo” (maiores detalhes na seção 4.10.1).
2. Os parâmetros que são utilizados no controle de congestionamento devem deteriorar-se, caso o endereço não seja usado por um período de tempo suficientemente longo.
3. Um *endpoint* utiliza a fase *Slow Start* na primeira transmissão para cada um dos endereços de destino.

É importante frisar que o controle de congestionamento no protocolo SCTP é aplicado sempre à associação inteira, e não a *streams* individuais.

4.9 Algoritmos *Slow-Start* and *Congestion Avoidance* no SCTP

Os algoritmos S*low Start e Congestion Avoidance* são usados para controlar a quantidade de dados que estão sendo inseridas na rede. Idêntico ao protocolo TCP, o SCTP *enpoint* usa as seguintes variáveis (todas possui o tamanho em *bytes*) para controlar a taxa de transmissão de dados, e ainda utiliza uma variável adicional de controle.

CWND (*Congestion Control Window*) – A janela de controle de congestionamento, é ajustada pelo transmissor de acordo com condições observadas na rede.

RWND (*Receiver Advertised Window*) – O receptor informa o tamanho da janela, que é ajustada baseada no espaço disponível no *buffer* para a chegada de pacotes.

SSTHRESH (*Slow-Start Threshold*) – Essa variável é utilizada pelo remetente para distinguir a fase *Slow Start* e *Congestion Avoidance*.

PARTIAL BYTES ACKED – Essa variável é utilizada durante a fase *Congestion Avoidance* para facilitar a configuração da variável *CWND*.

Um transmissor SCTP deve guardar um conjunto destas variáveis de controle para cada endereço destino de seu par (caso seu par seja *Multi-Homing*), a variável *RWND* é mantida para associação e não para cada endereço destino.

4.9.1 *Slow-Start*

O algoritmo *Slow Start* no protocolo SCTP é utilizado com a finalidade de determinar a capacidade disponível da rede para a transferência de dados, pois no início da transmissão de dados não é conhecida a condição da rede. Durante a fase *Slow Start* as variáveis são configuradas da seguinte maneira:

- O valor inicial de *CWND* antes da transmissão de dados ou após um período suficientemente longo inativo deve ser *CWND* < = 2 * *MTU*.
- Após ocorrer um *timeout* de retransmissão o valor de *CWND* = 1 * *MTU*.
- O valor inicial da variável *SSTHRESH* pode ser igual ao valor de *RWND*.

- O valor da variável *CWND* aumenta, não podendo ser maior que a variável *SSTHRESH*.
- Quando o *endpoint* é *Multi-Homing*, e este recebe um *SACK chunk* que seja maior que o *Cumulative TSN Ack Point*, então, ele deve atualizar o valor da variável *CWND,* do qual recebeu o reconhecimento.
- De acordo com [Ste02] quando o ponto final não transmite dados para um dado endereço de transporte, o *CWND* do endereço de transporte deve ser configurado para (2**MTU*).

4.9.2 *Congestion Avoidance*

O algoritmo *Congestion Avoidance* é utilizado quando o valor da variável *CWND* ultrapassa o valor da variável *SSTHRESH*. Então o valor da variável *SSTHRESH* deve ser aumentado por (1**MTU*). Para isso acontecer os seguintes passos devem ser seguidos:

1. O valor inicial da variável *Partial Bytes Acked* deve ser zero.
2. No recebimento de cada *SACK chunk* que ultrapassa o *Cumulative TSN Ack Point* o valor da variável *Partial Bytes Acked* será igual ao valor total de *bytes* de todos os *chunks* reconhecidos.
3. No transmissor quando o valor da variável *Partial Bytes Acked* for igual ou maior ao valor da variável *CWND* antes da chegada de um *SACK chunk*, então o valor da variável *CWND* deve ser aumentado pelo valor da MTU, e o valor da variável *Partial Bytes Acked* será (*Partial Bytes Acked - CWND*).
4. O valor da variável *Partial Bytes Acked* possui o valor zero quando o *endpoint* receber o reconhecimento de todos os dados que foram transmitidos.

4.9.3 Algoritmo *Fast Retransmit* no Protocolo SCTP

No protocolo SCTP o *Fast Retransmit* funciona da seguinte maneira. Quando um *endpoint* emissor receber quatro *SACKs chunks* (este *chunk* é enviado para indicar que recebeu um pacote SCTP e também informa todos os "buracos", na seqüência TSN do *DATA chunk*) consecutivos indicando a falta de um *DATA chunk,* este deve ser marcado para

retransmissão imediatamente. A partir desse momento as variáveis ficam configuradas da seguinte forma:

- CWND = SSTHRESH ; SSTHRESH = (CWND/2), e *Partial Bytes Acked* = 0 ;
- O temporizador de transmissão é reiniciado caso o último *SACK* seja menor que o número de seqüência de transmissão (TSN) enviado para aquele endereço.

4.10 Gerenciamento de Falhas no Protocolo SCTP

Nesta seção mostraremos os principais mecanismos utilizados no gerenciamento de falhas pelo protocolo SCTP. Eles são: monitoramento de falha no *endpoint* e monitoramento de falha no *path*.

4.10.1 Monitoramento de Falha no *Endpoint*

O protocolo SCTP para controlar falhas no *enpoint* guarda um contador que armazena o número total retransmissões consecutivas a seu par *endpoint* pertencente à associação (no *endpoint Multi-Homing*, as retransmissões incluem todos os endereços de transporte). O contador é reiniciado toda vez que um *chunk* de dados enviado a esse par *endpoint* é reconhecido (pela recepção de um *SACK*), ou um *HEARBEART-ACK* é recebido do par *endpoint*.

A partir do momento que o valor do contador exceder o limite indicado no parâmetro "*Association.Max.Retrans"* do protocolo SCTP, o *endpoint* considera o *endpoint* par inalcançável e para de transmitir dados, nessa situação a associação é fechada automaticamente.

4.10.2 Monitoramento de Falha no *Path*

O protocolo SCTP utiliza o *HEARBEAT chunk* que é bastante eficiente na detecção de falhas no *enpoint*. Os *HEARBEATs chunk* são enviados a todos os caminhos, caso não estejam sendo utilizados para a transmissão de *DATA chunks*.

No protocolo SCTP cada caminho é denominado "inativo" ou "ativo" [Jun00]. Um caminho é "inativo" quando uma transmissão estava acontecendo e este falhou

repentinamente devido a uma falha física na rede, impossibilitando a transmissão de dados. Quando um *HEARTBEAT chunk* é enviado a um endereço e não é reconhecido dentro do RTO, então o contador de erro é incrementado e quando o valor desse contador atingir o parâmetro "*Path. Max. Retrans*" esse caminho é considerado "inativo" e uma notificação deve ser enviada à ULP. Quando o caminho é utilizado pelo transmissor para enviar um pacote SCTP, e o receptor confirma que recebeu o pacote, então o caminho é considerado "ativo".

É importante salientar que o *endpoint Multi-Homing*, possui um contador de erro para cada um dos endereços de transporte destino ao par *endpoint*.

4.11 Segurança no protocolo SCTP

O *Four-Way-Handaske* no SCTP habilita mecanismos de segurança que permitem o SCTP ser robusto contra os ataques *blind* (escuro) *denial-of-service* (ataque onde o agressor não está habilitado a interceptar pacotes, mas utiliza a interface para enviar maliciosos pacotes para um ou mais nodos SCTP utilizando um endereço IP falso). Um exemplo de um ataque fraudulento seria a interseção e o emprego errado da informação autorizada, tais como números de cartão de crédito. O objetivo da fraude é obter serviços sem autorização e sem pagar por esses serviços.

O mecanismo *cookie* utilizado pelo protocolo SCTP, foi concebido para que os recursos sejam somente reservados para o servidor (*endpoint)* depois da validação do *cookie* feita pelo SCTP. Para exemplificarmos esse ataque temos o TCP *SYN flooding*. O objetivo do ataque *flooding* é causar a perda do serviço, incluindo potencial violação do *firewall* e o incorreto comportamento no sistema que é alvo através da exaustão de recursos.

O *Verification Tag* é usado durante a configuração de uma associação para assegurar que todo pacote pertence à associação corrente. Ele é incluído em todo pacote SCTP, tornando-se difícil à inserção de pacotes estranhos no estabelecimento de uma associação, para descobrir o valor de 32 *bits* isto é praticamente impossível, enquanto o tamanho do campo no TCP possui 16 *bits*, sendo mais vulnerável.

4.12 Considerações Finais

Neste capítulo foi visto o funcionamento do protocolo SCTP, apresentou-se também as principais características desse protocolo de transporte. Dentre elas, pode –se destacar o

formato do pacote SCTP, os vários tipos de *chunk* de controle e o *chunk* de dados, as fases de uma associação, a entrega ordenada e desordenada das mensagens, empacotamento de mensagens, as variáveis utilizadas no congestionamento e os mecanismos utilizados no gerenciamento de falhas. Foi mostrada a importância das características *Multi-Streaming* e *Multi-Homing* que não estão presentes no protocolo TCP.

No próximo capítulo será avaliado o desempenho do protocolo SCTP e TCP em uma rede com QoS e sem QoS. Nas simulações foi utilizada a ferramenta NS (*Network Simulator*).

Capítulo 5.0
Análise de Desempenho do SCTP

Neste capítulo será avaliado e comparado o desempenho do protocolo SCTP e TCP. Serão apresentados e analisados os resultados dos experimentos feitos com a ferramenta de simulação NS (*Network Simulator*).

5.1 Técnicas de Avaliação de Desempenho

As técnicas de **medição**, **modelagem analítica e simulação** [Jan91, Hig98] são utilizadas para estudar o comportamento da Internet.

A medição é uma técnica para compreender o comportamento atual da Internet, com relação a uma série de diferentes pontos de vista, como os protocolos mais utilizados, volume de tráfego, informações sobre o tamanho dos pacotes e obtenção do número de roteadores entre pontos distintos na Internet.

A modelagem analítica é uma técnica que requer um forte embasamento matemático e não precisa de equipamentos especiais. O risco de confiar em resultados analíticos é que as simplificações são tantas que o comportamento original da Internet pode ser perdido.

A simulação se baseia na execução e construção de programas, por isso é a técnica mais utilizada. O simulador *Network Simulator* [NS] foi utilizado por ser bastante difundido no meio acadêmico, possuir código fonte aberto e ser gratuito. Na próxima seção faremos uma breve descrição do simulador.

5.2 O simulador de redes *Network Simulator* (NS)

O *Network Simulator* – 2 [NS] surgiu como uma variante do simulador REAL (*Realistic and Larg*e) [Kes97] e em 1995 seu desenvolvimento foi suportado como parte do projeto VINT *(Virtual InterNetwork Testbe*d) [Vin]. Atualmente seu desenvolvimento e distribuição são mantidos pelo ISI (*Information Sciences Institute*), financiado pela DARPA (*Department Advanced Research Project Agency*.

O NS é um simulador baseado em eventos discretos e orientado a objetos para a simulação de redes. As bibiliotecas do protocolo SCTP não são distribuídas diretamente no pacote do NS, sendo disponibilizadas como *patchs* em [Pel02] que podem ser baixadas e adicionadas ao módulo básico através de recompilação do núcleo do simulador.

O NS possui um grande número de bibliotecas de protocolos implementadas, tais como os protocolo TCP,UDP e IP além de disciplinas de serviços,como, WFQ (*Weight Fair Queueing*), protocolos para redes móveis, como o IP móvel, tecnologias de redes sem fio locais e de longa distância, como, 802.11, *Bluetooth* e GPRS (*General Packet Radio Service*). O NS fornece também bibliotecas de funções para a geração de alguns tipos de tráfego como:

CBR (*Constant Bit Rate*) utilizado para simular tráfego constante e voz, ON-OFF para tráfego em rajada e voz comprimida, FTP para gerar tráfego correspondente a aplicações de transferência de arquivos e VBR (*Variable Bit Rate*) para tráfego com taxa de dados variável.

O NS possui também o *nam* (*Network Animator*) [Est00]. O *nam* é um animador de redes utilizado para entender o que ocorre durante a simulação. Pode-se visualizar a topologia da rede, bem como acompanhar o fluxo dos pacotes e seus conteúdos.

O NS adota duas linguagens de programação [Kam02]: C++ para o núcleo do simulador (*back-end*) e Otcl para construção de *scripts* e modelagem da simulação (*fron-end*).Os *scripts* que contém os comandos e o modelo a ser simulado são construídos em Otcl (*Object Tool Control Language*), uma versão orientada a objetos da linguagem de script tcl (*Tool Control Language*). Otcl é uma linguagem interpretada, sem necessidade de compilação, pois os scripts são tarefas interativas e freqüentemente alteradas.

5.3 Ambiente de Simulação

O *Hardware* e *Software* que foram utilizados nas simulações são os seguintes:

- Micro computador Pentium III 750 Mhz;
- Memória RAM de 374 MB;
- Espaço em disco de 100 MB para os resultados da simulação;
- Sistema Operacional Linux (Debian) versão 3.0;
- Foi utilizado o *Network Simulator* versão 2.1b8.

5.4 Métricas

Nas simulações foram utilizadas duas métricas para avaliar o desempenho do protocolo SCTP e TCP:

- Atraso (*Delay*): Caracteriza o atraso fim a fim entre fonte e destino dos pacotes;
- Variação do atraso (*Jitter*): Valor absoluto entre os tempos de chegada (c) de dois pacotes consecutivos menos os seus tempos de saída (s), ou seja, $|(cj-sj) – (ci – si)|$;

5.5 Topologia da Simulação

A topologia adotada na simulação é ilustrada na Figura 40. Composta de 17 nós, os pontos estão interligados ao roteadores por um enlace de 5Mb e um atraso de 1 ms. O enlace entre os nós R0 a R1 é de 2Mb com atraso de 20 ms, igualmente ao de R1 a R2.

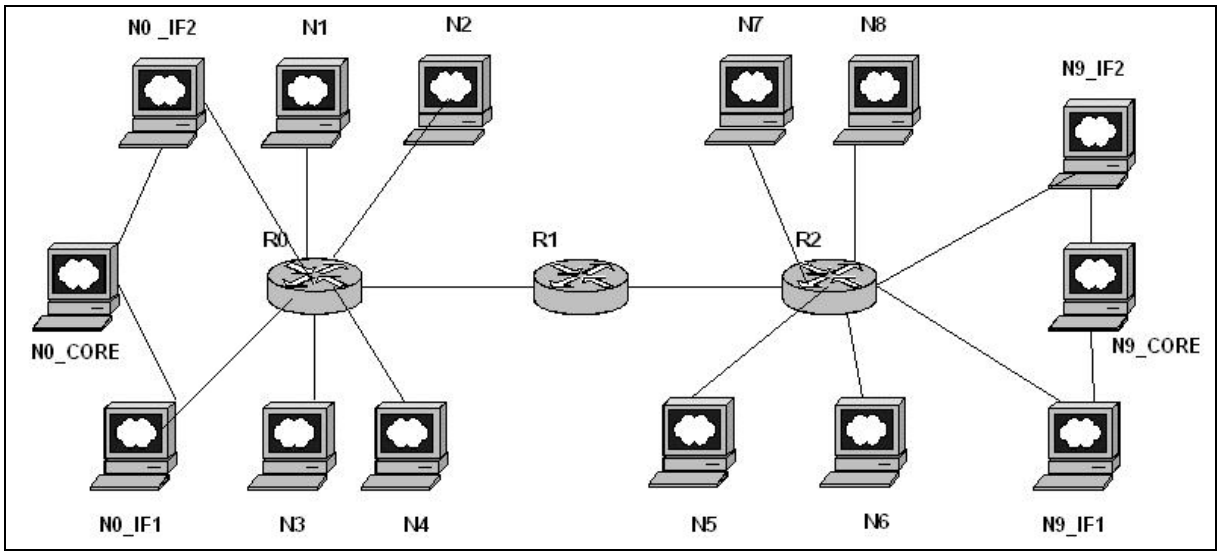

Figura 40: Topologia adotada na simulação

Optou-se por esta topologia por gerar uma situação de congestionamento fazendo com que a soma do tráfego gerado pelas fontes exceda a capacidade entre os enlaces R0 a R1 e R1a R2.

5.6 Parâmetros e Fatores da Simulação

As fontes de tráfego utilizadas na simulação são ilustradas na Tabela 7. Além disso, em todos os testes simulados todas as aplicações começam a transmitir no momento 0,10 segundos e terminam ao 100 segundos. com o término da simulação. Foram realizadas simulações com tempos maiores e os resultados obtidos não apresentaram variações significativas. O tempo de 100 segundos é suficientemente generoso para observar as métricas de interesse.

Na nossa topologia temos dois nodos que são (SCTP *Multi-Homing*) (N0_CORE, N9_CORE) representados pelas interfaces (N0_IF1, N0_IF2, N9_IF1, N9_IF2).

Origem	Agente	Aplicação	Taxa	Destino
N0_CORE	Sctp0	FTP	-	N9_CORE
N0_IF1	Sctp0	FTP	-	N9_IF1
N0_IF2	Sctp0	FTP	-	N9_IF1
N1	Tcp0	FTP	-	N5
N1	Voz0	ON/OFF	-	N8
N2	Sctp2	FTP	-	N5
N2	Tcp4	FTP	-	N6
N3	Sctp10	FTP		N6
N3	Sctp4	FTP	-	N7
N4	Tcp6	TELNET	-	N7
N4	Udp1	CBR	64k	N8

Tabela 7: Fontes de Tráfego utilizada na simulação para uma rede sem QoS

No tráfego CBR foram transmitidos pacotes de 100 *bytes* a uma taxa de 64 *Kbps*. Foram utilizadas também fontes FTP no (tráfego *background*). Foram utilizadas fontes ON/OFF para representar o tráfego de voz de ponto N1 a N8, com o *idle time* sendo exponencialmente distribuído. Essas fontes alteram entre a fase ON (período de transmissão) e OFF (período inativo), sendo considerado um tráfego em rajadas.

Para representar o tráfego de dados (tráfego principal) foram utilizadas fontes FTP, simulando a transferência de arquivos e fontes TELNET. Não foi configurada a taxa de transmissão para fontes FTP e nem TELNET, pois os protocolos TCP e SCTP adaptam a taxa de transmissão de acordo com as condições da rede. As fontes FTP e TELNET transmitiram pacotes de 1500 *bytes*.

O mecanismo de gerenciamento de filas nos roteadores utilizado é o *Drop Tail*, por representar o modelo da Internet atual.

A simulação foi replicada 30 vezes para garantir confiabilidade estatística aos resultados e permitir trabalhar corretamente com intervalos de confiança. Esse número foi escolhido porque testes com um número maior de replicações não mostraram nenhuma variação significativa nos resultados.

A Tabela 8 ilustra a configuração da fontes de tráfego utilizada em uma rede com QoS. As fontes de tráfego partem da mesma origem para o mesmo destino e possuem as mesmas características da rede sem QoS.

Origem	Agente	Aplicação	PHB	Valor do DSCP	Taxa	Destino
N0_CORE	sctp0	FTP	AF	10	-	N9_CORE
N0_IF1	sctp0	FTP	AF	10	-	N9_IF1
N0_IF2	sctp0	FTP	AF	10	-	N9_IF1
N1	Tcp0	FTP	AF	10	-	N5
N1	voz0	ON/OFF	BE	0	-	N8
N2	sctp2	FTP	AF	10	-	N5
N2	Tcp4	FTP	BE	0	-	N6
N3	Sctp10	FTP	AF	10		N6
N3	sctp4	FTP	BE	0	-	N7
N4	Tcp6	TELNET	BE	0	-	N7
N4	udp1	CBR	BE	0	64k	N8

Tabela 8: Fontes de Tráfego utilizada em uma rede com QoS

Foi utilizado o medidor *Token Bucket*. A reserva da largura de banda foi configurada da maneira abaixo. O mecanismo utilizado no gerenciamento de filas no roteadores de uma rede que implementa DiffServ foi o RED.

- PHB AF possui uma reserva de 800 *Kbps* (40% de 2 MB).
- O PHB EF possui uma reserva de 1200 *Kbps* (60 % de 2 MB).

5.7 Estudo de Caso

Para avaliar o comportamento do protocolo SCTP foram feitas várias medições e comparações com o protocolo de transporte TCP. Foram utilizados quatro cenários nas simulações que são:

- Cenário1 – Rede sem QoS (1 fonte FTP e 10 fontes CBR)
- Cenário 2 – Rede com QoS (1 fonte FTP e 10 fontes CBR)
- Cenário 3 – Rede sem QoS (2 fontes FTP e 2 fontes CBR)
- Cenário 4 –Rede com QoS (2 fontes FTP e 2 fontes CBR)

Estes cenários foram escolhidos, por haver uma variação de fontes no tráfego principal e *background*. Foi utilizado o serviço do melhor-esforço por representar o modelo da Internet atual, e utilizou – se também qualidade de serviço através da implementação da arquitetura DiffServ.

5.8 Resultados da Simulação

A análise dos dados gerados nas simulações está restrita as informações relacionadas com o atraso e *jitter* do protocolo SCTP e TCP. Está análise está baseada no intervalo de tempo entre a chegada sucessiva de pacotes. A Tabela 9 ilustra os indicadores estatísticos.

	Cenário 1				**Cenário 2**			
	Atraso		**Jitter**		**Atraso**		**Jitter**	
	SCTP	**TCP**	**SCTP**	**TCP**	**SCTP**	**TCP**	**SCTP**	**TCP**
Média	0.09880	0.10227	0.005709	0.00600	0.08544	0.08544	0.00446	0.00532
Mediana	0.10086	0.10432	0.0040	0.00424	0.08710	0.08962	0.00294	0.00422
Máximo	0.16267	0.16028	0.08521	0.10772	0.15023	0.15169	0.06835	0.07096
Mínimo	0.04337	0.04260	0.0	0.0	0.04240	0.04260	0.0	0.0
Amplitude	0.11930	0.11768	0.08521	0.10772	0.10782	0.10908	0.06835	0.07096
Variância	0.000212	0.000183	0.000046	0.000043	0.000334	0.000299	0.000033	0.000029
Desvio Padrão	0.01457	0.00659	0.00684	0.01353	0.01829	0.01730	0.00580	0.00541
Coeficiente de Variação	0.14749	0.13229	1.19931	1.09921	0.21408	0.19735	1.30086	1.01729
	Cenário 3				**Cenário 4**			
	Atraso		**Jitter**		**Atraso**		**Jitter**	
	SCTP	**TCP**	**SCTP**	**TCP**	**SCTP**	**TCP**	**SCTP**	**TCP**
Média	0.17524	0.18085	0.00598	0.00846	0.10005	0.10911	0.00390	0.00521
Mediana	0.18363	0.18698	0.00040	0.00422	0.09572	0.10833	0.00080	0.00422
Máximo	0.27784	0.27208	0.11081	0.12273	0.23965	0.23129	0.11185	0.09580
Mínimo	0.04433	0.04260	0.0	0.0	0.04240	0.04260	0.0	0.0
Amplitude	0.23351	0.22947	0.11081	0.12273	0.19725	0.18868	0.11185	0.09580
Variância	0.000967	0.000694	0.000142	0.000162	0.001052	0.001125	0.000065	0.000042
Desvio Padrão	0.03110	0.02634	0.01193	0.01276	0.03243	0.03354	0.00810	0.00651
Coeficiente De Variação	0.17750	0.14567	1.99366	1.50791	0.32419	0.30740	2.07684	1.24941

Tabela 9 : Indicadores Estatísticos

No cenário 1 o atraso médio do protocolo SCTP é menor que o TCP, já o atraso máximo do SCTP é maior que o TCP. O média do *jitter* do protocolo SCTP é menor que a do TCP. O *jitter* mínimo SCTP e TCP são iguais.

No cenário 2 a média do atraso dos protocolo SCTP é menor que o TCP. O *jitter* mínimo dos protocolos SCTP e TCP são iguais. O atraso e o *jitter* máximo do protocolo SCTP é menor que o do TCP.

No cenário 3 a média do atraso e *jitter* do protocolo SCTP é menor que o TCP. O atraso mínimo do protocolo SCTP é maior que o do TCP, enquanto que o *jitter* mínimo dos protocolos TCP e SCTP são iguais.

No cenário 4 a média do atraso e *jitter* do SCTP é menor que o do protocolo TCP. O *jitter* mínimo dos protocolos TCP e SCTP são iguais, enquanto que o *jitter* máximo do protocolo SCTP é menor que o TCP.

Observa-se que os valores do atraso e *jitter* do protocolo SCTP e TCP nos cenários 2 e 4 são menores que os valores dos cenários 1 e 3, pois nos cenários 2 e 4 há a utilização de QoS através da implementação de DiffServ. Nos cenários 2 e 4 os pacotes são encaminhados em classes de serviço, onde existe uma largura de banda pré-definida para cada classe. Nos cenários 1 e 3 os pacotes pertencem ao serviço do melhor-esforço, onde são descartados aleatoriamente durante o congestionamento.

5.9 Considerações Finais

Neste capítulo foi analisado o atraso e o *jitter* do protocolo SCTP e TCP. Utilizou-se a ferramenta de simulação NS (*Network Simulator*) por ser gratuita e de código fonte aberto.

Apresentou-se a topologia, fatores, parâmetros, métricas e os vários cenários utilizados para a análise dos resultados obtidos na simulação. No próximo capítulo serão mostrados as conclusões deste trabalho, contribuições e trabalhos futuros.

Capítulo 6.0
Conclusão

Este capítulo apresenta as conclusões relacionadas a proposta dessa dissertação. Apresenta também a contribuição e os trabalhos futuros como expansão desta pesquisa.

6.1 Considerações Finais

O crescimento explosivo da Internet nos ano 90 influenciou para que as redes TCP/IP possuíssem um grande número de computadores interconectados. Apesar disso, o protocolo TCP tem apresentado problemas e limitações com o surgimento de novas aplicações e serviços. Neste contexto, a dissertação em questão é uma primeira etapa para uma compreensão melhor do protocolo SCTP.

O serviço do melhor esforço provido pela Internet atual, não apresenta um desempenho satisfatório ao usuário final, pois os pacotes recebem o mesmo tipo de tratamento, não havendo garantias de largura de banda, atraso e perda de pacotes. No congestionamento os pacotes são descartados aleatoriamente. Com o surgimento dos serviços avançados (Telemedicina, Vídeo Conferência, etc), viu –se a necessidade de um tratamento diferenciado, pois o serviço do melhor-esforço não oferece garantias de QoS.

Foram vistas algumas vantagens do protocolo SCTP sobre o TCP. Ele possui proteção contra os ataques *denial of service* através do campo *Verification Tag* de 32 *bits* que é utilizado em todo pacote SCTP no estabelecimento de uma associação e ainda utiliza o mecanismo *cookie* para finalizar o estabelecimento de uma associação. Permite ainda *hosts Multi-Homing* (várias interfaces IP para uma particular associação), *Multi-Streaming* e empacotamento.

A análise dos dados obtida com a simulação através da ferramenta *Network Simulator*, foi descritiva, isto é, não foi realizada inferência estatística. Observou se que o atraso médio e *jitter* médio do protocolo SCTP é menor que o do TCP em todos os cenários analisados. O atraso médio e o *jitter* médio do protocolo SCTP nos cenários 2 e 4 onde houve a utilização de QoS através da arquitetura DiffServ apresentaram valores menores que o do cenários 1 e 3 que utilizam o serviço do melhor-esforço.

Conclui-se que protocolo SCTP pode tornar-se um protocolo de escolha dentro dos sistemas atuais e futuros por apresentar uma maior confiabilidade e segurança que o protocolo TCP.

6.2 Contribuições

A seguir apresentam-se as contribuições obtidas com o desenvolvimento deste trabalho:

- Estudo do protocolo TCP frente aos problemas de transporte mensagens de sinalização PSTN (*Public Switched Telephone Network*) dentro de redes IP. Com este estudo foi possível identificarmos as principais fraquezas que o TCP possui quando posto a transportar mensagens de sinalização;
- Foi feito um estudo sobre o protocolo SCTP e utilização de mecanismos para o provimento de QoS na Internet;
- O Estudo comparativo do protocolo TCP com o SCTP foi de grande valia para mostrar a viabilidade da utilização do protocolo SCTP na Internet atual e futuros sistemas. Esse estudo do SCTP mostrou as principais características e vantagens do protocolo sobre os protocolos de transporte TCP e UDP utilizados na Internet;

6.3 Trabalhos Futuros

A partir do nosso trabalho propõem-se alguns estudos que podem ser feitos como trabalhos futuros. Entre os trabalhos futuros destacam-se os seguintes:

- Utilizar outros algoritmos de gerenciamento de filas nos roteadores como SFQ (*Stochastic Fair Queuing*), CFQ (*Class Based Fair Queing*), WFQ (*Weighted Fair Queuing*) para observar o comportamento do protocolo TCP e SCTP;
- Realizar simulações com o protocolo SCTP e TCP em redes sem fio;
- Analisar o comportamento do protocolo SCTP e TCP com a utilização da arquitetura IntServ para a implementação de QoS.
- Utilizar a técnica de medição para avaliar o desempenho do protocolo SCTP e TCP.
- Realizar inferência estatística, como, por exemplo, teste de hipótese.

6.4 Referências Bibliográficas

[All98] Allman, M., Floyd, S. and Partridge, C**. Increasing TCP's Initial Window**, Internet RFC 2414 - September 1998.

[All99] Allman, M., Paxson, V. and Stevens, W. **TCP Congestion Control** Internet RFC 2581 - April 1999.

[Bla98] Black, D. et. Al., **Na Architeture for Differentiated Services**, Internet RFC 2475 - December 1998.

[Bra89] Braden R., **Requirements for Internet *Host*s -- Communication Layers**, RFC Internet RFC 1122 – October 1989.

[Bra94] Braden, R., Clark, D & Shenker, S., **Integrated Services in the Internet Architeture: an overview,** Internet RFC 1633 - June 1994.

[Bra97] Braden, R., et Al., **Resource Reservation Protocol (RSVP) version 1 Functional Especification**, Internet RFC 2205 - September 1997.

[Bra98] Braden, R., et Al, **Recommendations on Queue Management and Congestion Avoidance in the Internet,** Internet RFC 2309 – April 1998.

[Cam02a] Camarillo G., Schulzrinne, **Signalling Transport Protocols**, February 2002, Disponível na Interne em http:// www.cs.columbia.edu/~library/TR-repository/ reports/reports-2002/cucs-002-02.pdf

[Cam02b] Campos M. A., et Al, **Análise Estatística de Descarte de Pacotes em Redes TCP**, XX Congresso da SBRC, Rio de Janeiro/RJ, Maio de 2002.

[Cla88] Clark, D., **The Design Philosophy of the DARPA Internet Protocols**, Proceedings of ACM SIGCOMM' 88, August 1988

[Cla94] Clark, D., Braden R., and S. Shenker, **Integrated Services in the Internet Architecture: an Overview**, RFC-1633, 1994

[Cla98] Clark, D. and Fang, W. **Explicit Allocation of Best-Effort Packet delivery Service**, IEEE/ACM Transactions on Networking, Vol. 6, Nº 4, August 1998.

[Coe02] Coene L., **Stream Control Transmission Protocol Applicability Statement**, RFC 3257 – April 2002.

[Com94] Comer, D. **Internetworking with TCP/IP: Design**, **Implementation and Internals** 2nd ed., vol. 2, Prentice-Hall, New Jersey, USA - 1994.

[Con00] Conrad Phillip, et Al, **Improving Multimedia Performance Over Loss Networks Via SCTP**. September 2000.

[Cra98] Crawley, E., et. Al., **A Framework for QoS-based Routing in the Internet**, Internet RFC 2386 – August 1998.

[Cro99] Croll, A & Packman, E., **Managing Bandwith: Deploying QoS in Enterprise Networks**, Prentice Hall, 1999.

[Dob99] Dobreff, A., **Differentiated Services in ATM-Networks**, Project – Computer Networks and Distributed Systems University of Bern, Switzerland, November 1999.

[Eas01] Eastlaked D.**,** Jones P., **US Secure Hash Algorithm1 (SHA 1),** RFC 3174 - September 2001.

[Est00] Estrin, D., **Network Visualization with Nam, the VINT Network Animator**, IEEE Computer, November 2000.

[Fer98a] Ferguson, P. & Huston, G., **Quality of Service in the Internet: Delivering QoS on the Internet, and in Corporate Networks**, John Wiley & Sons, 1998.

[Fer98b] Ferguson, P. and Huston, G. **Quality of Service in the Internet: Fact, Fiction or Compromise?** Appeared in the proceedings of INET'98, July 1998.

[Fil98] Filho, Reinaldo P.**, Desvendando o TCP,** 10 de Abril de 1998, volume 2, número 4.

[Flo99a] Floyd, S. and Henderson, T. **The New Reno Modification to TCP's Fast Recovery Algorithm**, Internet RFC 2582 - April 1999.

[Hei99] Heinanen, J., Baker, F. & Wroclawski, J., **Assured Forwarding PHB,** Internet RFC 2597 - June 1999**.**

[Hig98] Higginbotton, G. **Performance Evoluation of Communication Networks Trade Cloth**, Artech House Publishers, April 1998.

[Jac92] Jacobson V., Braden R., Borman D., **TCP Extensions for High Performance,** Internet RFC 1323 – May 1992.

[Jac98] Jacobson V., **Congestion Avoidance Control**, ACM SIGCMON 88, August 1998.

[Jac99] Jacobson, V., Nichols, K. & Poduri, K., **An Expedited Forwarding PHB,** Internet RFC 2598 - June 1999.

[Jai92] Jain R., **Myths about Congestion Management in High Speed Networks**, Internetworking: Res. And Exp., Vol .3, 1992.

[Jun00] Jungmaier A., et Al ,**SCTP- A Multi-Link End-to-End Protocol for IP – Based Networks** , published in AEÜ, International Journal of Electronics and Communications, November 2000. Disponível na Internet em http://www.sctp.be/

[Jus00] Jussi, Lauk., **Integrated Services,** Research Seminar on IP QoS Department of Computer Science, University of Helsinki, October 2000.

[Kam00] Kamienski, C., Sadok, D., **Qualidade de Serviço na Internet Minicurso 2**, 18^0 Simpósio Brasileiro de Redes de Computadores, Maio 2000.

[Kam02] Kamienski, C.A., Sadok, D., Cavalcanti, D. A. T., Souza, D. M. T., Dias, K. L., **Simulando a Internet: Aplicações na pesquisa e no ensino**,21 JAI (Jornada de Atualização em Informática), Congresso da SBC, Florianópolis/SC, julho de 2002.

[Kar87] Karn, P. and Partridge, C., **Improving Round-Trip Time Estimates in Reliable Transport Protocols.** Appeared in the proceedings of ACMSIGCOMM 87, pp. 2-7, 1987.

[Kes97] Keshav, S., **REAL 5.0 Overview**, http://www.cs.cornell.edu/skeshav/real/overview.html August 1997.

[Lef96] Lefelho, C, et. Al., **Congestion Control for Best-Effort Service: Why We Need A New Paradigm**, IEE Network, January 1996

[Mag01] Magalhães, J. M. H., Guardieiro P. R., **Avaliação de um Ambiente de Serviços Diferenciados com Tráfego de Vídeo MPEG-4**, XIX Simpósio Brasileiro de Telecomunicações, SBrT'01, Fortaleza-CE, Setembro 2001

[Mar99] Martins Joberto, **Qualidade de Serviço (QoS) em Redes IP Princípios Básicos, Parâmetros e Mecanismos**, JSMNet Networking Reviews - Vol. 1 - N° 1,Setembro 1999. Disponível na Internet em http://www.jsmnet.com

[Mcc98] Mccabe James D., **Practical Computer Network Analysis and Design**, Morgan Kaufmann Series in Networking, 1998

[Mes01] Mesquita, Gur., Márcio, **MPLS – Multiprotocol Label Switching,** Julho de 2001. Disponível na Internet em http:www.larces.uece.br/tutoriais/MPLS_MARCIO_TUTORIAL.PDF

[Met02] Metz, C**., IP QoS:Traveling in First Class on the Internet**. Disponível na Internet em: fttp://www.computer.org/internet/v3n2/w2on-fig2.htm, consultada em Abril de 2002

[Mor02] Morgan Kaufmann Pub., **TCP Congestion Control**, Disponível na Internet em http://www.mkp.com/books_catalog/cn/book/node84.htm, consultada em Abril de 2002.

[Nla02] NLANR Engineering Service, **Trace and TCP Fundamentals**, Disponível na Internet em: http://ncne.nlanr.net/research/tcp/debugging/fundamentals.html , consultada em Abril de 2002.

[NS] **Network Simulator** (versão 2.1b8), Disponível na Internet em: http://www.isi.edu/nsnam/ns/.

[Odl99] Odlyzko, A. P., **The Current State and Likely Evolution of the Internet**, IEE Globecom 99, December 1999

[Ong02] Ong L., Yoakum J., **An Introduction to the Stream Control Transmission Protocol (SCTP),** RFC 3286 – May 2002.

[Pag00] Pagani Carlos E., Magalhães Maurício F., **Agregação de Trafego em um Comutador Adaptativo com Serviços Integrados IP sobre ATM,** 18^0 Simpósio Brasileiro de Redes de Computadores, Maio 2000.

[Pax99] Paxson, V. et. Al., **Known TCP Implementation Problems,** Internet RFC 2525 – March 1999.

[Pel02] PEL Home Page, **Protocol · Engineering · Laboratory.** Disponível na Internet em http://pel.cis.udel.edu/ ,Página consultada em Maio de 2002.

[Pod98] Poduri, K. and Nichols, K. **Simulation Studies of Increased Initial TCP Window Size**, Internet RFC 2415 - September 1998.

[Pos81] Postel, J. ed Al, **Transmission Control Protocol DARPA Internet Program Protocol Specification,** Internet RFC 793 – September 1981.

[Rav01] Ravier T., Brennan R**.,** Curran Thomas **, Experimental Studies of SCTP Multi-Homing,** November 2001. Disponível na Internet em http://telecoms.eeng.dcu.ie/symposium/.

[Ros99] Rosen, E; **Multiprotocol Label Switching Architeture,** Internet Draft, <draft-ietf-mpls-arch-05.txt; > April 1999.

[Sch00] Schmitd Ana, **O Protocolo RSVP e o Desempenho de Aplicações Multimídias**, 12 de maio de 2000 volume 4, número 3.

[Sct02] **SCTP for Begginners: SCTP Packets**. Página disponível na Internet http://tdrwww.exp-math.uni-essen.de/pages/forschung/sctp_fb/. Consulta em Maio de 2002.

[She97] Shenker, S., **Specification of Guaranteed Quality of Service**, RFC 2212 - September 1997.

[Sol92] Sollins, K. **The TFTP Protocol** (Revision 2) - RFC 1350, July 1992.

[Ste00] Stewart, R., et Al, **Stream Control Transmission Protocol**, RFC 2960 - Ocotber 2000.

[Ste02] Stewart R., **Stream Control Transmission Protocol Applicability (SCTP) Implementers Guide**, Internet Draft <draft-ietf-tsvwg-sctpimpguide-06.txt>,May 2002.

[Ste97] Stevens W., **TCP Slow Start, Congestion Avoidance, Fast Recovery Algorithms,** Internet RFC 2001, January 1997

[Sto00] Stone J., Stewart R., Otis D., **Stream Control Transmission Protocol (SCTP) Checksum Change**, RFC 3309 , September 2000.

[Str02] **Stream Control Protocol (SCTP),** International Engineering Consortium Tutorial disponível em: http://www.iec.org/online/tutorials/sctp/. Download em Maio de 2002.

[Tan96] Tanembaum, A. S**., Computer Networks,** Prentice Hall**,** 3rd ed., 1996.

[Tei98] Teitelman, B. & Hanss, T., **QoS Requirements for Internet2**, Internet2 QoS Work GroupDraft, Abril 1998.

[Tho97] Thomson, K., Miller, G. J. & Wilder, R., **Wide-Area Internet Traffic Patterns and Characteristics,** IEEE Network, November 1997.

[Vin] **VINT Project** (*Virtual InterNetwork Testbed*) Disponível na Internet em: http:// www.isi.edu/nsnam/vin.

[Whi97] White, P., **RSVP and Integrated Services in the Internet**: A Tutorial, IEEE Communications, págs. 100-106, May 1997.

[Xia99] Xiao, X.; L. M., **Internet QoS: A Big Picture,** IEEE Network, March/April 1999.

[Ziv99] Zivani, Artur**, Voz sobre Serviços Diferenciados na Internet,** Tese de Mestrado, Setembro 1999.

Anexos

Mostra-se o código utilizado nas simulações. Alguns parâmetros foram modificados durante os experimentos para os diferentes cenários utilizados

```
#-----------------------------------------------------------------
#                                                                 #
# Autor: Constantino Augusto Dias Neto                            #
#                                                                 #
# E-mail: cadn@cin.ufpe.br                                        #
#                                                                 #
# Data: Jun - 2002                                                #
#-----------------------------------------------------------------


#-- Referencia o arquivo tcl onde há proc.-------------------------------

source /home/ns_cadn/ns-allinone-2.1b8/ns-2.1b8/tcl/lib/ns-scripts-cadn.tcl

#-- Cria a variável para informar o PktStatus o tempo final do fluxo ou da
simulação --

set end 100.0

#--  tracing os pacotes  SCTP ------------------------------------------

Trace set show_sctphdr_ 1

#-- Cria o objeto Simulator que permite a realização da simulação -------

set ns [new Simulator]

#-- Define cores para os fluxos de dados --------------------------------

$ns color 0 Blue
$ns color 1 Green
$ns color 2 orange
$ns color 3 purple
$ns color 4 white
$ns color 5 pink
$ns color 6 black
$ns color 7 Red
$ns color 7 Yellow
$ns color 8 Brown
$ns color 9 Brown

#-- Cria os parâmetros para o DiffServ ----------------------------------

#BE 60%
set cir_BE  12000000
set cbs_BE  10000

#AF 40%
set cir_AF     800000
set cbs_AF      10000

set packetSize 1500
set packetSizeVoice 64

set InitialTime 0.1
set FinalTime  100

#Núumero de fontes
```

```
set num_ftp 1
set num_cbr 10
set num_exp_n1_n8 $num_cbr
set num_ftp_n2_n6 $num_ftp
set num_ftp_n3_n7 $num_ftp
set num_telnet_n4_n7 $num_ftp
set num_cbr_n4_n8 $num_cbr
set num_ftp_n3_n6 $num_ftp


#-- Arquivo nam trace ------------------------------------------------
-

#set nf [open sctp.nam w]
#$ns namtrace-all $nf

#--- Abre o arquivo trace que armazenar todos os fluxos da simulação ------

set allchan [open /tmp/cenario1_Qos/all.tr w]
$ns trace-all $allchan

#-- Arquivos de saída para o cálculo da vazão e utilização do link --------
--

set f0 [open vazao_tcp.tr w]
set f2 [open vazao_sctp.tr w]

set lUtil1 [open util_link_r0_r1.tr w]
set lUtil  [open util_link_r1_r2.tr w]

set lUtil_pdepartures  [open r0_r1_pkt_cheg.txt w]
set lUtil_pdepartures1 [open r1_r2_pkt_cheg.txt w]

set lUtil_pdrops  [open r0_r1_drops.txt w]
set lUtil_pdrops1 [open r1_r2_drops.txt w]

#-- Cria os nodos,roteadores e define os label ----------------------------

set n0_core [$ns node] ;   $n0_core label "n0"        ; $n0_core color Red
set n0_if1 [$ns node]  ;   $n0_if1 label "n0_if1"     ; $n0_if1 color Red
set n0_if2 [$ns node]  ;   $n0_if2 label "n0_if2"     ; $n0_if2 color Red
set n1 [$ns node]      ;   $n1 label "n1"

set n2 [$ns node]      ;   $n2 label "n2"
set n3 [$ns node]      ;   $n3 label "n3"
set n4 [$ns node]      ;   $n4 label "n4"
set n5 [$ns node]      ;   $n5 label "n5"
set n6 [$ns node]      ;   $n6 label "n6"
set n7 [$ns node]      ;   $n7 label "n7"
set n8 [$ns node]      ;   $n8 label "n8"
set n9_core [$ns node] ;   $n9_core label "n9"        ; $n9_core color Red
set n9_if1 [$ns node]  ;   $n9_if1 label  "n9_if1"    ; $n9_if1 color Red
set n9_if2 [$ns node]  ;   $n9_if2 label  "n9_if2"    ; $n9_if2 color Red
set r0 [$ns node]      ;   $r0 label "r0"             ; $r0 color Blue
set r1 [$ns node]      ;   $r1 label "r1"             ; $r1 color Blue
set r2 [$ns node]      ;   $r2 label "r2"             ; $r2 color Blue

#-- Adiciona as interface ao endpoint multihome ------------------------

$ns multihome-add-interface $n0_core $n0_if1
$ns multihome-add-interface $n0_core $n0_if2
```

```
$ns multihome-add-interface $n9_core $n9_if1
$ns multihome-add-interface $n9_core $n9_if2

#--- Conecta os nodos ----------------------------------------------------

set alg [lindex $argv 0 ]

if {$alg == "NQoS"} {

    $ns duplex-link $n0_if1 $r0 5Mb 1ms DropTail
    $ns duplex-link $n0_if2 $r0 5Mb 1ms DropTail
    $ns duplex-link $n1     $r0 5Mb 1ms DropTail
    $ns duplex-link $n2     $r0 5Mb 1ms DropTail
    $ns duplex-link $n3     $r0 5Mb 1ms DropTail
    $ns duplex-link $n4     $r0 5Mb 1ms DropTail
    $ns duplex-link $n5     $r2 5Mb 1ms DropTail
    $ns duplex-link $n6     $r2 5Mb 1ms DropTail
    $ns duplex-link $n7     $r2 5Mb 1ms DropTail
    $ns duplex-link $n8     $r2 5Mb 1ms DropTail
    $ns duplex-link $n9_if1 $r2 5Mb 1ms DropTail
    $ns duplex-link $n9_if2 $r2 5Mb 1ms DropTail
    $ns duplex-link $r1     $r0 2Mb 20ms DropTail
    $ns duplex-link $r1     $r2 2Mb 20ms DropTail

 }

if {$alg == "QoS"} {

    $ns duplex-link $n0_if1 $r0 5Mb 1ms DropTail
    $ns duplex-link $n0_if2 $r0 5Mb 1ms DropTail
    $ns duplex-link $n1     $r0 5Mb 1ms DropTail
    $ns duplex-link $n2     $r0 5Mb 1ms DropTail
    $ns duplex-link $n3     $r0 5Mb 1ms DropTail
    $ns duplex-link $n4     $r0 5Mb 1ms DropTail

    $ns simplex-link $r0 $r1 2Mb 20ms dsRED/edge
    $ns simplex-link $r1 $r0 2Mb 20ms dsRED/core

    $ns simplex-link $r2 $r1 2Mb 20ms dsRED/edge
    $ns simplex-link $r1 $r2 2Mb 20ms dsRED/core


    $ns duplex-link $r2  $n5      5Mb 1ms DropTail
    $ns duplex-link $r2  $n6      5Mb 1ms DropTail
    $ns duplex-link $r2  $n7      5Mb 1ms DropTail
    $ns duplex-link $r2  $n8      5Mb 1ms DropTail
    $ns duplex-link $r2  $n9_if1  5Mb 1ms DropTail
    $ns duplex-link $r2  $n9_if2  5Mb 1ms DropTail
 }



#-- Define as orientaçcoes dos nodos ------------------------------

$ns duplex-link-op $n1 $r0 orient up
$ns duplex-link-op $n2 $r0 orient up-right
$ns duplex-link-op $n3 $r0 orient down
$ns duplex-link-op $n4 $r0 orient down-right
$ns duplex-link-op $n5 $r2 orient down
$ns duplex-link-op $n6 $r2 orient down-left
```

```
$ns duplex-link-op $n7 $r2 orient up
$ns duplex-link-op $n8 $r2 orient up-left

$ns duplex-link-op $r0 $r1 orient right
$ns duplex-link-op $r1 $r2 orient right

#-- Monitora o angulo de visaão (no Nam) das filas nos enlace --------

$ns duplex-link-op $r0 $r1 queuePos 0.5
$ns duplex-link-op $r1 $r2 queuePos 0.5

if {$alg == "QoS"} {

#--- edge queue

   set qr0r1 [[$ns link $r0 $r1] queue]
   set qr2r1 [[$ns link $r2 $r1] queue]

#--- core Queue

   set qr1r0 [[$ns link $r1 $r0] queue]
   set qr1r2 [[$ns link $r1 $r2] queue]

## Configura os parâmetros DS RED dos roteadores ----------------------

  $qr0r1 meanPktSize $packetSize
  $qr0r1 set numQueues_ 1
  $qr0r1 setNumPrec 2

  $qr0r1 addPolicyEntry [$n1 id] [$n5 id] TokenBucket 10 $cir_AF $cbs_AF
  $qr0r1 addPolicyEntry [$n1 id] [$n8 id] TokenBucket 10 $cir_AF $cbs_AF
  $qr0r1 addPolicyEntry [$n2 id] [$n5 id] TokenBucket 10 $cir_AF $cbs_AF
  $qr0r1 addPolicyEntry [$n2 id] [$n6 id] TokenBucket 0  $cir_BE $cbs_BE
  $qr0r1 addPolicyEntry [$n3 id] [$n6 id] TokenBucket 10 $cir_AF $cbs_AF
  $qr0r1 addPolicyEntry [$n3 id] [$n7 id] TokenBucket 0  $cir_BE $cbs_BE
  $qr0r1 addPolicyEntry [$n4 id] [$n7 id] TokenBucket 0  $cir_BE $cbs_BE
  $qr0r1 addPolicyEntry [$n4 id] [$n8 id] TokenBucket 0  $cir_BE $cbs_BE

  $qr0r1 addPolicyEntry [$n0_if1 id] [$n9_if1 id] TokenBucket 0  $cir_BE
$cbs_BE
  $qr0r1 addPolicyEntry [$n0_if1 id] [$n9_if2 id] TokenBucket 10  $cir_AF
cbs_AF
  $qr0r1 addPolicyEntry [$n0_if2 id] [$n9_if1 id] TokenBucket 0   $cir_BE
cbs_BE
  $qr0r1 addPolicyEntry [$n0_if2 id] [$n9_if2 id] TokenBucket 10  $cir_AF
cbs_AF

  $qr0r1 addPolicerEntry TokenBucket 10 12
  $qr0r1 addPolicerEntry TokenBucket 0 0

  $qr0r1 addPHBEntry 10 0 0
  $qr0r1 addPHBEntry 12 0 1
  $qr0r1 addPHBEntry 0  0 0
  $qr0r1 configQ 0 0 20 40 0.02
  $qr0r1 configQ 0 1 10 20 0.10
  #$qr0r1 configQ 1 0 60 110 0.30

  #$qr0r1 configQ 0 0 15 35 0.06
  #$qr0r1 configQ 0 1 5  15 0.30
  #$qr0r1 configQ 0 1 45 90 0.2
```

```
 ## Configura os parâmetros de DS RED de r2 para r1 ----------------------
-----

  $qr2r1 meanPktSize $packetSize
  $qr2r1 set numQueues_ 1
  $qr2r1 setNumPrec 2

  $qr2r1 addPolicyEntry [$n5 id] [$n1 id]  TokenBucket 10  $cir_AF $cbs_AF
  $qr2r1 addPolicyEntry [$n8 id] [$n1 id]  TokenBucket 10  $cir_AF $cbs_AF
  $qr2r1 addPolicyEntry [$n5 id] [$n2 id]  TokenBucket 10  $cir_AF $cbs_AF
  $qr2r1 addPolicyEntry [$n6 id] [$n2 id]  TokenBucket 0   $cir_BE $cbs_BE
  $qr2r1 addPolicyEntry [$n6 id] [$n3 id]  TokenBucket 10  $cir_AF $cbs_AF
  $qr2r1 addPolicyEntry [$n7 id] [$n3 id]  TokenBucket 0   $cir_BE $cbs_BE
  $qr2r1 addPolicyEntry [$n7 id] [$n4 id]  TokenBucket 0   $cir_BE $cbs_BE
  $qr2r1 addPolicyEntry [$n8 id] [$n4 id]  TokenBucket 0   $cir_BE $cbs_BE

  $qr2r1 addPolicyEntry [$n9_if1 id] [$n0_if1 id]  TokenBucket 0  $cir_BE
cbs_BE
  $qr2r1 addPolicyEntry [$n9_if2 id] [$n0_if1 id]  TokenBucket 10 $cir_AF
cbs_AF
  $qr2r1 addPolicyEntry [$n9_if1 id] [$n0_if2 id]  TokenBucket 0  $cir_BE
cbs_BE
  $qr2r1 addPolicyEntry [$n9_if2 id] [$n0_if2 id]  TokenBucket 10 $cir_AF
cbs_AF

  $qr2r1 addPolicerEntry TokenBucket 10 12
  $qr2r1 addPolicerEntry TokenBucket 0  0

  $qr2r1 addPHBEntry 10 0 0
  $qr2r1 addPHBEntry 12 0 1
  $qr2r1 addPHBEntry 0  0 0

  $qr2r1 configQ 0 0 20 40 0.02
  $qr2r1 configQ 0 1 10 20 0.10

  ## Set DS RED parameters from r1 to r0:
  $qr1r2 setSchedularMode PRI
  $qr1r0 meanPktSize $packetSize
  $qr1r0 set numQueues_ 1
  $qr1r0 setNumPrec 2

  $qr1r0 addPHBEntry 10 0 0
  $qr1r0 addPHBEntry 12 0 1
  $qr1r0 addPHBEntry 0  1 0

  $qr1r0 configQ 0 0 20 40 0.02
  $qr1r0 configQ 0 1 10 20 0.10


  # Set DS RED parameters from r1 to r2:
  $qr1r2 meanPktSize $packetSize
  $qr1r2 set numQueues_ 2
  $qr1r2 setNumPrec 2

  $qr1r2 addPHBEntry 10 0 0
  $qr1r2 addPHBEntry 12 0 1
  $qr1r2 addPHBEntry 0  1 0

  $qr1r2 configQ 0 0 20 40 0.02
  $qr1r2 configQ 0 1 10 20 0.10
  $qr1r2 configQ 1 0 60 110 0.30
```

```
 }

#-- Geraçãcao das Fontes de Traáfego # --------

#--Do n0_core para o n9_core --

set sctp0 [new Agent/SCTP]
$ns multihome-attach-agent $n0_core $sctp0
$sctp0 set debugMask_ -1
$sctp0 set debugFileIndex_ 0
$sctp0 set mtu_ 1500
$sctp0 set dataChunkSize_ 1448
$sctp0 set numOutStreams_ 2
$sctp0 set unordered_ 1
$sctp0 set oneHeartbeatTimer_ 0
$sctp0 set fid_ 0

set ftp0 [new Application/FTP]
$ftp0 attach-agent $sctp0
$ftp0 set packetSize_ $packetSize

set sctp1 [new Agent/SCTP]
$ns multihome-attach-agent $n9_core $sctp1
$sctp1 set debugMask_ -1
$sctp1 set debugFileIndex_ 1
$sctp1 set mtu_ 1500
#$sctp1 set initialRwnd_ 131072


#-- Do n1 para o n5 --

set tcp0 [new Agent/TCP]
$tcp0 set packetSize_ $packetSize
$ns attach-agent $n1 $tcp0

set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp0
$ftp2 set packetSize_ $packetSize
$tcp0 set fid_ 1

set tanque_tcp1 [new Agent/TCPSink]
$ns attach-agent $n5 $tanque_tcp1

#-- Do n1 para o n8 --

for {set i 1} {$i<= $num_exp_n1_n8} {incr i} {

    set udp2($i) [new Agent/UDP]
    $ns attach-agent $n1 $udp2($i)

    set voz0($i) [new Application/Traffic/Exponential]
    $voz0($i)  attach-agent $udp2($i)
    $voz0($i) set packetSize_ $packetSizeVoice
    $voz0($i) set burst_time_ 1004ms
    $voz0($i) set idle_time_ 1587ms
    $voz0($i) set rate_ 6.3k
    $udp2($i) set fid_ 2

    set null1($i) [new Agent/Null]
    $ns attach-agent $n8  $null1($i)
```

```
    $ns connect $udp2($i) $null1($i)
}

#-- Do n2 para o n5 --

set sctp2 [new Agent/SCTP]
#$sctp2 set numOutStreams_ 2
#$sctp2 set unordered_ 1
$ns attach-agent $n2 $sctp2

set ftp1 [new Application/FTP]
$ftp1 attach-agent $sctp2
$ftp1 set packetSize_ $packetSize
$sctp2 set fid_ 3

set sctp3 [new Agent/SCTP]
$ns attach-agent $n5 $sctp3

#-- Do n2 para o n6 ------------

for {set i 1} {$i<= $num_ftp_n2_n6} {incr i} {

    set tcp4($i) [new Agent/TCP]
    $ns attach-agent $n2 $tcp4($i)

    set tcp5($i) [new Agent/TCPSink]
    $ns attach-agent $n6 $tcp5($i)
    $ns connect $tcp4($i) $tcp5($i)

    $tcp4($i) set packetSize_ $packetSize
    $tcp4($i) set fid_ 4
    set ftp6($i) [$tcp4($i) attach-source FTP]

    set ftp6($i) [new Application/FTP]
    $ftp6($i) attach-agent $tcp4($i)

}

#-- Do n3 para o n6 --

for {set i 1} {$i <= $num_ftp_n3_n6} {incr i} {

     set sctp10($i) [new Agent/SCTP]
     $ns attach-agent $n3 $sctp10($i)

     set ftp7($i) [new Application/FTP]
     $ftp7($i) attach-agent $sctp10($i)
     $ftp7($i) set packetSize_ $packetSize
     $sctp10($i) set fid_ 5

     set sctp11($i) [new Agent/SCTP]
     $ns attach-agent $n6 $sctp11($i)
     $ns connect $sctp10($i) $sctp11($i)
}

#-- Do n3 para o n7 --

for {set i 1} {$i <= $num_ftp_n3_n7} {incr i} {

    set sctp4($i) [new Agent/SCTP]
    $ns attach-agent $n3 $sctp4($i)
```

```
    set ftp3($i) [new Application/FTP]
    $ftp3($i) attach-agent $sctp4($i)
    $ftp3($i) set packetSize_ $packetSize
    $sctp4($i) set fid_ 6

    set sctp5($i) [new Agent/SCTP]
    $ns attach-agent $n7 $sctp5($i)
    $ns connect $sctp4($i) $sctp5($i)
}

#-- Do n4 para o n7 --

for {set i 1} {$i <= $num_telnet_n4_n7} {incr i} {
    set tcp6($i) [new Agent/TCP]
    $ns attach-agent $n4 $tcp6($i)

    set telnet0($i) [new Application/Telnet]
    $telnet0($i) attach-agent $tcp6($i)
    $telnet0($i) set packetSize_ $packetSize
    $tcp6($i) set fid_ 7

    set tcp7($i) [new Agent/TCPSink]
    $ns attach-agent $n7 $tcp7($i)
    $ns connect $tcp6($i) $tcp7($i)
}

#-- Do nodo n4 para o n8 --

for {set i 1} {$i <= $num_cbr_n4_n8} {incr i} {

    set udp1($i) [new Agent/UDP]
    $ns attach-agent $n4 $udp1($i)

    set cbr1($i) [new Application/Traffic/CBR]
    $cbr1($i) attach-agent $udp1($i)
    $cbr1($i) set packetSize_ 100
    $cbr1($i) set rate_ 64k
    $udp1($i) set class_ 8

    set tanque_udp1($i) [new Agent/LossMonitor]
    $ns attach-agent $n8 $tanque_udp1($i)
    $ns connect $udp1($i) $tanque_udp1($i)

}

#-- Procedimento de finalização da simulacação ----

proc finish {} {
    global ns nf allchan

    set PERL "/usr/bin/perl"
    set USERHOME "/home/cadn/ns/ns_cadn"
    set NSHOME "$USERHOME/ns-allinone-2.1b8"
    set XGRAPH "$NSHOME/bin/xgraph"
    set SETFID "$NSHOME/ns-2.1b8/bin/set_flow_id"
    set RAW2XG_SCTP "$NSHOME/ns-2.1b8/bin/raw2xg-sctp"
    set GETRC "$NSHOME/ns-2.1b8/bin/getrc"

    $ns flush-trace
    #close $nf
```

```
    close $allchan

    #exec nam sctp.nam &
    exit 0
    puts "Simulacao concluida"
}

#-- Procedimento para calcular vazão (para geração dos gráficos) ---------

proc record {} {

      global f0 f1 f2  tanque_tcp1 tanque_sctp

      #Get an instance of the simulator
        set ns [Simulator instance]

      #Set the time after which the procedure should be called again
      set time 0.5

      #How many bytes have been received by the traffic sinks?
          set bw0 [$tanque_tcp1 set bytes_]
          set bw2 [$sctp7 set bytes_]

      #Get the current time
          set now [$ns now]

      #Calculate the bandwidth (in KBits/s) and write it to the files
           puts $f0 "$now [expr $bw0/$time*8/1000]"
           puts $f1 "$now [expr $bw2/$time*8/1000]"

      #Reset the bytes_ values on the traffic sinks
          $tanque_tcp1 set bytes_ 0
          $tanque_sctp set bytes_ 0

      #Re-schedule the procedure
         $ns at [expr $now+$time] "record"
}

#----- Procedimento para medir a utilização do link ---------------------

set interval 0.5

set qmon_  [$ns monitor-queue $r0 $r1 [$ns get-ns-traceall] $interval]
set qmon1_ [$ns monitor-queue $r1 $r2 [$ns get-ns-traceall] $interval]

proc utilization_link { queueMon_ file } {

        $queueMon_ instvar bdepartures_ pdrops_ parrivals_

      #Get an instance of the simulator
        set ns [Simulator instance]

        #Set the time after which the procedure should be called again
        set time 0.5

        #Calculate a utilization link Kbits/s
        set utlzn [expr $bdepartures_ / $time * 8.0/1000 ]

        #Get the current time
        set now [$ns now]
```

```
        puts $file "$now\t$utlzn"

      #Reset the bytes_ values on the traffic sinks
        $queueMon_ set bdepartures_ 0

        #Re-schedule the procedure
        $ns at [expr $now+$time] "utilization_link $queueMon_ $file"

 }

#------ Procedimento para medir  a qtd de pacotes chegados -------------

set interval 10.0

set qtd_loss  [$ns monitor-queue $r0 $r1 [$ns get-ns-traceall] $interval]
set qtd_loss1 [$ns monitor-queue $r1 $r2 [$ns get-ns-traceall] $interval]

proc utilization_link_loss { queueMon_ file } {

        $queueMon_ instvar  pdepartures_

      #Get an instance of the simulator
       set ns [Simulator instance]

      #Set the time after which the procedure should be called again
       set time 10.0

      #Calculate a utilization link Kbits/s
       set utlzn_pdepartures [expr $pdepartures_ ]

     #Get the current time
      set now [$ns now]

     puts $file "$now\t$utlzn_pdepartures"

      #Reset the bytes_ values on the traffic sinks
      $queueMon_ set pdepartures_ 0

      #Re-schedule the procedure
      $ns at [expr $now+$time] "utilization_link_loss $queueMon_ $file"

}

#-- Procedimento para medir a qtd de pacotes perdidos ------------


set interval 10.0

set qtd_loss  [$ns monitor-queue $r0 $r1 [$ns get-ns-traceall] $interval]
set qtd_loss1 [$ns monitor-queue $r1 $r2 [$ns get-ns-traceall] $interval]

proc qtd_pkt_loss { queueMon_ file } {

        $queueMon_ instvar  pdrops_

      #Get an instance of the simulator
        set ns [Simulator instance]

        #Set the time after which the procedure should be called again
        set time 10.0
```

```
        #Calculate a utilization link Kbits/s
        set utlzn_pdrops [expr $pdrops_ ]

        #Get the current time
        set now [$ns now]

        puts $file "$now\t$utlzn_pdrops"

      #Reset the bytes_ values on the traffic sinks
        $queueMon_ set pdrops_ 0

        #Re-schedule the procedure
        $ns at [expr $now+$time] "qtd_pkt_loss $queueMon_ $file"

}#--- Componente de contabilização de atraso e jitter PktStats ----

#==  Calcula o atraso e o jitter do n0_core ao n9_core utilizando a
interface =

 newPktStats $r2 $n9_if2 $sctp0 $sctp1 sctp_n9if2.out

#== Calcula o atraso e o jitter do n1 ao n5 utilizando o agente TCP
========================

  newPktStats $r2 $n5 $tcp0 $tanque_tcp1 tcp_n1_n5.out

#== Calcula o atraso e o jitter do n2 ao n5 utilizando o agente SCTP =

  newPktStats $r2 $n5 $sctp2 $sctp3 sctp_n2_n5.out


#-- Conecta os agentes ----

$ns connect $sctp0 $sctp1
$ns connect $tcp0 $tanque_tcp1
$ns connect $sctp2 $sctp3

#---- Configura destino primário antes iniciar a associação ------------

$sctp0 set-primary-destination $n9_if1

#-- Faz a mudança da interface de origem do endpoint Multi-Homing ---

$ns at 50.0"$sctp0 force-source $n0_if2"

#-- Sequencia da simulaçcao (inicia e termina as fontes de tráfego) ---

$ns at 0.0  "record"

$ns at 0.0  "utilization_link $qmon_ $lUtil"
$ns at 0.0  "utilization_link $qmon1_ $lUtil1"

$ns at 0.0  "utilization_link_loss $qtd_loss  $lUtil_pdepartures"
$ns at 0.0  "utilization_link_loss $qtd_loss1 $lUtil_pdepartures1"

$ns at 0.0  "qtd_pkt_loss $qmon_ $lUtil_pdrops"
$ns at 0.0  "qtd_pkt_loss $qmon1_ $lUtil_pdrops1"

for {set i 1 } {$i<= $num_exp_n1_n8} {incr i} {
    $ns at $InitialTime  "$voz0($i) start"
    $ns at $FinalTime    "$voz0($i) stop"
```

```
}

for {set i 1 } {$i<= $num_ftp_n2_n6} {incr i} {
    $ns at $InitialTime  "$ftp6($i) start"
    $ns at $FinalTime    "$ftp6($i) stop"
}

for {set i 1 } {$i<= $num_ftp_n3_n6} {incr i} {
    $ns at $InitialTime "$ftp7($i) start"
    $ns at $FinalTime   "$ftp7($i) stop"
}

for {set i 1} {$i<= $num_ftp_n3_n7} {incr i} {
    $ns at $InitialTime "$ftp3($i) start"
    $ns at $FinalTime   "$ftp3($i) stop"
}

for {set i 1} {$i<= $num_telnet_n4_n7} {incr i} {
    $ns at $InitialTime "$telnet0($i) start"
    $ns at $FinalTime   "$telnet0($i) stop"
}

for {set i 1} {$i<= $num_cbr_n4_n8} {incr i} {
    $ns at $InitialTime "$cbr1($i) start"
    $ns at $FinalTime "$cbr1($i) stop"
}

if {$alg == "QoS"} {

    $qr0r1 printPolicyTable
    $qr0r1 printPolicerTable
    $ns at [expr 1 * $FinalTime / 4] "$qr1r2 printStats"
    $ns at [expr 2 * $FinalTime / 2] "$qr1r2 printStats"
}

$ns at $InitialTime "$ftp0 start"
$ns at $InitialTime "$ftp1 start"
$ns at $InitialTime "$ftp2 start"

$ns at $FinalTime  "$ftp0 stop"
$ns at $FinalTime  "$ftp1 stop"
$ns at $FinalTime  "$ftp2 stop"

$ns at [expr $FinalTime + 1.0] "finish"

#--- Executa -------------

$ns run
```