UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Elvis Pfützenreuter

Aplicabilidade e desempenho do protocolo de transporte SCTP

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Prof. Luis Fernando Friedrich

Orientador

Florianópolis, dezembro de 2004

Aplicabilidade e desempenho do protocolo de transporte SCTP

Elvis Pfützenreuter

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Raul Sidnei Wazlawick
Coordenador do Curso

Banca Examinadora

_____

Luis Fernando Friedrich
Orientador

_____

Rômulo Silva de Oliveira

_____

Mário Antonio Ribeiro Dantas

_____

Vitório Bruno Mazzola

SUMÁRIO

Lista de siglas e abreviaturas **6**

Resumo **11**

1 Introdução **13**

1.1 Histórico 13

1.2 Objetivo geral 14

1.3 Objetivos específicos 15

1.4 Metodologia empregada 15

2 História do protocolo SCTP **16**

2.1 Antecedentes 16

2.2 Descrição geral da rede de sinalização telefônica SS7 17

2.3 Integração entre redes SS7 e IP 20

2.4 A consagração do SCTP como protocolo de transporte 23

2.5 Outros protocolos de transporte em desenvolvimento 24

2.6 Conclusões 25

3 Apresentação do SCTP **27**

3.1 Associações e fluxos *(streams)* 27

3.2 Mensagens indivisíveis 28

3.3 Multicaminhos *(multi-homing)* 29

3.4 Extensões propostas ao SCTP 31

3.5 Implementações 36

4 Aplicabilidade do SCTP **37**

4.1 SCTP em substituição a TCP 37

4.2 SCTP em substituição a UDP 41

4.3 Novas possibilidades de uso do TCP/IP e da Internet 42

5 Comparação de desempenho com TCP 44

5.1 Objetivos . . . 44

5.2 Aplicativos construídos para o teste . . . 45

5.3 Sistema operacional e implementação do SCTP . . . 47

5.4 Ferramentas de simulação de rede . . . 48

5.5 Computadores utilizados nos testes . . . 50

5.6 Cenários de rede . . . 50

5.7 Resultados . . . 53

5.7.1 Interface de *loopback* . . . 53

5.7.2 Rede 100Mbps . . . 57

5.7.3 Rede 100Mbps com perda de pacotes . . . 59

5.7.4 Rede 10Mbps com latência . . . 61

5.7.5 10Mbps com latência e perda de pacotes . . . 63

5.7.6 Rede 1Mbps, latência e duplicação de pacotes . . . 65

5.7.7 Rede 1Mbps, latência, duplicação e perda de pacotes . . . 65

5.7.8 Rede 11Mbps *wireless* . . . 65

5.7.9 Dois clientes 100Mbps . . . 66

5.7.10 Multicaminhos 10Mbps e 11Mbps . . . 67

5.8 Validade dos testes . . . 68

5.9 Conclusões . . . 68

6 Comparação de desempenho com TCP e UDP em aplicações reais 70

6.1 Objetivos . . . 70

6.2 Adaptação dos protocolos de aplicação ao SCTP . . . 71

6.2.1 HTTP . . . 71

6.2.2 SMB . . . 74

6.2.3 RTP . . . 74

6.3 Aplicativos adaptados para os testes . . . 75

6.3.1 HTTP . . . 75

6.3.2 SMB . . . 76

6.3.3 RTP . . . 77

6.3.4 Outros aplicativos . . . 77

6.4 Resultados . . . 78

6.4.1 HTTP . . . 78

6.4.2 SMB . . . 80

6.4.3 RTP . . . 81

7 Conclusões e trabalhos futuros 86

ANEXO 1: Características de segurança do SCTP 89

ANEXO 2: Detalhes de funcionamento do protocolo SCTP 97

ANEXO 3: Extensões da interface BSD Sockets para suporte ao SCTP 104

Referências Bibliográficas 116

Lista de Tabelas

5.1 Vazão em kbps, conexões *loopback* 53

5.2 Latência em μs, conexões *loopback* 55

5.3 Vazão e latência em conexões *loopback*, em função do número de fluxos . 56

5.4 Vazão em kbps, rede 100Mbps 57

5.5 Latência em μs, rede 100Mbps 58

5.6 Vazão (em kbps), rede 100Mbps com perda de pacotes 59

5.7 Latência em μs, rede 100Mbps com perda de pacotes 60

5.8 Vazão em kbps, rede 10Mbps com latência de rede 62

5.9 Latência de transação em μs, rede 10Mbps com latência de rede 63

5.10 Vazão em kbps, rede 10Mbps com latência de rede e perda de pacotes . . 63

5.11 Latência de transação em μs, rede 10Mbps com latência de rede e perda de pacotes 64

5.12 Vazão e latência de transação, rede 1Mbps com de latência de rede e duplicação de pacotes 65

5.13 Vazão e latência de transação, rede 1Mbps com latência de rede, perda e duplicação de pacotes 65

5.14 Vazão e latência, rede 11Mbps *wireless* 66

5.15 Vazão e latência, rede 100Mbps – um servidor e dois clientes 66

5.16 Vazão em kbps, conexões multicaminhos 10Mbps e 11Mbps 67

5.17 Latência em μs, conexões multicaminhos 10Mbps e 11Mbps 67

6.1 Vazão HTTP em Kbytes/s, rede de 100Mbps 79

6.2 Vazão HTTP, rede de 1Mbps com perdas e latência 80

6.3 Vazão e latência SMB (valores em segundos, vide seção 6.4.2) 81

Lista de Figuras

2.1 Pilha de protocolos SS7 17

2.2 Pilha de protocolos com camada de adaptação M3UA 22

2.3 Pilhas de protocolos com camadas de adaptação M2UA e M2PA 23

5.1 Exemplo de *script* para simulação de rede com latência e perda 49

5.2 Vazão em conexões *loopback* 54

5.3 Latência em conexões *loopback* 55

5.4 Vazão, rede 100Mbps 57

5.5 Latência, rede 100Mbps 58

5.6 Vazão em kbps, rede 100Mbps com perda de pacotes 59

5.7 Latência, rede 100Mbps com perda de pacotes 60

5.8 Vazão em kbps, rede 10Mbps com latência de rede 61

5.9 Latência de transação, rede 10Mbps com latência de rede 62

5.10 Vazão, rede 10Mbps com latência de rede e perda de pacotes 63

5.11 Latência de transação, rede 10Mbps com latência de rede e perda de pacotes 64

6.1 Vazão HTTP, rede de 100Mbps 78

6.2 Vazão HTTP, rede de 1Mbps com perdas e latência 79

6.3 Desempenho RFC 2250 sobre UDP, SCTP e PR–SCTP 82

6.4 Desempenho RFC 3119 sobre UDP, SCTP e PR–SCTP 83

6.5 Desempenho do protocolo FEC sobre UDP, SCTP e PR–SCTP 84

Lista de siglas e abreviaturas

Ad-hoc: Em redes *wireless* 802.11, rede sem nenhum equipamento ativo (*access point*).

ADSL: *Asymetrical Digital Subscriber Line*

AH: *Authentication Header*. Vide IPSEC.

API: *Application Programming Interface.* Conjunto de funções que um aplicativo deve usar para comunicar-se com uma biblioteca ou com o sistema operacional.

AS: *Autonomous System.* Conjunto de redes tratado como um bloco único e opaco pelo roteamento global da Internet.

ATM: *Asyncronous Transfer Mode.*

BGP: *Border Gateway Protocol.* Protocolo de roteamento dinânico utilizado entre ASs.

Blind spoof: Tipo de ataque contra o protocolo de transporte TCP.

BSD Unix: Sistema operacional compatível com UNIX.

BSD/Sockets: API do BSD Unix para acesso aos recursos de rede.

CCS: *Common Channel Signaling.* Tecnologia onde o mesmo canal digital é utilizado para conteúdo (voz) e dados de sinalização.

CIDR: *Classless InterDomain Routing.* Estabelece que redes topologicamente próximas tenham os mesmos prefixos de endereçamento.

CIFS: *Common Internet File System.* Vide SMB.

CODA: Sistema de arquivos distribuído que suporta desconexão temporária de rede entre cliente e servidor.

CRC: *Cyclic Redundancy Check.* algoritmo de somatório de verificação baseado em divisão de polinômios.

CRC-32: versão de CRC que gera somatório de 32 *bits*.

CRC-32c: versão do CRC que gera somatótio de 32 *bits*, com polinômio divisor diferente do CRC-32.

DCCP: *Datagram Congestion Control Protocol*.

Denial-of-Service (DoS) atack: Ataque de negação de serviço, que impede os usuários legítimos de usar determinado serviço.

DNS: *Domain Name System*.

ECN: *Explicit Congestion Notification.* Notificação explícita de congestionamento, feita pelo roteador intermediário através de um *bit* do cabeçalho de rede IP, e participada ao emissor através do protocolo de transporte.

ESP: *Encapsulating Security Payload*. Vide IPSEC.

FEC: *Forward Error Correction.* Algoritmo de somatório de verificação que, transmitido junto com o dado original, permite detectar e corrigir erros no destinatário.

FDDI: *Fiber Distributed Data Interface*.

FTP: *File Transfer Protocol*. Protocolo para transferência de arquivos.

H.323: Conjunto de protocolos para transmissão comprimida de voz e vídeo, usada por alguns sistemas de VoIP.

Head-of-Line (HOL) Blocking: Problema em alguns protocolos confirmados que causa atraso na entrega dos dados.

HTTP: *HyperText Transfer Protocol*.

IANA: *Internet Assigned Numbers Authority*.

ICMP: *Internet Control Message Protocol*.

IETF: *Internet Engineering Task Force.*

IPSEC: *Internet Protocol SECurity*.

IP: *Internet Protocol.* Protocolo de rede da pilha TCP/IP.

IPTables: Arquitetura de *firewall* do sistema operacional Linux, versão 2.4 em diante.

IPv4: *Internet Protocol version 4.*

IPv6: *Internet Protocol version 4.*

ISDN: *Integrated Services Digital Network.* Padrão de telefonia inteiramente digital.

ISN: *Initial Sequence Number*. TSN inicial de uma conexão TCP ou associação SCTP.

ITU: *International Telecommunication Union.*

ITU-T: *International Telecommunication Union – Telecommunication*. Setor de padrões da ITU.

Kernel: Núcleo do sistema operacional.

Kernel level: Código que roda dentro do *kernel*.

LDAP: *Lightweight Directory Access Protocol*.

LSB: *Least Significant Bit*. *Bit* mais à direita de um número binário.

LK-SCTP: *Linux Kernel SCTP*.

M2UA: *MTP-2 User Adaptation*.

M2PA: *MTP-2 Peer-to-Peer Adaptation*.

M3UA: *MTP-3 User Adaptation*.

MDTP: *Multi-Network Datagram Transport Protocol*.

MSB: *Most Significant Bit*. *Bit* mais à esquerda de um número binário.

MTU: *Maximum Transmission Unit*.

MTP: *Message Transfer Part*. A subpilha inferior da pilha de rede SS7. Subdividida em MTP-1, MTP-2 e MTP-3.

NAT: *Network Address Translation*.

Netem: Disciplina de tráfego de rede do sistema operacional Linux.

NetBIOS: *Network Basic Input/Output Services*. Vide SMB.

NFS: *Network File System*.

NTP: *Network Time Protocol*.

Octeto: *Byte* com invariavelmente 8 *bits*.

OSI: *Open Systems Interconnection*. Pilha de protocolos de rede criada pela ITU-T.

PMTU: *Path Maximum Transmission Unit*.

POSIX: *Portable Open System Interface for UNIX*. Conjunto de padrões que define um sistema vulgarmente conhecido como ”UNIX-compatível”.

PR-SCTP: *Partial Reliability SCTP*. Extensão do SCTP que permite relaxar a confiabilidade de mensagens.

RFC: *Request For Comment.*

RPC: *Remote Procedure Call*.

RSVP: *Resource reSerVation Protocol.*

RTP: *Real Time Protocol.*

RTO: *Retransmission TimeOut.* Tempo que o transmissor espera para receber uma confirmação do receptor. Após esse tempo, o pacote é considerado perdido e retransmitido.

RTCP: *Real-time Control Protocol*.

RTT: *Round-Trip Time*. Tempo decorrido entre a transmissão do pacote e o recebimento da respectiva confirmação. É essencialmente a soma das latências de ida e de volta da rede.

SCCP: *Signaling Conection Control Part.* Camada de aplicação da pilha SS7.

SCTP: *Stream Control Transmission Protocol*.

SFTP: *Secure File Transfer Protocol.*

SG: *Signaling Gateway*. Equipamento que faz papel de *gateway* numa rede SS7.

SIO: *Service Information Octet*. Em redes SS7, um sub-endereço dentro de um nó da rede. Análogo ao número de porta em TCP/IP.

SIGTRAN: Comitê *Signaling Transport*.

SQL: *Structured Query Language*.

SMB: *Service Message Block.* Protocolo de serviço de arquivos e impressoras.

SMS: *Short Message System*.

Soquete: em *BSD/Sockets*, um manipulador de arquivo que corresponde a uma conexão de rede.

Soquete UNIX: método de comunicação interprocessos acessível através da API *BSD/Sockets*.

Spoof: Nome genérico dado a ataques que envolvem falseamento da origem, que mascara o invasor.

SS7: *Signaling System #7*. Pilha de rede de comutação de pacotes usada em telefonia.

SSH: *Secure Shell Protocol*.

SSL: *Secure Sockets Layer*.

SSN: *Stream Sequence Number.*

STL: *Standard Template Library.*

STP: *Signal Transfer Point*. Equipamento que faz o papel de roteador numa rede SS7.

SYN Flood: Ataque de negação de serviço contra o protocolo TCP.

TCB: *Transmission Control Block.*

TCP: *Transmission Control Protocol.*

TCP/IP: *Transmission Control Protocol/Internet Protocol.* Sigla pela qual a pilha de protocolos da Internet é vulgarmente conhecida.

TCPM: No contexto desta dissertação, protocolo de aplicação criado para testes de desempenho.

Timestamp: Marca de tempo.

TLS: vide SSL

TLV: *Type, Length and Value.* Tipo de estrutura de dados de tamanho variável.

TSN: *Transmission Sequence Number*.

UDP: *User Datagram Protocol*.

UNIX: Família de sistemas operacionais.

UNIX socket: vide soquete UNIX.

UNIXM: No contexto desta dissertação, protocolo de aplicação criado para testes de desempenho.

U-SCTP: *Unreliable SCTP.*

WebDAV: *Web Distributed Authoring and Versioning*. Extensão do HTTP para manipulação de arquivos.

VPN: *Virtual Private Network.*

VoIP: *Voice over IP.*

XML: *eXtensible Markup Language*.

XTP: *eXpress Transport Protocol*.

RESUMO

Nas pilhas de protocolos de rede típicas, como TCP/IP, a camada de rede oferece transmissão não confiável de datagramas – um serviço muito primitivo para ser usado diretamente pelas aplicações em geral. É a camada de transporte a responsável por oferecer serviços de rede confiáveis e confortáveis às aplicações.

A pilha TCP/IP oferece tradicionalmente apenas dois protocolos de transporte: TCP e UDP. O UDP implementa apenas o recurso de portas, sem acrescentar confiabilidade à rede. O TCP, por outro lado, oferece um serviço confiável, com conexões ponto-a-ponto e transmissão de *bytes*, abstraindo completamente as características da rede. TCP e UDP são dois extremos; algumas aplicações desejariam usar ao mesmo tempo recursos de ambos os protocolos, e/ou ter controle mais direto sobre alguns aspectos da rede. Tal necessidade surgiu na sinalização telefônica, o que motivou a criação de um novo transporte, o SCTP.

O SCTP *(Stream Control Transmission Protocol)* guarda diversas semelhanças com o TCP, mas também apresenta diversos recursos adicionais interessantes: transmissão de mensagens indivisíveis, multiplos fluxos de mensagens por conexão, variação da confiabilidade das mensagens, bem como melhorias de segurança.

A proposta deste trabalho é apresentar os recursos do SCTP, estudar a aplicabilidade do mesmo a protocolos de aplicação que hoje utilizam TCP ou UDP como transporte, e realizar testes de desempenho sobre uma implementação real do SCTP, com a simulação de diversas condições típicas de rede.

O desenvolvimento do trabalho envolveu uma razoável revisão bibliográfica sobre protocolos em todas as camadas, criação e adaptação de aplicativos para execução dos testes, uso de ferramentas de simulação de rede, escolha de métricas de avaliação de desempenho, e adaptação de protocolos de aplicação ao SCTP.

ABSTRACT

The network layer of typical network stacks e.g. TCP/IP offer best-effort, non-guaranteed datagram transmission service, too primitive for direct use by general applications. The transport layer is in charge of offering a reliable and comfortable network service for the applications.

The TCP/IP stack traditionally offers only two transport protocols: TCP and UDP. UDP just implements port numbers and does not add any reliability to network service. In the other hand, TCP offers a reliable, point-to-point connection-oriented service for byte transmission, insulating completely the application from network characteristics. TCP and UDP offer two extremes of transport service; some applications would like to have resources of both protocols at the same time and/or have a more direct control over the network. Such needs arose in telephony signaling, which motivated the creation of a new transport protocol called SCTP.

SCTP (Stream Control Transmission Protocol) is like TCP in many aspects, but also brings several new and interesting features: transmission of indivisible messages, multiple independent streams of messages per connection, options of partial reliability for some messages, as well as security improvements.

This dissertation will present the new features of SCTP, study the applicability of SCTP for application protocols that use TCP or UDP as transport service today, and make performance tests upon a real SCTP implementation in several typical network scenarios.

Dissertation development involved a reasonable bibliographic revision about protocols at all layers of a network, creation and adaptation of applications for testing, use of network simulation tools, selection of metrics for performance evaluation, and adaptation of application protocols to SCTP.

1. Introdução

1.1. Histórico

A grande maioria dos sistemas distribuídos fracamente acoplados, desde o mais trivial sistema cliente/servidor até os *clusters* Beowulf e MOSIX, fazem uso de TCP/IP como meio de comunicação de rede. A escolha recai sobre a pilha TCP/IP por ser universalmente suportada, e sem dúvida por ser a pilha de protocolos em uso na Internet mundial.

Um requisito no desenvolvimento de todo sistema distribuído fracamente acoplado é a transmissão confirmada de mensagens indivisíveis. No entanto, a quase totalidade das implementações de TCP/IP oferece apenas transporte não confirmado de mensagens indivisíveis (UDP), ou transporte confirmado de uma seqüência de octetos (TCP).

Para adequar-se àquilo que o TCP/IP oferece, cada protocolo de aplicação tem de implementar seu próprio método de separação de mensagens sobre TCP; ou então implementar controle de erros sobre UDP – o que for menos inconveniente para a aplicação em questão. Conceber e otimizar um esquema de controle de erros é sem dúvida muito mais difícil. Assim, a maioria dos protocolos opta por usar TCP e implementar a separação de mensagens.

Os protocolos clássicos da Internet que operam sobre TCP (SMTP, FTP, HTTP) utilizam linhas de texto legível ASCII para conversação; o fim-de-linha faz o papel de separador de mensagem. Existe uma tendência moderna em usar-se XML sobre HTTP como método de RPC. O uso de linhas de texto é ineficiente se comparado a estruturas binárias, mas facilita a separação de mensagens, o que explica em parte sua adoção generalizada.

Já nos protocolos que usam UDP como transporte, predomina o uso de estruturas binárias extremamente eficientes em tamanho e processamento. Exemplos: DNS, NFS, NTP. É muito fácil tratar uma mensagem UDP, pois ela é recebida exatamente como foi remetida. Também usam UDP os protocolos de multimídia em tempo real, pois nestes o controle de erros resume-se ao reordenamento no receptor.

São usuários compulsórios de UDP as aplicações que necessitam de *broadcast* ou *multicast*, pois a pilha TCP/IP suporta apenas difusão não confirmada, o que implica em usar transporte não confirmado.

Mas a tendência é a diminuição do uso de UDP. Mesmo a multimídia em tempo real comercial tem migrado para TCP, embora este seja totalmente inadequado para a tarefa.

A migração é motivada não só pela dificuldade em se usar UDP, mas também por motivos mais mundanos: TCP consegue furar mais facilmente o bloqueio de *firewalls*, *proxies* e roteadores NAT. Para ficar num exemplo: *RealPlayer*, uma das primeiras aplicações para multimídia em tempo real na Internet, começou usando UDP como transporte. Hoje, praticamente todas as *streams Real* são servidas sobre TCP.

Ainda outro fator para o abandono do UDP é a inexistência de qualquer recurso de segurança contra seqüestro de conexões. Uma aplicação UDP está por sua conta contra seqüestro de conexões e forjamento de pacotes. O TCP é menos frágil a esses ataques.

O SCTP (*Stream Control Transmission Protocol*) é um protocolo de transporte relativamente novo, que facilita a integração de diversas tecnologias com TCP/IP e Internet. A principal característica do SCTP é a transmissão de mensagens indivisíveis confirmadas.

O foco inicial do SCTP foi a integração das pilhas de rede SS7 e TCP/IP, na troca de mensagens de sinalização telefônica. Sinalização telefônica é o fluxo de informações administrativas, como mensagens SMS, tarifação, serviços ao usuário, sinal de ocupado etc, transmitida por uma rede de comutação de pacotes. A transmissão do conteúdo (voz) faz uso de uma rede de comutação de circuitos separada.

Nessa função criticamente importante, o SCTP já é largamente utilizado, e cada vez mais na medida que as empresas telefônicas migram a rede de sinalização inteiramente para TCP/IP.

Mas o SCTP também tem potencial para ser usado fora do ambiente de telecomunicações. Pode substituir com vantagens os protocolos clássicos de transporte (TCP e UDP) em boa parte dos protocolos de aplicação da Internet. Um estímulo adicional ao uso do SCTP é a presença de características modernas de segurança.

1.2. Objetivo geral

O objetivo geral deste trabalho é apresentar o protocolo de transporte SCTP como uma alternativa viável e consistente ao TCP e ao UDP na implementação de sistemas distribuídos.

O protocolo já existe como padrão da Internet (RFC 2960) desde o ano 2000, e desempenha um papel vital em redes de telecomunicações. Conforme será explanado nos capítulos a seguir, o SCTP resolve vários problemas inerentes ao TCP, inclusive problemas de segurança.

Ainda assim, sua existência é desconhecida por grande parte dos profissionais de informática e mesmo do mundo acadêmico.

1.3. Objetivos específicos

- Estudo da adaptabilidade de protocolos de aplicação existentes ao SCTP;
- Testes básicos do acesso aos recursos do SCTP pela API *BSD/Sockets*;
- Criação de aplicativos básicos baseados em SCTP para testes de vazão e latência;
- Escolha de protocolos de aplicação e *softwares* a serem adaptados para o SCTP;
- Adaptação dos *softwares* ao SCTP e teste de funcionalidade básica;
- Criação de cenários típicos de rede, pela variação de banda, latência e perda de pacotes;
- Criação de ambiente multicaminhos, teste de funcionalidade e resistência a falhas de rede;
- Testes de desempenho nos ambientes criados.

1.4. Metodologia empregada

A metodologia empregada para chegar ao presente trabalho foi uma extensa pesquisa bibliográfica, sobre os seguintes tópicos:

- o próprio protocolo SCTP;
- extensões propostas ao SCTP ainda em forma de *drafts*;
- as extensões à API BSD/Sockets para suporte ao SCTP;
- a pilha de protocolos SS7, utilizada no sistema telefônico mundial;
- problemas de segurança do TCP e UDP;
- controle de congestionamento do TCP;
- camadas de adaptação entre SS7 e TCP/IP;
- algoritmos de *checksum* utilizados nos protocolos TCP/IP.

2. História do protocolo SCTP

2.1. Antecedentes

Na pilha de protocolos OSI, a camada de transporte oferece serviços de transferência de dados em diversas modalidades: mensagens confiáveis, dados confiáveis (seqüência de octetos), dados não confiáveis, datagrama com reconhecimento, datagrama sem reconhecimento e pedido/resposta. São, no total, seis opções (MAZZOLA).

Já na pilha TCP/IP tradicional, há apenas os dois extremos: transporte confirmado de seqüência de octetos (TCP) e transporte não confirmado de datagramas (UDP).

Para os serviços orientados a conexão, o OSI oferece vários níveis de confiabilidade. Em linhas gerais, escolhe-se uma confiabilidade inversamente proporcional à oferecida pela camada de rede, para que a soma das duas resulte em confiabilidade suficiente, com o mínimo de sobrecarga.

Essa característica do OSI trai sua origem – ele foi projetado por um comitê de empresas de telecomunicações. Em redes de telefonia, a confiabilidade de cada meio de transmissão é perfeitamente conhecida, geralmente muito alta e estável ao longo do tempo, e faz sentido modular a garantia do serviço de transporte.

O protocolo TCP funciona mesmo que o meio subjacente seja pouco confiável e instável; porém não faculta à aplicação relaxar nenhuma de suas garantias mesmo que o meio seja 100% confiável.

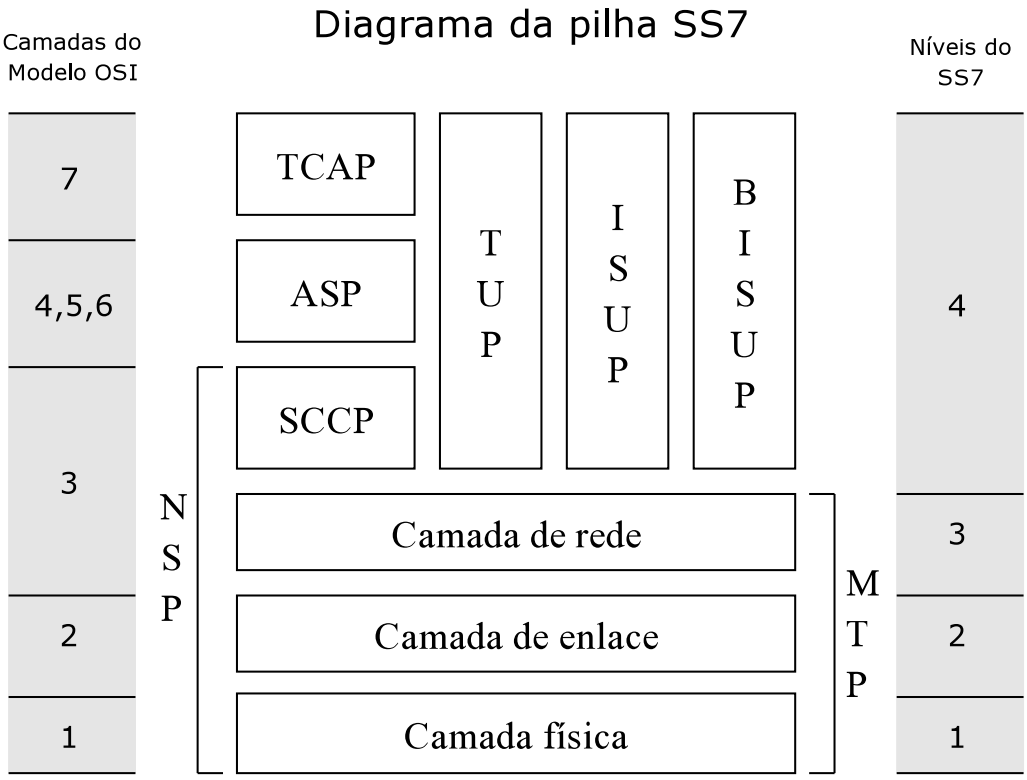
Em telefonia, conforme RUSSELL e ARIAS-RODRIGUEZ, é muito comum a troca de mensagens atômicas entre sistemas; é essencial um serviço de datagrama confirmado, ou de mensagens atômicas confirmadas.

Um serviço de mensagem indivisível confirmada sempre fez falta ao TCP/IP, muito embora essa lacuna seja surpreendentemente pouco lembrada na literatura de redes.

Essa falta poderia ser suprida por uma camada de sessão, que se encarregaria de codificar e separar as mensagens individuais. Mas TCP/IP não define formalmente uma camada de sessão; qualquer protocolo de sessão ou apresentação tem de ser implementado como uma subcamada inferior do protocolo de aplicação (STEVENS, 1994). É o que acontece com o SSL – *Secure Sockets Layer* – que corresponderia às camadas de sessão e apresentação do OSI.

Assim, na prática, cada protocolo de aplicação TCP/IP tem de implementar seu próprio mecanismo de separação de mensagens.

Figura 2.1: Pilha de protocolos SS7



(a) Pilha de protocolos SS7

Fonte: RUSSELL

2.2. Descrição geral da rede de sinalização telefônica SS7

SS7 (*Signaling System #7*) é uma pilha de protocolos de comutação de pacotes, criada com a finalidade de transmitir mensagens de sinalização telefônica. RUSSELL, CAMARILLO e ARIAS-RODRIGUEZ são as referências bibliográficas ao SS7 utilizadas neste trabalho. A pilha com os principais protocolos em cada camada é mostrada na figura 2.1.

O SS7 encarrega-se apenas de mensagens de dados, e não do conteúdo em si (voz ou imagem). O conteúdo é transmitido por comutação de circuitos, para que a qualidade de serviço seja garantida. (A tendência é usar comutação de pacotes com garantia de qualidade também para o conteúdo, por economicamente mais vantajoso.)

Conforme RUSSELL, há uma distinção básica entre mensagens: as *relacionadas* a um circuito (e.g. relacionadas a uma ligação telefônica em andamento) e as *não-relacionadas*.

As mensagens relacionadas não necessariamente trafegam pelo mesmo caminho que o conteúdo. Entre centrais telefônicas, as mensagens podem usar caminhos diferentes; já na "última milha" (ligação entre a central e o terminal telefônico) costuma existir um único caminho elétrico. Em ISDN, um canal de 16kbps é reservado para sinalização.

A pilha SS7, apesar de auto-suficiente, pode delegar camadas baixas a outros protocolos (ARIAS-RODRIGUEZ). O ATM tem sido muito usado nessa função pois tem, entre outros recursos, garantia de qualidade de serviço. Mais recentemente, o TCP/IP começou também a ser usado como meio de transporte para o SS7.

As três camadas mais baixas do SS7 têm funções semelhantes às das pilhas OSI e TCP/IP. Uma diferença importante em relação ao TCP/IP é que as camadas de enlace e rede são responsáveis, em seus respectivos contextos, pela retransmissão de pacotes perdidos. Em TCP/IP tal responsabilidade recai apenas sobre a camada de transporte. OSI permite responsabilizar tanto a rede quanto o transporte pela retransmissão.

Em SS7, o estado dos enlaces redundantes é informado às camadas superiores, que podem tomar decisões baseadas nessa informação. Em TCP/IP, não existe forma simples de a aplicação obter essa informação. Já a pilha OSI permite que uma conexão no nível da camada de sessão faça uso de diversas conexões na camada de transporte (ou vice-versa), sobre as quais o aplicativo pode obter informações, se quiser (MAZZOLA).

Quando o SS7 utiliza outro protocolo como enlace ou rede, esse outro protocolo tem de oferecer as garantias supracitadas. O ATM as oferece nativamente. Para que o TCP/IP as ofereça, é necessário um protocolo de transporte confiável, como TCP ou SCTP.

As redes SS7 concentram grande parte da inteligência na própria rede, o que alivia os terminais. Isso é totalmente diferente da Internet, onde toda a inteligência está concentrada nos terminais, e a rede apenas entrega pacotes individuais na base do melhor esforço, sem oferecer nenhuma garantia. (É bem verdade que haveria considerável inteligência numa rede TCP/IP em que os roteadores implementassem RSVP e/ou IGMP. Mas ainda não é esta a realidade da Internet atual.)

A topologia de uma inter-rede SS7 é bastante normatizada e hierarquizada. Os enlaces são classificados conforme o tipo e nivel hierárquico dos equipamentos interligados; o número mínimo e máximo de enlaces redundantes também é normatizado. Graças a tudo isso, os algoritmos de roteamento dinâmico, embutidos na pilha SS7, podem ser relativamente simples e muito eficientes na tarefa de prover o serviço mais confiável possível. RUSSELL traz uma excelente explanação das regras da topologia SS7 bem como dos tipos de enlace.

As redes de telefonia utilizam apenas transmissão confiável de mensagens individuais. O SS7 prevê transmissão de dados por seqüência de octetos, mas na prática, até o presente momento, esse recurso não é usado (RUSSELL).

O endereçamento SS7 é feito através de *point codes*, de tamanho máximo de 32 *bits*. Cada sistema nacional de telefonia usa um tamanho de sua escolha. Nos EUA, é 24 *bits*. O *point code* internacional é de 14 *bits*.

O roteamento pode ser feito de duas maneiras: total ou parcial. Na forma total, cada *point code* tem uma rota específica, de forma análoga a uma rota estática por destino IP, o que seria ineficiente se fosse usado para todos os terminais. Na forma parcial, uma fração variável do endereço é usada como argumento de roteamento, de forma análoga ao roteamento por agregação IP.

O número de *point codes* é relativamente escasso, e o roteamento por agregação agrava essa escassez. Esse problema é mitigado de duas formas.

Primeiro, a própria subpilha MTP possui um parâmetro SIO (*Service Indicator Octet*) de 4 *bits*, que permite identificar de forma limitada serviços diferentes dentro de um mesmo equipamento. Esse parâmetro é análogo ao *Protocol Field* do cabeçalho IP;

Segundo, a subcamada superior de rede SCCP implementa um endereçamento suplementar, análogo às portas dos protocolos de transporte TCP/IP. Numa rede típica, um equipamento de central telefônica possui um *point code* e cada terminal telefônico conectado a ele possui um sub-endereço SCCP.

Na rede SS7, os equipamentos denominados STP (*Signal Transfer Point*) fazem o papel de roteadores, e em algumas situações atuam de forma análoga a roteadores NAT. Por exemplo, numa comunicação internacional, os *point codes* de cada país precisam ser traduzidos de/para *point codes* internacionais.

Em resumo, são as principais diferenças do SS7 em relação a pilhas como TCP/IP:

- presume uma camada física confiável, ou um protocolo de transporte subjacente que ofereça semelhante confiabilidade;
- transporta de forma confiável apenas mensagens individuais sem ordem intrínseca; não faz implementa transmissão de seqüência de octetos. Opcionalmente, permite agrupar mensagens em transações para entrega ordenada;
- as mensagens nunca ultrapassam o tamanho dos datagramas. Tal limitação reflete, por exemplo, no tamanho máximo de uma mensagem SMS celular. O serviço de transporte é um híbrido de datagrama confirmado (porque não fragmenta mensagens grandes) e mensagem indivisível confirmada (porque permite entregar grupos de mensagens ordenadamente);
- suporta diretamente enlaces redundantes e notifica as aplicações das mudanças nos estados desses enlaces;

- a topologia da rede é fortemente normatizada, a função de cada enlace é bem definida, e os algoritmos de roteamento dinâmico são parte integrante da pilha de protocolos;
- os componentes da rede têm inteligência relativamente grande;
- a rede tende a ter parâmetros estáveis (performance, latência, caminho percorrido entre dois pontos etc.);
- a troca de mensagens costuma ser entre equipamentos e não entre usuários (são os equipamentos, e não seres humanos, que provocam diretamente a comunicação).

2.3. Integração entre redes SS7 e IP

Embora a Internet (e implicitamente o TCP/IP) e a rede de telefonia SS7 sejam completamente diferentes, inclusive quanto a seus objetivos finais, existem motivos fortes para a busca da integração:

- acesso facilitado de uma a serviços da outra;
- barateamento de alguns serviços de telefonia através do uso de TCP/IP e/ou da Internet;
- integração de dispositivos telefônicos fixos e móveis à Internet;
- alternativa de conectividade em áreas onde não haja cobertura por uma das redes.

Se toda comunicação de dados de telefonia é feita através de mensagens indivisíveis, é requisito básico achar um meio de transmitir tais mensagens através de uma rede TCP/IP.

Conforme 2.1, o TCP/IP tradicional não apresenta protocolos de mensagem indivisível confirmada. Coube então a um comitê formado em 1998, o *Signaling Transport* (SIGTRAN) escolher ou criar um protocolo para suprir essa falta.

A primeira idéia, segundo ARIAS-RODRIGUEZ, foi simplesmente adotar o TCP para o transporte de mensagens, por ser tradicional e bem conhecido, e tentar melhorar seus pontos fracos.

No entanto, cedo reconheceu-se que o TCP era totalmente inadequado, pois delega à aplicação toda a complexidade de separação de mensagens. Além disso, o objetivo era escolher ou criar um protocolo de transporte que pudesse funcionar sobre *ambos* os protocolos de rede, TCP/IP e SS7 (e por conseqüência nos respectivos dispositivos), para o que o TCP definitivamente não é adequado.

Embora o UDP em si fosse ainda mais inadequado, chegou-se à conclusão que um protocolo de aplicação operando sobre UDP seria o ideal. Embora ainda não existisse, o protocolo almejado foi denominado *Common Transport Protocol* – CTP. Seus requisitos de *design* (ONG, 1999) eram:

- Servir como transporte para os protocolos de mensagens de telefonia;
- Suportar extensões facilmente;
- Oferecer as garantias de um serviço confirmado e suportar situações adversas de rede como congestionamento;
- Ser resistente a falhas e permitir o uso de caminhos redundantes de rede, bem como notificar as aplicações das mudanças de estado da rede;
- Segurança com alguma autenticação básica das partes;
- Permitir alteração dos tempos de *timeout* e retransmissão pela aplicação;
- Multiplexar várias instâncias de aplicação em uma única instância de transporte, com garantia de ordenamento em cada instância de aplicação;
- Permitir o transporte de mensagens maiores que o MTU do caminho.

Batizado o protocolo e descritos seus objetivos, o SIGTRAN passou a aceitar projetos.

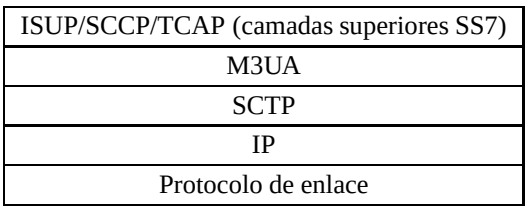
Algumas propostas apresentadas foram: *UDP for TCAP* (T/UDP), *Simple SCCP Tunneling Protocol* (SSTP) e PURDET. Outros protocolos já existentes também foram considerados: *Service Specific Connection-Oriented Protocol* (SCOP), da ITU-T, H.323 Anexo E, também da ITU-T, e o próprio *Real Time Protocol* – o RTP atribui um *timestamp* a cada pacote, o que permite o reordenamento no destino. No entanto, o RTP não garante, por si só, a entrega.

Entrementes, sem qualquer relação com o SIGTRAN, foi submetido ao IETF a proposta de um novo protocolo, denominado MDTP, por Randall R. Stewart e Qiaobing Xie. Seu principal objetivo era eliminar algumas fraquezas do TCP e suportar multicaminhos. O MDTP operava sobre UDP, portanto classificado como um protocolo de aplicação.

Incidentalmente, o MDTP supria quase todos os requisitos do CTP. Tinha ainda uma grande vantagem adicional: os autores tinham desenvolvido uma implementação de prova de conceito, cujo desempenho era comparável ao TCP.

Eleito como CTP, o MDTP foi melhorado e estendido até resultar no SCTP atual.

Figura 2.2: Pilha de protocolos com camada de adaptação M3UA



Fonte: SIDEBOTTOM et al.

O MDTP original nunca chegou a ser publicado como RFC. Muito embora o MDTP tenha servido de base ao SCTP, o número de revisões foi grande e as modificações foram profundas. É de pouca valia estudar o MDTP para entender o SCTP atual, embora possa ser interessante do ponto de vista histórico (vide ARIAS-RODRIGUEZ para informações sobre o MDTP e os passos evolutivos do SCTP).

A integração entre SS7 e TCP/IP faz uso um dispositivo denominado SG (*Signaling Gateway*) que intermedia as comunicações entre terminais SS7 puros e terminais IP/SS7. Dependendo da modalidade de adaptação, o SG atua como roteador ou como *gateway*.

Além do SG, é necessário haver camadas de adaptação para os terminais IP/SS7, pois diversos elementos presentes em SCTP e TCP/IP (porta, endereço, associação) são diferentes ou inexistentes em SS7. A camada de adaptação é inserida entre o SCTP e as camadas superiores SS7 implementadas pelo terminal.

Uma camada de adaptação disponível é a M3UA (SIDEBOTTOM et al, 2002). A pilha TCP/IP mais o M3UA toma o lugar de todo o MTP, conforme mostra a figura 2.2.

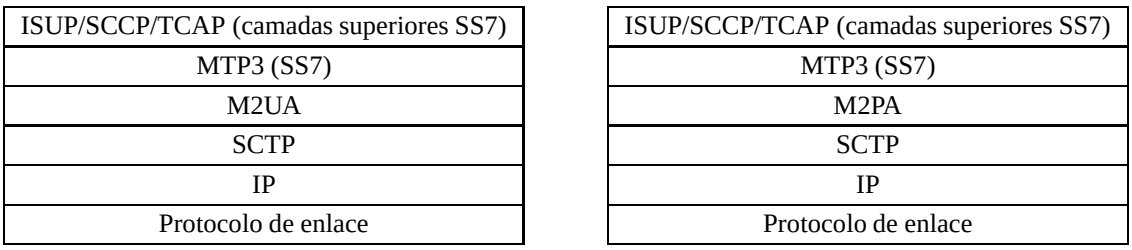
O M3UA possibilita ao SG repassar mensagens SS7 a um terminal IP/SS7. Porém, devido à inexistência da camada de rede SS7 (MTP-3), este terminal não possuirá um endereço SS7 *(point code)* e não poderá ser diretamente endereçado por outros terminais SS7. O SG, nesse caso, faz papel de *gateway*.

A camada M3UA serve para disponibilizar um serviço TCP/IP à rede SS7 da forma mais simples possível, assumindo que a quase totalidade da rede de sinalização ainda seja SS7 pura.

Outras camadas de adaptação disponíveis são M2UA (MORNEAULT et al, 2002) e M2PA (GEORGE), que em conjunto com TCP/IP substituem as camadas MTP-1 e MTP-2 mas mantém a camada MTP-3 do SS7. A figura 2.3 mostra as duas camadas de adaptação entre o SCTP e o MTP3.

A camada de adaptação M2UA destina-se a terminais TCP/IP isolados, que possuem *point-codes* SS7. A camada MTP-3 fica diretamente sobre M2UA nos terminais,

Figura 2.3: Pilhas de protocolos com camadas de adaptação M2UA e M2PA



Fonte: GEORGE et al. e MORNEAULT et al. (2002)

porém não nos SGs. Nestes últimos, é necessário haver uma camada especial (NIF) de comunicação entre MTP2 e M2UA, que toma o lugar do MTP-3.

Assim, o acesso da rede SS7 ao terminal TCP/IP também não é completamente transparente com M2UA. A rede TCP/IP ainda é tratada de forma discriminatória pelo resto da rede SS7, e o "caminho" para a rede TCP/IP passará por um SG bem determinado, diminuindo as possibilidades de redundância e mesclagem.

Ao menos o acesso aos serviços SS7 é bidirecional com M2UA, pois como foi dito o terminal TCP/IP possui um *point code* globalmente endereçável.

A camada de adaptação M2PA oferece os serviços necessários para que a camada MTP-3 funcione diretamente sobre ela tanto nos terminais como nos SGs. Aqui o SG faz o papel de simples roteador, tal qual o STP numa rede SS7 pura. O terminal IP/SS7 possuirá um *point code* globalmente endereçável como em M2UA.

A camada M2PA permite a integração total de SS7 e IP, bem como a construção de redes de sinalização telefônica parcial ou totalmente baseadas em IP.

2.4. A consagração do SCTP como protocolo de transporte

Ao longo do trabalho de criação do SCTP, percebeu-se que este protocolo poderia ter aplicação fora do âmbito da sinalização telefônica. Muitas aplicações poderiam usar o SCTP em *substituição* ao TCP.

Assim como o MDTP, o SCTP também funcionava originalmente sobre UDP, e classificava-se como um protocolo de aplicação. A mudança mais radical no processo de evolução do SCTP foi a migração para a camada de transporte.

O posicionamento do SCTP como um legítimo protocolo de transporte empresta respeitabilidade frente ao TCP, e abre caminho à máxima performance possível. Além disso, o SCTP passa a ter seu próprio escopo de portas, o que evita a presença de serviços confirmados no escopo de portas bem conhecidas UDP.

Essa decisão poderia atrasar o uso generalizado do SCTP, pois via de regra a camada de transporte implementa-se no *kernel*, e os usuários teriam de esperar os respectivos fornecedores de sistemas operacionais implementarem SCTP. Já um protocolo de aplicação pode ser implementado como uma simples biblioteca. Se o SCTP fosse usado apenas para mensagens de telefonia, não haveria vantagem em migrá-lo para a camada de transporte.

A definição formal do protocolo está na RFC 2960 (STEWART et al, 2000). No entanto, esta RFC tem diversos erros e omissões. O *SCTP Implementer's Guide* (STEWART et al., 2003c) contém a listagem das diversas correções que serão futuramente incorporadas à RFC original.

As capacidades do SCTP são facilmente aproveitáveis por praticamente qualquer protocolo de aplicação que utilize TCP. Mas é certo que a aplicação que adotar SCTP certamente terá de continuar suportando TCP durante um longo período de transição, para atender aos clientes de legado que ainda não implementam SCTP.

Assim, decidiu-se que as portas TCP reservadas pelo IANA (as cognominadas "portas bem conhecidas", e.g. porta 80 para Web) estariam também automaticamente reservadas para SCTP.

O IANA livra-se de controlar um novo escopo de portas, e implicitamente permite o uso concorrente de TCP e SCTP para o mesmo serviço no mesmo número de porta.

Há um recurso adicional de multiplexação de tráfego no SCTP. Cada mensagem SCTP possui um rótulo ou identificador de 32 *bits*, informado pela aplicação. Isso permite que serviços completamente diferentes façam uso de uma única associação. (A associação SCTP é análoga à conexão TCP. A diferença de nomenclatura justifica-se pelos novos recursos do SCTP.)

O rótulo por mensagem é um recurso primariamente voltado à sinalização telefônica, mas que pode ganhar importância na Internet, na medida em que escasseiem as portas bem conhecidas TCP/SCTP.

Ainda é objeto de debate se os códigos dos rótulos serão controlados de forma centralizada pelo IANA tal qual é feito com as portas.

2.5. Outros protocolos de transporte em desenvolvimento

O SCTP não cobre todas as lacunas deixadas entre o UDP e o TCP. Diversos outros protocolos têm sido propostos para aumentar o leque de recursos da camada de transporte do TCP/IP, dentre os quais citamos alguns que de alguma forma sobrepõem total ou parcialmente as propostas do SCTP.

O protocolo XTP (*eXpress Transfer Protocol*) oferece praticamente todos os serviços hoje oferecidos separadamente por TCP e UDP, e com diversas vantagens sobre o TCP e.g. menor sobrecarga e retransmissão mais ágil de pacotes perdidos.

Na ausência do suporte ao XTP em um dos terminais, as pilhas de rede comutam automaticamente para TCP, de forma transparente às aplicações. Neste último ponto o XTP leva uma grande vantagem sobre o SCTP.

Além disso, o XTP oferece recursos inéditos de *multicast* confiável (recurso muito atraente para sistemas distribuídos), e controle dos parâmetros de qualidade de serviço. Finalmente, o XTP não depende do protocolo de rede IP; ele pode ser adaptado diretamente a outros protocolos de rede ou mesmo diretamente sobre protocolos de enlace (e.g. ATM).

Vide DANTAS e MENTAT para mais informações sobre o XTP. DANTAS traz em seu livro diversos testes de desempenho do XTP versus TCP, inclusive com o uso de *multicast* confiável para distribuição de arquivos.

O protocolo DCCP *(Datagram Congestion Control Protocol)* é bem menos ambicioso. Ele oferece apenas um serviço ponto-a-ponto essencialmente igual a UDP (não confiável), sem suporte a *multicast*, porém com controle de congestionamento. Vários algoritmos de controle de congestionamento podem ser suportados; a aplicação escolhe o melhor conforme o tipo de tráfego. FLOYD descreve tanto o protocolo DCCP quanto a motivação de sua criação.

O protocolo NETBLT (*NETwork Block Transfer Prototocol*) tem como prinicipal objetivo a transferência de grandes quantidades de dados, ou seja, proporcionar grande vazão, inclusive em redes de grande latência.

Por outro lado, o VMTP (*Versatile Message Transfer Protocol*) foi desenvolvido para transportar transações no estilo RPC (em que a baixa latência é mais importante), e fornecer uma infra-estrutura de transporte voltada para sistemas distribuídos, contando inclusive com serviços de *multicast* (vide DANTAS para informações sobre NETBLE e VMTP).

2.6. Conclusões

Embora o SCTP seja ainda pouco conhecido na comunidade de tecnologia da informação em geral, já é um fato consumado em telecomunicações, e aparece em praticamente toda literatura moderna dessa área.

Como foi visto, o SCTP é uma peça indispensável na integração entre redes SS7 e IP. Em todo o mundo, essa integração anda a passos largos, e as novas redes de sinalização

telefônica tendem a ser parcial ou totalmente baseadas em IP, o que concorre inclusive para a redução do preço dos serviços de telefonia ao consumidor final.

O SCTP é um protocolo novo, com características modernas, que absorveu as lições aprendidas nas últimas décadas com os demais protocolos de transporte da pilha TCP/IP.

O comitê SIGTRAN fez um excelente trabalho ao procurar não “reinventar a roda” no processo de criação do SCTP a partir do MDTP. Também foi feliz em tornar o SCTP adequado ao uso na Internet pública, sem perder de vista o objetivo inicial do comitê que era o suporte ao transporte de sinalização telefônica.

3. Apresentação do SCTP

A RFC 2960 (STEWART et al, 2000) contém a descrição oficial do protocolo SCTP. A definição ali contida diz:

> "O protocolo SCTP é um protocolo de transporte confiável operando no topo de uma rede de pacotes sem conexão como IP. Oferece os seguintes serviços a seus usuários:
>
> - entrega confirmada, livre de erros e não duplicada de dados de usuário;
> - fragmentação de dados em conformidade com o MTU descoberto do caminho;
> - entrega seqüencial de mensagens de usuário em múltiplos fluxos, com opção para entrega por ordem de chegada de mensagens de usuário individuais;
> - empacotamento opcional de múltiplas mensagens de usuário em um único pacote SCTP; e
> - tolerância a falhas de rede através do suporte a multicaminhos em qualquer ou ambas as extremidades de uma associação".

3.1. Associações e fluxos *(streams)*

Dois terminais estabelecem uma *associação* SCTP, e não uma conexão como em TCP. Isso porque cada conexão TCP possui apenas um fluxo *full-duplex*; já uma associação SCTP possui um número arbitrário de fluxos, negociado entre as partes na criação da associação.

Cada fluxo SCTP é um canal de comunicação unidirecional. Pode perfeitamente haver um número desigual de fluxos para cada direção. Uma associação SCTP que pretenda simplesmente emular TCP usa dois fluxos, um em cada direção.

Diferente do TCP, não existe em SCTP o estado meio-fechado (*half-closed*) nem para associações nem para fluxos.

3.2. Mensagens indivisíveis

A definição do SCTP fala em transmissão de mensagens, ao invés de octetos. Isso porque cada mensagem é recebida atomicamente, como um bloco indivisível, exatamente da forma que foi transmitida. A aplicação nunca receberá a mensagem pela metade, ou em pedaços.

Nesse aspecto, o SCTP parece um serviço de datagrama confirmado, pois as aplicações têm a garantia da atomicidade da mensagem.

No caso do UDP, a garantia da atomicidade é incidental pois a mensagem tem de caber no datagrama, que é sempre entregue (ou perdido) atomicamente.

O TCP não oferece nenhuma garantia do gênero. Os dados são transmitidos como uma seqüência não formatada de octetos, fornecendo à aplicação a metáfora de um enlace serial assíncrono 100% confiável. Cabe às aplicações interpretar a seqüência de octetos e separar as mensagens contidas nela, baseando-se unicamente nos dados. O mais próximo que o TCP oferece em termos de separação de mensagens é o *flag* PSH (obsoleto) e o estado meio-fechado da conexão.

Assim como o TCP, o SCTP é confirmado; as mensagens são entregues na ordem da transmissão e sem duplicações. As mensagens podem ter tamanho virtualmente ilimitado (até $2^{32} - 1$ octetos); o protocolo encarrega-se de fragmentar a mensagem em diversos datagramas SCTP e remontá-la no destino.

Nada impede a aplicação de ignorar completamente a formatação em mensagens e tratar os dados como uma seqüência de octetos, ao estilo TCP. Mesmo assim, a vazão bruta do SCTP comparável a TCP dado o mesmo cenário de rede.

Ordem de entrega das das mensagens

O SCTP é um protocolo confirmado tal qual TCP. Os algoritmos de reordenamento, *timeout* e retransmissão de pacotes, controle de congestionamento e descoberta do PMTU, também são implementados no SCTP. O algoritmo de controle de congestionamento é realmente o mesmo do TCP, pois é comprovadamente eficaz, e o uso do mesmo algoritmo assegura lealdade na convivência entre os dois protocolos (STEWART & XIE, 2002).

Em TCP, a seqüência de octetos sempre é entregue na ordem estrita em que foi remetida. Isso é uma garantia importante: se uma seqüência de octetos for embaralhada, torna-se inútil. Porém se várias mensagens forem transmitidas através de uma mesma conexão, a perda de um pacote IP pode atrasar a entrega de todas as mensagens – problema conhecido com *Head-of-Line Blocking*.

Um exemplo corriqueiro é uma requisição múltipla HTTP através de uma única conexão TCP. Se o servidor devolver dez arquivos, cada arquivo contido em aproximadamente um datagrama, e o primeiro datagrama perder-se, a entrega dos últimos nove datagramas (e portanto de nove arquivos) será adiada até que o pacote perdido seja retransmitido.

Já no SCTP, as mensagens são entregues à aplicação na ordem em que foram transmitidas, *apenas dentro do contexto de cada fluxo*. Mensagens pertencentes a fluxos diferentes não são necessariamente entregues na ordem da transmissão.

Aplicando o exemplo do HTTP ao SCTP, pode-se usar dez fluxos, um por arquivo solicitado. A perda de um datagrama atrasa apenas a entrega das mensagens de um único fluxo; o navegador pode receber e exibir os outros nove arquivos, enquanto o SCTP providencia a retransmissão do pacote perdido.

O TCP fornece a metáfora de mensagem urgente, ou fora-de-banda, entregue separadamente da seqüência principal. Porém essa mensagem é de apenas um octeto por pacote TCP. E se a aplicação não ler o octeto urgente logo, uma nova mensagem urgente sobrescreve a primeira. Praticamente nenhum protocolo de aplicação faz uso desse recurso.

Em SCTP também existe a mensagem urgente que, embora atrelada a um fluxo, é entregue assim que recebida (fora de ordem), mesmo que haja outras mensagens esperando por retransmissão. A mensagem urgente SCTP pode ter tamanho arbitrário; a aplicação pode emití-las em qualquer número; não há perda de mensagens urgentes se a aplicação demorar-se em lê-las.

Se a aplicação é capaz de lidar com mensagens desordenadas, ou mesmo não se importa com a ordem de chegada delas, pode remeter todas as mensagens de um fluxo como urgentes, de modo a evitar qualquer atraso por perda de pacotes.

3.3. Multicaminhos *(multi-homing)*

Outro recurso totalmente novo do SCTP em relação aos demais protocolos de transporte é o suporte direto a multicaminhos.

O computador *multi-homed* é aquele ligado a duas ou mais redes IP, sem necessariamente ser um roteador. Particular importância tem o computador ligado por dois ou mais caminhos à Internet mundial, por proporcionar tolerância a falhas e garantir a continuidade do serviço.

Via de regra, para uma entidade que não seja um *Autonomous System* (AS), estar ligado por vários caminhos à Internet não proporciona confiabilidade às conexões TCP

e UDP em si. A rota para cada endereço do computador *multi-homed* é estática. Se um enlace cair, as conexões atreladas àquele endereço têm de ser fechadas, e então reabertas para o outro endereço. O serviço continua, porém à custa de transtorno ao usuário e/ou complexidade na aplicação.

Se a entidade for um AS, dependendo dos acordos que ela tem com os AS adjacentes, o roteamento dinâmico BGP encarrega-se de achar uma rota alternativa para o endereço, e as conexões em andamento não morrem.

O SCTP implementa tolerância a falhas para a associacão, sem necessidade da entidade ser um AS. Cada terminal informa ao outro uma lista de IPs que são seus endereços na rede. O IP utilizado durante a criação da associação é o IP primário, principal canal de comunicação com o terminal remoto, que provavelmente foi obtido via resolução DNS. Os demais IPs informados são secundários, e são utilizados caso falhe a comunicação com o IP primário.

Essa hierarquia entre IP primário e secundários vai ao encontro de um outro caso típico de multicaminhos: computador ligado a um enlace primário terrestre (rápido e barato) e a outro enlace por satélite (caro, lento e com alta latência).

O SCTP controla o estado dos diversos IPs do terminal remoto por mecanismos de *heartbeat*, de modo a sempre conhecer a situação de cada caminho e fazer deduções (e.g. se todos os caminhos aparentemente falharam é mais provável que o próprio terminal remoto tenha falhado, e não as redes).

O suporte direto do SCTP a multicaminhos facilita muito a implantação de alta disponibilidade de rede para uma ampla gama de serviços da Internet, o que de outra forma seria possível apenas se a entidade fosse um sistema autônomo (AS), com todos os custos e responsabilidades administrativas associados. (É interessante notar que a pilha OSI também oferece esse recurso.)

O SCTP permite inclusive que uma mesma associação utilize simultaneamente caminhos IPv4 e IPv6. Também permite que o endereço seja informado para o participante remoto como um nome DNS. (Estes últimos recursos são de implementação opcional.)

Como já foi dito, o algoritmo de controle de congestionamento do SCTP é essencialmente o mesmo do TCP. Talvez a principal diferença em relação a TCP seja o suporte a multicaminhos, pois cada caminho tem sua própria latência, taxa de perda de pacotes etc. A existência de múltiplos caminhos também abre espaço para melhorias nos algoritmos de retransmissão, que ainda são objeto de estudo.

3.4. Extensões propostas ao SCTP

O SCTP é facilmente extensível. Graças a isso, pode evoluir rapidamente sem quebrar compatibilidade com versões anteriores.

Tal característica estimula pesquisadores do mundo todo a propor extensões. Muitas acabaram sendo rejeitadas, a maioria por total descabimento. No entanto, algumas extensões a priori rejeitadas são realmente interessantes, e continuam sendo objeto de debate, e com chances de serem aceitas no futuro.

Também é interessante notar que, devido à extensibilidade do SCTP, é fácil a qualquer implementação oferecer extensões fora da definição oficial do protocolo. Toda extensão rejeitada pelo IETF tem a segunda chance de tornar-se um padrão de fato, e assim forçar uma reavaliação por parte do IETF. Tal fato deu-se com a extensão PR–SCTP (STEWART et al., 2003a).

Confiabilidade parcial

Mensagens ou fluxos não confirmados teriam essencialmente as mesmas características do UDP. Porém, como existem dentro do contexto de uma associação SCTP, teriam vantagens importantes:

- maior segurança contra alguns tipos de ataque;
- suporte embutido a multicaminhos;
- mensagens confirmadas de sinalização/controle podem ficar atreladas à mesma associação;
- várias mensagens pequenas, confirmadas ou não, podem ser empacotadas num mesmo datagrama SCTP;
- facilidade no tratamento em roteadores NAT.

Tal extensão tornaria o SCTP particularmente adequado para VoIP, considerando as limitações da Internet atual.

Há duas propostas semelhantes nessa direção: U–SCTP (XIE et al, 2001) e PR–SCTP (STEWART et al, 2003a). Em ambas, figuram Stewart e Xie entre os autores, os principais criadores do protocolo SCTP.

A extensão U–SCTP (*Unreliable SCTP*) propõe a negociação de fluxos não confirmados durante a criação da associação. Todas as mensagens de um fluxo não confirmado seriam conseqüentemente não confirmadas e sem garantia de entrega ordenada.

A extensão PR–SCTP (*SCTP Partial Reliability*) propõe o relaxamento seletivo de garantias em mensagens individuais. Nessa proposta, podem conviver no mesmo fluxo mensagens confirmadas e não confirmadas, assim como já convivem hoje mensagens ordenadas e urgentes. A aplicação pode modular a confiabilidade da mensagem em quatro níveis:

- confirmada (existente no SCTP original);
- confirmada sem garantia de ordenamento (existente no SCTP original);
- confirmada com prazo de validade;
- confirmada sem garantia de reordenamento e com prazo de validade.

Ambas as propostas foram *a priori* rejeitadas porque o SCTP é, por definição um protocolo confirmado, e julgou-se que adicionar confiabilidade parcial seria conflitante com os demais objetivos do protocolo.

Mas o PR–SCTP acabou oficialmente aceito e normatizado na RFC 3758 (STEWART et al., 2004b). O suporte a esta extensão é opcional. Mesmo desde antes da RFC, algumas implementações já ofereciam esta extensão, com variados graus de maturidade.

A versão final do PR–SCTP é relativamente simples de adicionar a qualquer implementação SCTP. Basta duas alterações principais:

- Novo trecho “avançar TSN”, que indica ao terminal remoto que a janela de recepção deve ser avançada, pois algumas mensagens pendentes não serão mais entregues;
- Novo trecho “suporte ao avanço de TSN” incluído em INIT/INIT-ACK que indica ao terminal remoto o suporte ao mecanismo descrito supra;
- Na API, mudar a semântica do prazo de validade. No SCTP normal, o prazo de validade só pode invalidar mensagens em fila cuja transmissão ainda não foi tentada. No PR–SCTP, mensagens vencidas serão expiradas mesmo que sua transmissão já tenha sido tentada.

A extensão só pode ser usada se suportada pelas duas pontas. De acordo com as regras de tratamento de trechos desconhecidos, a pilha SCTP que não suporta PR–SCTP retornará um erro recuperável ao receber o trecho “suporte ao avanço de TSN”, o que indica ao terminal remoto a inexistência da extensão.

Um fluxo com características semelhantes a UDP pode ser emulado através de mensagens sem garantia de reordenamento e com prazo de validade igual ao RTT da rede. Como é a aplicação quem atribui o prazo de validade das mensagens em PR–SCTP, ela terá de monitorar o RTT da associação e/ou interpretar os avisos de mensagens não transmitidas por *timeout*, preocupações que não existem em UDP. (Como a própria API do SCTP oferece recursos de monitoramento do RTT, a aplicação ao menos não precisa calcular o RTT por conta própria.) Também não é possível utilizar *multicast* e *broadcast* com PR–SCTP.

O PR–SCTP oferece duas vantagens importantes sobre UDP: mantém os algoritmos de controle de congestionamento do SCTP, e garante lealdade com TCP. Numa situação de congestionamento severo de rede, a maioria das mensagens expiraria ainda na fila de transmissão, aliviando a rede. Nessa mesma situação, todos os pacotes UDP remetidos pela aplicação seriam transmitidos, ainda que descartados nos roteadores intermediários, piorando o congestionamento e potencialmente tornando o tráfego desleal em relação a outras conexões.

Na API *BSD/Sockets*, e possivelmente em outras APIs, transmitir um pacote UDP não bloqueia; leva apenas o tempo necessário para o pacote ser enfileirado na camada de enlace. A aplicação pode transmitir pacotes UDP tão rápido quanto suportado pelo enlace diretamente conectado ao computador. (Naturalmente, não existe qualquer garantia que os roteadores intermediários serão capazes de lidar com esse tráfego.) Se este comportamento é desejado pela aplicação, ela deve usar UDP. Do contrário, deve usar protocolos com controle de congestionamento como TCP ou SCTP.

Criação de fluxos sob demanda

O número de fluxos em cada sentido é combinado entre as partes durante a criação da associação, e permanece imutável até o encerramento. Isso é perfeitamente adequado para protocolos de aplicação como HTTP, onde o número de requisições é conhecido no momento da associação.

Em outras aplicações, o número de fluxos necessários não é conhecido no estabelecimento da associação. A alternativa atual é reservar um número arbitrariamente grande de fluxos, mesmo que a maioria deles nunca seja usado. Uma solução proposta para esses casos é a criação *sob demanda* de fluxos.

Do ponto de vista do aplicativo, a criação sob demanda funcionaria como a abertura de várias conexões TCP com o servidor, porém com sobrecarga muito menor.

Um número arbitrariamente grande de fluxos ociosos não aumenta, a princípio, o consumo de memória no *kernel* do sistema operacional. Isso significa que os fluxos

sob demanda podem ser eficientemente implementados. Mas também significa que a reserva prévia de um grande número de fluxos também pode ser eficiente e não demanda mudanças no protocolo SCTP.

Assim, embora essa proposta tenha uma certa elegância, dificilmente será admitida ao SCTP.

Estado meio-fechado (*half-closed*) para associações

Em TCP, qualquer dos participantes pode fechar a conexão apenas no sentido da transmissão, continuando a receber dados até que a conexão seja definitivamente encerrada. Nesse intervalo, a conexão está fechada em apenas um sentido, daí o nome. O SCTP não possui esse recurso.

Aqueles que apóiam a decisão de *design* do SCTP apontam que o estado meio-fechado é um apêndice desnecessário, pois os protocolos de aplicação devem basear-se unicamente nos dados para identificar o fim da transmissão. Além disso, o SCTP transmite mensagens de forma indivisível: não existe necessidade de sinalizar fim-de-mensagem na camada de transporte, pois é fácil implementar uma mensagem de fim de transmissão.

Já outros consideram o uso do estado meio-fechado do TCP uma forma legítima e simples de um aplicativo sinalizar fim de transmissão. Além do que sua inexistência no SCTP dificulta a adaptação de alguns protocolos de aplicação.

O estado meio-fechado refere-se às associações. Não faz sentido para os fluxos pois são unidirecionais e seu tempo de vida é exatamente o mesmo da associação.

Balanceamento de carga

O projeto RivuS (vide RIVUS) propõe uma extensão onde múltiplos caminhos possam ser utilizados não apenas para redundância mas também para balanceamento de carga. A extensão é ativada mediante solicitação da aplicação.

A transmissão de pacotes seria feita de forma balanceada para todos os endereços IP do destino, e não apenas para o endereço ativo. Assim, todos os enlaces ativos seriam plenamente utilizados todo o tempo, aumentando a vazão.

Reconfiguração dinâmica de IP (AddIP)

Esta extensão proposta por STEWART et al. (2003b) prevê adição e exclusão de endereços IP secundários em uma associação ativa, bem como a redefinição do IP primário.

A motivação da proposta são os equipamentos que podem sofrer reconfiguração dinâmica do endereço IP, bem como adição e remoção de interfaces de rede, num ambiente com associações de vida longa e criticamente necessárias.

Esta extensão já está presente em algumas implementações.

Redes por satélite

Os enlaces por satélite possuem um produto latência $\times$ largura de banda muito alto, e uma latência particularmente alta. Para esses enlaces, é interessante:

- aumentar o número inicial de datagramas transmitido pelo algoritmo *slow start*, se o enlace for confiável o bastante;
- utilizar uma janela de recepção (*rwnd*) maior. Como isso ocupa memória do *kernel*, deve ser feito apenas para as associações onde tal expediente traga dividendos de performance;
- aumentar a estimativa inicial do *round-trip time* (RTT) e do *timeout* de transmissão (RTO);
- se a perda de pacotes for grande e constante (o que caracteriza enlace pouco confiável, mas seria *a priori* interpretado como congestionamento), ajustar o algoritmo de controle de congestionamento à situação.

A pilha TCP/IP não sabe nada, nem procura descobrir, sobre as camadas físicas que lhe servem de meio. A aplicação deve explicitamente solicitar ao SCTP que modifique os valores dos algoritmos de congestionamento para funcionar melhor via satélite.

Requisição de camada de adaptação

STEWART et al. (2003b) propõe um novo trecho que permite indicar ao terminal remoto qual a camada de adaptação necessária (M2UA, M3UA etc.) para tratar as mensagens.

Este trecho é transmitido juntamente com INIT ou INIT-ACK. A camada de adaptação pode apenas ser solicitada na criação da associação; não pode ser solicitada/alterada numa associação estabelecida.

Este recurso atua em conjunção com o parâmetro “identificador de protocolo” presente no trecho DATA, permitindo ao SCTP interagir perfeitamente com o SS7, sem a utilização da metáfora da “porta bem conhecida” que não existe em SS7.

É previsto que a lista de códigos de camada de adaptação sejam mantida pelo IANA, a exemplo do que acontece com os números de porta. A implementação SCTP deve considerar o código como opaco; é responsabilidade da aplicação interpretá-lo ou ignorá-lo.

3.5. Implementações

Um grande atrativo do protocolo MDTP, que deu origem ao SCTP, foi a implementação de referência. Essa implementação foi a base de testes das mudanças sugeridas ao MDTP, que culminaram no SCTP, e continua sendo mantida pelos autores pelo menos até a data deste trabalho.

Como ela é *user-level* e utiliza apenas chamadas padrão POSIX, funciona em qualquer sistema operacional compatível com POSIX. A implementação do protocolo de transporte fora do *kernel* diminui a performance, mas facilita a manutenção, portabilidade e distribuição do *software*.

Para o sistema operacional *Windows*, é oferecida ao menos uma implementação comercial, também *user-level* e distribuída em forma de biblioteca. A maioria dos UNIXes comerciais têm trabalhado em implementações *kernel-level* nativas para o SCTP. Também há implementações comerciais para UNIX.

Quanto aos sistemas operacionais de livre distribuição, há pelo menos quatro iniciativas de implementação do SCTP, todas *kernel-level*:

- LK-SCTP: para o Linux, coligada ao projeto KAME e em estágio próximo de produção;
- KAME: para o FreeBSD. O projeto KAME também implementa protocolos IPv6 e IPSEC, com destacada qualidade, a ponto de o Linux ter adotado essas implementações como referência;
- RIVUS: para todos os BSD de livre distribuição;
- OpenSS7: projeto que visa implementar uma pilha SS7 de livre distribuição (ainda em estágio imaturo) que codificou sua própria versão do SCTP para o Linux.

Uma lista completa e atualizada pode ser obtida em SCTP.ORG.

4. Aplicabilidade do SCTP

4.1. SCTP em substituição a TCP

Custos e benefícios

O SCTP pode ser visto como uma evolução do TCP (STEWART & XIE, 2002). Qualquer aplicação que utilize TCP também pode usar SCTP, com resultados iguais ou talvez melhores. Essa possibilidade é tão concreta que as portas de serviços TCP bem conhecidos registradas junto ao IANA (e.g. porta 80 para Web, 21 para FTP etc.) são automaticamente reservadas em SCTP para a mesma aplicação.

Em uma simples troca de TCP para SCTP, sem nenhuma mudança na adaptação ao transporte, a aplicação adquire de plano as vantagens do *checksum* CRC-32c (resistência à corrupção de dados), e do recurso de multicaminhos (tolerância a falhas de rede). O SCTP também trabalha mais duro que o TCP para manter a associação aberta.

Todas essas vantagens são colhidas praticamente sem custo de manutenção nas implementações (vide ANEXO 3).

Para fazer uso dos fluxos, atomicidade das mensagens e mensagens urgentes, a adaptação ao protocolo de transporte tem de sofrer mudanças, com correspondente impacto nas implementações. Se se pretende que a adaptação tenha adoção generalizada, também será necessário submeter a mudança ao escrutínio público, culminando com a publicação de uma RFC. Cabe uma análise de custo/benefício bem mais cuidadosa.

Os fluxos SCTP aproveitam a qualquer protocolo de aplicação que faça uso de várias conexões TCP em paralelo para executar uma única tarefa. O uso de uma única associação economiza recursos, melhora a lealdade entre usuários diferentes e facilita o tratamento em roteadores NAT.

A atomicidade de mensagens traz benefícios para protocolos onde as mensagens sejam enquadráveis numa estrutura da linguagem C e/ou tenha tamanho previsível, pois elimina completamente a necessidade da separação de mensagens na camada de aplicação.

Juntamente com cada cada mensagem atômica SCTP é possível transmitir um identificador de protocolo, de 32 *bits*. É um meta-dado que facilita a tarefa de entregar eficientemente a mensagem a seu tratador. Também é possível usar o recurso de mensagem urgente se for útil à aplicação.

Mensagens com tamanho imprevisível (potencialmente muito grande, como por exemplo um arquivo inteiro) não devem ser transmitidas atomicamente, pois a implementação pode impor limites relativamente estreitos ao tamanho máximo da mensagem SCTP (vide ANEXO 2). Esse tipo de mensagem deve continuar sendo tratada inteiramente na camada de aplicação.

HTTP

O protocolo de aplicação HTTP é muito simples, o que favoreceu sobremaneira sua popularidade. Basicamente, o cliente requisita um arquivo do servidor, e o servidor devolve o arquivo desejado, em formato binário. Feito isso, a conexão TCP é fechada.

Na requisição, o cliente geralmente anexa uma lista de parâmetros informativos sobre o sistema operacional e o navegador. Dependendo da requisição, também são passados parâmetros de usuário e.g. os dados de preenchimento de um formulário HTML. A presença dessa lista de parâmetros não afeta a simplicidade do protocolo, pois é de competência do servidor interpretar essa lista como lhe aprouver.

Numa página Web típica, há uma página HTML base que contém o texto, mais um número relativamente elevado de elementos gráficos. Cada um desses elementos é um arquivo separado no servidor, que deve ser individualmente requisitado pelo cliente.

Há dois métodos de requisitar todos esses elementos. A forma primitiva, suportada por todo cliente e servidor HTTP, é estabelecer uma conexão TCP/IP para cada requisição. Embora muito simples, esse método possui, segundo ARIAS-RODRIGUEZ, sérias desvantagens:

- Consome muitos recursos no cliente e em particular no servidor, devido ao grande número de conexões TCP abertas. Como o número de conexões TCP simultâneas é limitado pelo *kernel*, um servidor HTTP muito requisitado pode rapidamente saturar;
- O tráfego gerado é “desleal” em relação a outros serviços, pois abre diversas conexões para uma única tarefa e indiretamente aloca mais largura de banda para ela;
- A quase totalidade de clientes HTTP (navegadores, *proxies* etc.) procura limitar o número de conexões simultâneas a um mesmo servidor. Na medida que uma encerra, outra é aberta. Isso mitiga o problema mas adiciona complexidade aos clientes;
- A determinação do número máximo de conexões a um mesmo servidor é subjetiva e dá margem a “deslealdade” de um navegador perante outros;

- Também há desperdício de recursos da Internet, pois cada arquivo, por menor que seja, implicará no tráfego de no mínimo nove pacotes IP (sete para iniciar e terminar a conexão TCP, e no mínimo dois para transmissão de dados HTTP úteis).

Pode-se utilizar a extensão do HTTP que permite seqüenciar vários arquivos através de uma única conexão TCP/IP. Este método resolve de forma efetiva o excesso de conexões TCP/IP, porém:

- adiciona complexidade às aplicações, que têm de interpretar a seqüência de caracteres e separar as diversas requisições e arquivos, bem como tratar a ocorrência de erros e omissões;
- potencializa o *Head of Line Blocking*. Se o servidor HTTP entregar dez arquivos, e um pacote IP do primeiro arquivo for perdido, a entrega dos outros será adiada até que esse pacote perdido seja retransmitido.

Todos os problemas acima são resolvidos pela utilização de fluxos SCTP:

- cada fluxo de transmissão contém uma requisição, e cada fluxo de retorno contém um arquivo;
- o atraso de um fluxo, causada pela perda de um pacote IP, não atrasa a entrega das demais;
- essa separação não causa ineficiência pois várias mensagens pequenas de fluxos diferentes são empacotadas num mesmo datagrama IP;
- o fato de cada requisição figurar em sua próprio fluxo permite simplificar os aplicativos, que não precisam separar múltiplas requisições;
- a sobrecarga dos terminais é reduzida pois apenas uma associação entre cliente e servidor é necessária;
- a sobrecarga de rede é reduzida pois apenas uma associação precisa ser criada e destruída (7 pacotes IP), independente do número de arquivos transmitidos. O uso de múltiplos fluxos não implica em sobrecarga.

O número de arquivos HTTP a requisitar, e portanto o número de fluxos a serem usados, não é conhecido senão depois de o navegador requisitar a página principal. Em páginas com *frames*, ainda é necessário requisitar as sub-páginas. Outro detalhe a ser considerado é que a atomicidade de mensagens SCTP não parece ter utilidade para HTTP.

Sistemas de arquivos distribuídos

Praticamente todos os sistemas de arquivos distribuídos (NFS, CIFS, CODA, AFS, Intermezzo) podem tirar proveito da atomicidade das mensagens. A bibioteca de tipos de requisições é bem definida, e o tamanho máximo das mensagens é limitado. A atomicidade elimina ou simplifica a decodificação, e agiliza o processamento.

Se o sistema de arquivos está operando em modo assíncrono, todas as mensagens podem ser rotuladas como "urgentes" para evitar *HOL blocking*.

Multimídia via Internet

Alguns serviços multimídia, até mesmo de tempo real, usam HTTP sobre TCP como meio de transporte. Embora totalmente inadequado, ainda assim é utilizado, por passar facilmente em *proxies*, roteadores NAT e *firewalls*.

Multimídia sobre TCP até funciona na prática, desde que haja grande sobra de largura de banda e baixa perda de pacotes. Mas cedo ou tarde ocorre um congestionamento mais severo e a ressincronização torna-se impossível. Isso é particularmente verdadeiro para multimídia em tempo real, onde a geração de dados não pára e os dados retransmitidos são simplesmente jogados fora pelo receptor, pois sua validade já expirou.

O uso de SCTP em lugar de TCP, embora também inadequado para esse fim por também ser confirmado, traria algumas vantagens marginais:

- o fluxo multimídia poderia ser mandado integralmente em mensagens urgentes, que podem ser entregues fora de ordem, o que evita o *HOL blocking* (desde que o protocolo de aplicação seja capaz de lidar com fluxo fora de ordem);
- a notificação do SCTP acerca de pacotes perdidos é mais elaborada, o que potencialmente economiza retransmissões;
- os limites mínimo e máximo do RTO (*timeout* de retransmissão) podem ser alterados para melhor adequar-se à tarefa (porém, muito cuidado deve ser tomado para não tornar o fluxo desnecessariamente agressivo).

O SCTP torna-se ideal para multimídia em tempo real, melhor que TCP e UDP, quando implementa a extensão de confiabilidade parcial (PR–SCTP).

4.2. SCTP em substituição a UDP

Algumas literaturas distinguem UDP de TCP classificando o primeiro como um protocolo “sem conexão”. Na verdade, existem sim conexões UDP, identificadas unicamente pelos endereços e números de porta; do contrário não seriam tratáveis por roteadores NAT; mas o UDP *delega à aplicação* a responsabilidade administrativa pelas conexões.

O que parece uma fraqueza pode ser uma grande vantagem. As conexões TCP e associações SCTP residem na memória do *kernel*, que é um recurso escasso. As “conexões” UDP residem na aplicação, que tem à sua disposição memória virtual e outras amenidades.

Em TCP/IP, a difusão (*broadcast* e *multicast*) é não-confirmada, e só pode ser utilizada por protocolos de transporte não confirmados. O motivo de usar-se difusão é desonerar o transmissor de conhecer nominalmente cada um dos receptores, e por extensão evitar o custo de uma conexão por receptor. Novamente o ponto é livrar-se da responsabilidade pelas conexões, desta vez transferindo o ônus para a camada de enlace e/ou para os roteadores intermediários.

Em aplicações usuárias de UDP onde ocorre grande número de conexões de vida muito curta e baixo tráfego de dados – o exemplo perfeito é DNS – não há vantagem em migrar para SCTP. O mesmo para aplicações que fazem uso de *broadcast* e *multicast*. Sem a extensão de confiabilidade parcial (PR–SCTP), são poucas as oportunidades de usar SCTP em lugar de UDP.

No caso específico do DNS, apenas as conexões de transferência de zona (que sempre são TCP, inclusive por motivos de segurança) e a conexão de uma estação de trabalho com o seu servidor DNS cache (que é de longa duração) teriam alguma vantagem em usar SCTP.

UDP não oferece por conta própria nenhuma garantia de reordenamento e retransmissão. Embora isso traga a vantagem marginal de evitar o *head-of-line blocking*, perda e desordem de dados são sempre indesejáveis. Cada aplicação baseada em UDP tem de compensá-las de alguma forma.

Com a extensão PR–SCTP de confiabilidade parcial, cria-se oportunidade de substituição do UDP, em aplicações que criem conexões de vida mais longa e/ou tráfego de dados expressivo. Exemplos: multimídia (em tempo real ou não) via Internet e telefonia sobre IP. Tais aplicações teriam enorme vantagem em utilizar PR–SCTP, pois desonera a aplicação de algumas tarefas, e o tráfego de sinalização (confirmado) pode trafegar pela mesma associação que o conteúdo.

Também é mais fácil para o roteador NAT tratar uma associação SCTP única, do que tratar um fluxo TCP mais fluxos UDP associados. Para o roteador NAT saber que determinado fluxo UDP é relacionado ao fluxo TCP de controle (como acontece por exemplo no protocolo H.323), precisa analisar o tráfego da camada de aplicação – o que viola a arquitetura de camadas e é ineficiente.

4.3. Novas possibilidades de uso do TCP/IP e da Internet

O SCTP opera sobre IP, e é sujeito as limitações deste. Cada novo recurso do SCTP pode ser implementado na camada de aplicação. Por que então implementar ainda outro protocolo de transporte, se ele não cria nada por si só?

O SCTP não é exatamente um *criador* de novas aplicações, mas sim um *promotor* de aplicabilidade, pois compila grande número de recursos e melhorias, de forma ordenada, universalmente compatível, aliviando a complexidade dos protocolos de aplicação, e com grande potencial de performance se for implementada no *kernel*.

Tomando como exemplo o recurso de multicaminhos, que é um recurso SCTP transparente à aplicação. É perfeitamente possível a uma aplicação implementar por si esse recurso. Porém ela teria de ser baseada em UDP, e implementar também o controle de erros. Ou então basear-se em TCP e criar um produto cartesiano de conexões TCP, lidar a com a burocracia de administração dessas conexões, retransmitir dados perdidos no caminho quando uma conexão cair etc. E uma vez que uma aplicação conseguisse implementar e testar multicaminhos, esse recurso não aproveitaria a nenhuma outra aplicação.

Sinalização telefônica

O transporte de sinalização telefônica é a grande motivação da existência do protocolo SCTP. Praticamente todos os recursos do SCTP, bem como a boa parte dos *drafts* posteriores, objetivam atender a esse público-alvo.

A integração do SCTP com redes de telefonia é explanada em detalhes na seção 2.2.

Dispositivos portáteis, eletrônica embarcada e redes fabris

O uso de TCP/IP nesse tipo de aplicação seria desejável pelo baixo custo e grande disponibilidade de *hardware*, *software* e massa crítica de conhecimento relacionados. Além disso, abre toda uma gama de possibilidades de integração a computadores de uso geral, a outras redes ou mesmo à Internet. Mas o protocolo de transporte TCP é, porém, inadequado a esse tipo de rede. O protocolo UDP é muito carente de recursos. Nenhum dos dois protocolos oferece, ao mesmo tempo, garantia de entrega, tolerância a falhas e baixa latência – características do SCTP.

A especificação do SCTP é flexível permite às implementações restringir o número máximo de fluxos, bem como limitar o tamanho da mensagem ao PMTU, tornando-as enxutas e apropriadas a dispositivos limitados em memória e CPU.

Enlaces via satélite

O TCP tem fraco desempenho típico em redes via satélite, por uma combinação dos seguintes motivos:

- uso de janelas de transmissão muito pequenas – as aplicações não têm a "cultura" do uso da opção de escalonamento de janela, nem ao menos do uso de *buffers* de recepção acima do *default*;
- algoritmos de congestionamento não adaptados a enlaces *wireless*, que perdem muitos pacotes;
- a confirmação seletiva (SACK) do TCP é limitada, além do suporte a SACK ser opcional;
- excessivo número de conexões concomitantes em aplicações típicas (e.g. HTTP), cada uma incorrendo em sobrecarga e lentidão inicial devido ao *slow start*.

O uso de um *proxy* nesse tipo de enlace melhora a experiência do usuário porém limita sua aplicabilidade.

O SCTP suporta grandes janelas de transmissão (embora ainda dependa da iniciativa das aplicações para usá-las), potencializa o uso de menor número de associações dada a mesma tarefa, e possui um SACK mais elaborado. Tais recursos são padrão do protocolo, não opcionais como no TCP.

A adaptação dos algoritmos de congestionamento ao enlace via satélite ainda é objeto de discussão (vide 3.4).

5. Comparação de desempenho com TCP

5.1. Objetivos

Os testes desenvolvidos neste capítulo objetivam comparar o desempenho do SCTP frente ao TCP, nos critérios de vazão e latência.

O TCP foi escolhido como adversário por ser o protocolo mais utilizado na Internet hoje, por ser um protocolo confirmado como é o SCTP, e por praticamente todos os protocolos de aplicação existentes, que poderiam aproveitar-se das características avançadas do SCTP, hoje usam TCP como transporte.

Diversos cenários de rede foram construídos, utilizando-se recursos de *hardware* e *software*. Foram objeto de variação: largura de banda, taxa de perda constante de pacotes, taxa de duplicação de pacotes, latência de rede e intermitência de funcionamento da rede.

Em cada cenário de rede, duas características básicas foram testadas: vazão e latência. Em conjunto, elas fornecem dados suficientes para concluir sobre o desempenho do SCTP em cada cenário de rede.

Vazão é o volume de dados trafegado por unidade de tempo. A maioria dos usuários percebe mais claramente a vazão que a latência (e.g. acompanhando o progresso de um *download*), e toma vazão e desempenho por sinônimos, embora a vazão isoladamente não garanta bom desempenho. Em todos os testes deste capítulo, a unidade de vazão utilizada é o kbps (1000 *bits* por segundo), por ser a mais comumente utilizada em redes de computadores, ainda que o usuário final costume perceber a vazão em KBytes/s.

Latência é o tempo decorrido entre a formulação de uma requisição, e o recebimento da resposta. Embora isto não seja óbvio para o usuário final, a latência é o parâmetro de desempenho mais importante num sistema cliente/servidor, em particular quando as requisições são serializadas (i.e. a próxima requisição depende do resultado da anterior). Em todos os testes deste capítulo, a unidade de latência é o microssegundo (μs).

Em alguns testes, onde a latência de rede foi objeto de variação, a latência apurada no teste foi denominada **latência de transação**, pois ela simula uma transação cliente/servidor. Usou-se pontualmente esta denominação para evitar qualquer confusão entre as duas grandezas.

5.2. Aplicativos construídos para o teste

Os aplicativos utilizados nos testes deste capítulo foram escritos pelo próprio autor, nas linguagens C e C++, e seu código-fonte pode ser encontrado no CD anexo à dissertação.

Todos os aplicativos descritos abaixo permitem variar o tamanho das mensagens transmitidas, número de fluxos utilizados em SCTP, endereços para associação (*bind*) e o volume total trafegado em *bytes*.

Exceto quando explicitamente afirmado o contrário, todos os testes utilizaram os soquetes “estilo TCP”. A API *BSD/Sockets* oferece acesso ao SCTP através de soquetes “estilo TCP” e “estilo UDP”. Para maiores informações, consulte o ANEXO 3.

Por padrão, os aplicativos trafegam 100 milhões de *bytes*, em cada direção, por rodada. Em algumas situações, quando o cenário simulou uma rede muito lenta, e o uso de um volume menor de dados não prejudicaria a qualidade das amostras, esse volume foi reduzido para diminuir o tempo necessário à execução do teste.

TCPM: Separação de mensagens para TCP

Os aplicativos de teste permitem utilizar diversos protocolos de rede: TCP, SCTP e soquetes UNIX. No caso do TCP, pode-se utilizá-lo “puro”, ou empilhado com um protocolo de aplicação denominado TCPM. O TCPM foi criado especialmente para os fins deste trabalho, e possibilita a separação de mensagens na camada de aplicação. Naturalmente, isto aumenta a sobrecarga de processamento, portanto o TCPM terá desempenho menor que o TCP puro dado o mesmo cenário de rede.

O principal motivo de introduzir essa codificação/decodificação de mensagens em TCP foi tornar a comparação mais justa com SCTP, onde esse custo está sempre presente. O protocolo TCPM é o mais simples possível: um *byte* marcando o início da mensagem, outro marcando o final. Qualquer protocolo real seria mais complexo e custaria mais processamento.

Para protocolos de aplicação que não separam mensagens (e.g. conexão de dados FTP), ou usem mensagens arbitrariamente longas que obriguem a separação na camada de aplicação, impor essa carga extra ao TCP seria injusto pois ela não existe na prática. Porém, para esses protocolos, o TCP é uma escolha mais adequada que SCTP. Os testes visam protocolos de aplicação onde sejam aplicáveis os recursos novos do SCTP.

Os testes com soquetes UNIX também têm a opção de utilizar o transporte puro, ou empilhado com o protocolo de aplicação UNIXM. O UNIXM é idêntico ao TCPM, possuindo um nome diferente apenas para melhor identificar o transporte.

Testes de vazão

Os aplicativos construídos para o teste de vazão enviam um certo volume de *bytes*, divididos em mensagens individuais. O envio acontece nas duas direções; se o volume total configurado for e.g. 10MBytes, o volume total trafegado em ambas as direções será 20MBytes.

Os aplicativos não tentam sincronizar de qualquer forma os dois fluxos (exceto *udp_server2*, vide abaixo); é perfeitamente possível que uma direção termine antes que outra, prejudicando o resultado final do teste. Isto é propositado – um bom protocolo de transporte deve permitir uma comunicação *full-duplex* equilibrada, e o teste de vazão testa (indiretamente) esse equilíbrio.

Segue a lista de utilitários de teste de vazão:

tcp_server: Servidor que utiliza soquetes estilo TCP. Permite usar os protocolos "soquete UNIX", TCP e SCTP. Para soquetes UNIX e TCP, ainda há a opção de codificar as mensagens (UNIXM e TCPM) de modo que sejam separáveis na camada de aplicação, como seria normalmente feito na maioria dos protocolos de aplicação reais.

tcp_client: Lado cliente do teste, com as mesmas características de *tcp_server*.

udp_server: Servidor SCTP que faz uso de soquetes estilo UDP.

udp_server2: Implementação alternativa de *udp_server* em C++, que usa um identificador opaco da associação ao invés do endereço, sendo assim mais correta para uso com multicaminhos em lugar de *udp_server*. O C++ foi utilizado pois a biblioteca STL facilitou a implementação da lista de clientes ativos. Por razões de lógica interna, este servidor impõe uma vazão uniforme nos 2 sentidos.

udp_client: Cliente que utiliza protocolo SCTP e soquetes estilo UDP.

Os testes de vazão podem ser executados tanto com TCP e soquetes UNIX puros (sem separação de mensagens), quanto com TCPM e UNIXM (com separação de mensagens). Nos cenários de *loopback* e rede *100Mbps*, ambas as possibilidades foram utilizadas para efeitos de comparação.

Testes de latência

Os aplicativos para teste de latência, da mesma forma que os testes de vazão, trafegam um dado volume em ambas as direções, dividido em mensagens de tamanho fixo. A diferença é que, para cada mensagem enviada pelo cliente, exatamente uma é retornada pelo servidor, simulando assim um sistema cliente/servidor típico.

O lado cliente é responsável por computar a latência total entre a remessa da pergunta e o retorno da resposta. O tempo fora desse intervalo não é computado. Cada transação é cronometrada utilizando-se a chamada POSIX *gettimeofday()* e o valor retornado ao final é a média das latências.

A chamada *gettimeofday()* tem precisão de μs em computadores e sistemas suficientemente recentes. O tempo consumido pela chamada foi considerado negligenciável frente às latências a serem mensuradas. No computador mais rápido do laboratório de testes, a latência da chamada foi de apenas 0,25μs, e o intervalo mais curto mensurado nos testes de rede foi de 18μs, portanto a sobrecarga é de apenas 3% no pior caso (considerando duas chamadas por medição).

Os aplicativos criados para o teste foram: *tcp_server_lat*, *udp_server_lat*, *udp_server2_lat*, *udp_client_lat* e *tcp_client_lat*. Eles compartilham das mesmas características dos aplicativos correspondentes para medida de vazão. O servidor *tcp_server_lat* impõe o uso dos protocolos de aplicação TCPM e UNIXM, pois é necessário ter meios de separar a mensagem para determinar se ela chegou completamente.

Coleta de dados

Nos testes de vazão, os valores exibidos nos gráficos e tabelas deste trabalho são médias de 4 passagens. Foram feitas 5 rodadas de execução por teste; o primeiro resultado é descartado e os demais compõem a média.

Nos testes de latência, cujo valor retornado já é uma média que converge rapidamente, apenas duas rodadas foram executadas. O valor contido nos gráficos e tabelas deste trabalho é a média das 2 rodadas.

O resultado das passagens individuais, média e desvio padrão podem ser encontrados na planilha *amostras.xls* do CD que acompanha este trabalho.

5.3. Sistema operacional e implementação do SCTP

O sistema operacional utilizado foi o Linux, versão de *kernel* 2.6.10rc1 (*kernel* 2.6.9 atualizado com algumas mudanças que foram incorporadas ao 2.6.10 final). Foi o *kernel* estável mais recente disponível à época dos testes. O protocolo SCTP sofreu atualizações importantes até a versão 2.6.8, como o suporte a PR–SCTP. A implementação do SCTP utilizada é a LK–SCTP *(Linux Kernel SCTP)*, incluída nas séries 2.4 e 2.6 do *kernel* do Linux.

Exceto onde explicitamente afirmado o contrário, o sistema operacional, o TCP e o LK–SCTP foram utilizados com suas configurações *default*, sem qualquer otimização

de parâmetros. Presume-se que a maioria dos usuários use um sistema da mesma forma, sem qualquer otimização, e que uma configuração *default* sub-ótima seria simplesmente percebida como um problema inerente.

A importância dos aplicativos *user-level* é reduzida nos testes deste trabalho pois o desempenho depende majoritariamente do *kernel*. A distribuição utilizada foi o Conectiva Linux 10. O *software user-level* mais relevante é o compilador C. A distribuição fornece o GCC versão 3.3.3.

A implementação está sendo desenvolvida desde 2001, e já foi objeto de vários testes independentes. Possui inclusive sua própria biblioteca de testes de regressão, que verificam, a cada lançamento de nova versão, se todos os aspectos do protocolo estão funcionando corretamente. Porém, a maturidade será atingida apenas quando o protocolo for largamente usado por aplicações reais.

Até a data desta dissertação (2004), o principal objetivo da equipe LK–SCTP é produzir uma implementação correta e completa (a versão mais recente suporta PR–SCTP e AddIP). O desempenho, embora já tenha melhorado bastante desde ao longo do tempo, ainda tem prioridade mais baixa.

O LK–SCTP não faz qualquer esforço em otimizar o consumo de memória, para o caso de aplicações que reservem um grande número de fluxos mas deixe-os quase todos ociosos. Um dos objetivos deste trabalho era propor tal otimização, utilizável de forma opcional através de interface *setsockopt()*. A adaptação do HTTP ao SCTP acabou revelando que tal otimização seria de pouca utilidade (vide 6.2.1).

A deficiência mais flagrante da implementação (pelo menos até a última versão do LK-SCTP analisada) é alocar memória em blocos muito grandes para os SSN dos fluxos – até 256KB numa associação com 65536 fluxos em cada direção, usando interfaces obsoletas (*kmalloc()* e *__get_free_pages()*). Tais interfaces alocam blocos contínuos na RAM física. Em situações de RAM escassa, as páginas livres provavelmente não estarão agrupadas, e alocações de grandes áreas contínuas não serão satisfeitas.

A interface atualmente recomendada para alocação dinâmica dentro do *kernel* do Linux é o *slab cache*, com unidades de alocação não maiores que uma página de RAM da arquitetura. Não tão simples como alocar áreas contínuas, mas garante o bom funcionamento em situações-limite. Alocação de grandes áreas contínuas deve restringir-se aos *drivers* de dispositivo que tenham essa necessidade.

5.4. Ferramentas de simulação de rede

Para modelar redes com características reais, com latência variável e perda de pacotes, foram utilizadas algumas ferramentas do Linux: os recursos de roteamento avançado ***netem*** e *filter*, e o *firewall* IPTables.

Figura 5.1: Exemplo de *script* para simulação de rede com latência e perda

```
tc qdisc del dev eth0 root
tc qdisc del dev eth0 ingress
# simulação de latência e duplicaçâo é na saída (egress)
tc qdisc add dev eth0 root handle 1:0 netem delay 30ms duplicate 0.1%
tc qdisc add dev eth0 handle ffff: ingress
# simulação da banda, no caso 1Mbps, é na entrada (ingress)
# o excesso de banda é descartado (drop)
tc filter add dev eth0 parent ffff: protocol ip prio 1 \
       u32 match ip src 0.0.0.0/0 \
       police rate 1Mbit burst 100k drop \
       flowid :1
tc -s -d qdisc ls dev eth0
# perda de 0,1% constante
# random so aceita percentagens inteiras, então usamos 2 filtros
# para simular uma perda com decimais
iptables -F
iptables -N PASSODOIS
iptables -A INPUT -m random --average 1 -j PASSODOIS
iptables -A PASSODOIS -m random --average 10 -j DROP
iptables -A PASSODOIS -j ACCEPT
iptables -P INPUT ACCEPT
```

A disciplina de tráfego *netem* (SHREMMINGER) possibilita introduzir latência, perda, duplicação e reordenamento de pacotes na saída de uma interface de rede. No estado atual desta ferramenta (*kernel* 2.6.10rc1), os recursos de perda de pacotes e reordenamento ainda têm problemas e não podem ser usados para simulação.

O gerenciamento de fila *filter* foi utilizado para simular um canal com largura de banda reduzida, que congestiona e perde pacotes ao ser saturado. Nos testes deste trabalho ele foi posicionado na entrada de pacotes do receptor (*ingress shaping*), pois simula melhor uma rede congestionada. Se fosse posicionado na saída, os pacotes nem chegariam a ocupar largura de banda real. Além disso, os protocolos de transporte do Linux detectam e reagem imediatamente a pacotes descartados logo de saída (seja por *shaping* ou por *firewall*), o que distorceria os resultados.

Finalmente, o *firewall* IPTables foi utilizado para simular perda de pacotes (remendando assim a insuficiência do *netem*) e a queda de *link* para testes com multicaminhos. O recurso de perda aleatória de pacotes não está disponível por padrão na maioria das distribuições Linux, mas é prontamente disponível como código-fonte e foi incorporado ao *kernel* utilizado nos testes.

A perda total de pacotes do canal será a perda constante simulada no IPTables, mais a perda variável simulada pelo *ingress shaping* em função da saturação do canal.

A figura 5.1 mostra um exemplo de *script* destinado à simulação de rede com latência e perda. O *script* deve ser executado nos dois computadores terminais envolvidos no teste. Em nenhum dos testes foi necessário executar *scripts* em roteadores intermediários.

5.5. Computadores utilizados nos testes

SOLDIER: Athlon XP 1800+, placa-mãe ASUS A7V5X (*chipset* VIA KT266), 512MB RAM DDR 266MHz, uma placa de rede RTL 8139 100Mbps *on-board*.

COP: Athlon *Thunderbird* 1200 MHz, placa-mãe ASUS A7V (*chipset* VIA KT133), 512MB RAM 133MHz, duas placas de rede RTL 8139 100Mbps , uma placa de rede RTL 8029 PCI 10Mbps.

ARP: Pentium II 400MHz, placa-mãe Intel SE440BX2, 256MB RAM 100MHz, uma placa de rede RTL 8139 100Mbps, uma placa de rede PCCard Avaya Silver *wireless* 802.11b 11Mbps, adaptador PCI–PCMCIA Ricoh.

BIKER: Pentium II 233MHz, *chipset* de placa-mãe SE440BX2 para *notebook*, 160MB RAM 100MHz, uma placa de rede PCCard 3Com 3C589 Combo 10Mbps, uma placa de rede PCCard Avaya Silver *wireless* 802.11b 11Mbps.

Os computadores SOLDIER e COP foram os mais freqüentemente utilizados nos testes, pois eram os mais rápidos, o que minimiza a deficiência das placas de rede de baixo desempenho.

5.6. Cenários de rede

Interface de loopback

Na interface de *loopback* (endereço IP 127.0.0.1), não há perda ou latência de comunicação de rede; o único "custo" do tráfego é o consumo de CPU dos protocolos de rede, transporte e aplicação. Os protocolos com melhor performance são os que consomem menos CPU, e eventualmente mais otimizados em detalhes de baixo nível (como *cache hit* de CPU).

O soquete UNIX foi incluído neste teste pois é um mecanismo de comunicação interprocessos sem qualquer *overhead* de rede, e portanto deve indicar a maior performance possível no equipamento utilizado.

Os testes de *loopback* variaram o tamanho das mensagens trocadas pelos aplicativos de teste (de 10 a 10000 *bytes*) e o número de fluxos SCTP (de 1 a 10000 fluxos).

O computador utilizado nestes testes foi o SOLDIER (Athlon XP 1800+).

Rede Ethernet 100Mbps

Foi utilizada uma rede Ethernet 100Mbps com *switch*, sem a introdução de nenhuma perda ou latência artificial. Os testes variaram o tamanho das mensagens trocadas entre os participantes, de 10 a 10000 *bytes*.

A máquina COP (Athlon 1.2GHz) foi utilizada como servidor e a máquina SOLDIER (Athlon XP 1800+) como cliente.

Rede Ethernet 100Mbps com perda de pacotes

Foi utilizada como base uma rede Ethernet 100Mbps com *switch*. Neste teste, foi introduzida uma perda artificial de pacotes, na recepção (*input*). Assim, o pacote perdido ocupa largura de banda. O objetivo é testar a adaptabilidade dos protocolos a redes com perdas.

Os aplicativos usaram sempre mensagens de 500 *bytes* de tamanho. Os testes foram feitos com as seguintes taxas de perda de pacotes: 0%, 0,1%, 1%, 3% e 10%.

A máquina COP (Athlon 1.2GHz) foi utilizada como servidor e a máquina SOLDIER (Athlon XP 1800+) como cliente.

Rede Ethernet 10Mbps com latência

Foi utilizada uma rede Ethernet 100Mbps com *switch*. De modo a limitar a vazão em 10Mbps sem recorrer a *traffic shaping* por *software*, a comunicação ocorreu através de uma placa de rede 10Mbps instalada no computador COP.

O tamanho das mensagens utilizado pelos aplicativos de teste foi de 500 *bytes*. A latência de rede foi variada entre 5ms e 300ms, o que corresponde a RTTs de 10ms a 600ms. O objetivo deste teste é medir a influência da latência de rede no desempenho dos protocolos.

A máquina COP (Athlon 1.2GHz) foi utilizada como servidor e a máquina SOLDIER (Athlon XP 1800+) como cliente.

Rede Ethernet 10Mbps com perda de pacotes e latência

Foi utilizada uma rede Ethernet 100Mbps com *switch*. De modo a limitar a vazão em 10Mbps sem recorrer a *traffic shaping* por *software*, a comunicação ocorreu através de uma placa de rede 10Mbps instalada no computador COP. Neste teste também foi introduzida uma perda constante de 1% dos pacotes em ambas as direções.

O tamanho das mensagens utilizado pelos aplicativos de teste foi de 500 *bytes*. A latência de rede foi variada entre 5ms e 300ms, o que corresponde a RTTs de 10ms a 600ms. O objetivo deste teste é medir a influência da latência de rede no desempenho dos protocolos, na presença de uma perda constante.

A máquina COP (Athlon 1.2GHz) foi utilizada como servidor e a máquina SOLDIER (Athlon XP 1800+) como cliente.

Rede 1Mbps com latência e duplicação de pacotes

Como base, foi utilizada uma rede Ethernet 100Mbps com *switch*. A largura de banda de 1Mbps foi simulada através de *traffic shaping*, que simula um congestionamento de rede e descarta pacotes que ultrapassarem a velocidade determinada. Assim, este cenário tem perda de pacotes, mas esta perda é variável, e o controle de congestionamento do protocolo de transporte tende a zerá-la.

Também em *software* foi simulada uma latência variável de 50ms +/- 25ms (correlação de 50% entre latências seguidas) e 1% de duplicação de pacotes.

O tamanho das mensagens utilizado pelos aplicativos de teste foi de 500 *bytes*. A máquina COP (Athlon 1.2GHz) foi utilizada como servidor e a máquina SOLDIER (Athlon XP 1800+) como cliente.

Rede 1Mbps com latência, perda e duplicação de pacotes

Como base, foi utilizada uma rede Ethernet 100Mbps com *switch*. A largura de banda de 1Mbps foi simulada através de *traffic shaping.* Outras condições simuladas: latência variável de 50ms +/- 25ms (correlação de 50% entre latências seguidas), 1% de duplicação de pacotes, perda constante de 0,1% de pacotes.

O tamanho das mensagens utilizado pelos aplicativos de teste foi de 500 *bytes*. A máquina COP (Athlon 1.2GHz) foi utilizada como servidor e a máquina SOLDIER (Athlon XP 1800+) como cliente.

Rede 11Mbps wireless 802.11b *ad-hoc*

Foi utilizada uma rede *wireless* 802.11b 11Mbps *ad-hoc* entre os computadores BIKER e ARP. Uma rede Ethernet de 100Mbps com *switch*, sem qualquer perda ou latência artificiais, foi utilizada entre os computadores ARP e SOLDIER. A vazão total é limitada pelo enlace de menor capacidade no caminho; a latência de rede total é soma da latência das duas redes, mais aquela eventualmente introduzida pelo roteador.

Os pontos *wireless* ficaram propositadamente distantes, de modo a fornecer um canal de qualidade sofrível. A vazão bruta unidirecional, medida experimentalmente, foi de 3Mbps. A rede 802.11b não faz comunicação *full-duplex.*

O tamanho das mensagens utilizado pelos aplicativos de teste foi de 500 *bytes*. A máquina BIKER (Pentium II/400 MHz) foi utilizada como cliente, e a máquina SOLDIER (Athlon XP 1800+) como servidor. A máquina ARP operou como roteador intermediário.

Rede Ethernet 100Mbps, um servidor, dois clientes simultâneos

Tabela 5.1: Vazão em kbps, conexões *loopback*

Protocolo / tamanho msg. em *bytes*	10	100	1000	10000
Unix	8900	93870	718602	2452172
UnixM	4980	43640	218084	438726
TCP	11644	115320	643522	1030614
TCPM	1802	16030	128220	265872
SCTP	1384	15842	108626	255372
SCTP estilo UDP	1334	15584	109066	228040
SCTP estilo UDP v. 2	1344	13160	98902	282236

Foi utilizada uma rede Ethernet 100Mbps com *switch*, sem a introdução de nenhuma perda ou latência artificial.

O tamanho das mensagens utilizado pelos aplicativos de teste foi de 500 *bytes*. As máquinas COP e SOLDIER foram utilizadas como clientes, e a máquina ARP como servidor.

Multicaminhos 10Mbps e 11Mbps

O caminho primário de rede é de 10Mbps Ethernet (a placa de rede do computador BIKER é de 10Mbps). O caminho secundário é 802.11b *ad-hoc*, 11Mbps. A máquina ARP operou como roteador intermediário no caminho secundário.

Nos testes de multicaminhos, o tráfego do *link* primário é liberado por 10 segundos, a seguir bloqueado por 10 segundos. Esse ciclo é mantido continuamente. Foram usadas regras de *firewall* e de roteamento para evitar roteamento assimétrico em qualquer dos dois estados. Nos testes dos caminhos individuais, o tráfego é contínuo.

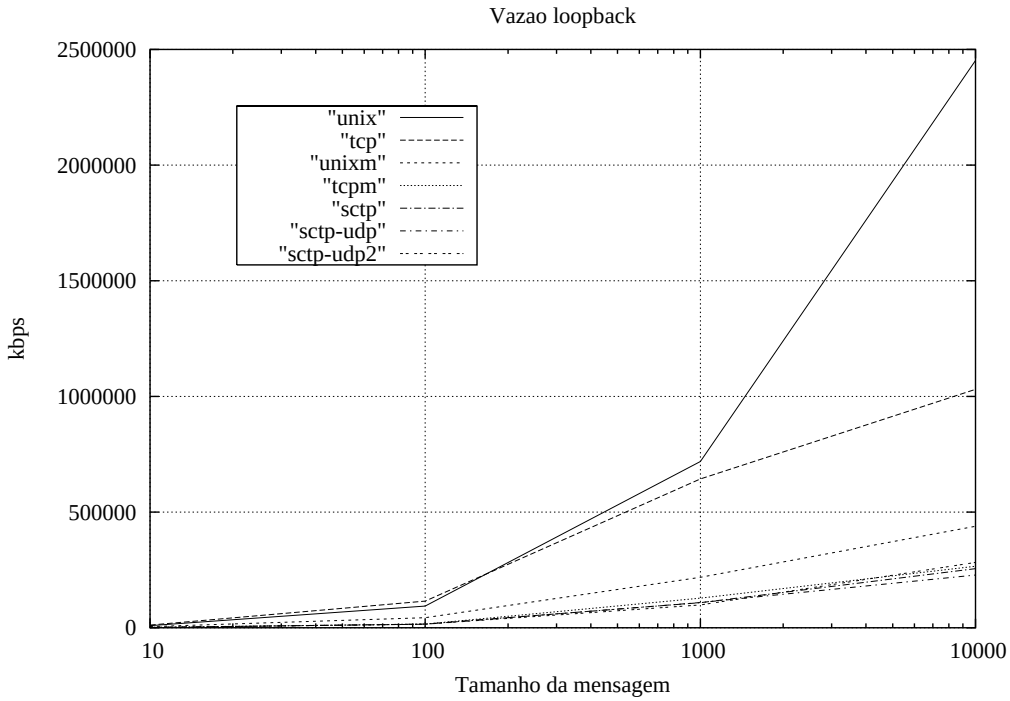
O tamanho das mensagens utilizado pelos aplicativos de teste foi de 500 *bytes*. A máquina BIKER (Pentium II/233 MHz) foi utilizada como servidor, e a máquina COP (Athlon 1.2GHz) como cliente.

5.7. Resultados

5.7.1. Interface de *loopback*

A figura 5.2 e a tabela 5.1 exibem os resultados do teste de vazão. Como é de se esperar, o TCP teve performance muito superior ao SCTP. Reputa-se isso ao cálculo do somatório CRC-32c que ocorre a cada pacote em SCTP, e a compulsória separação de

Figura 5.2: Vazão em conexões *loopback*



mensagens que obriga uma troca de contexto no receptor para cada mensagem. Por outro lado, quando foi imposta a separação de mensagens para TCP e soquetes UNIX (TCPM e UNIXM), o desempenho de todos os protocolos ficou bastante parecido.

Um teste posterior, feito pelo autor, desligando-se o cálculo do CRC-32c no *kernel* do Linux, mostrou que o ganho de desempenho é surpreendentemente pequeno (10% a 15% em *loopback*). Isto ainda deve ser objeto de estudo mais cuidadoso, mas indica que *a priori* o grande responsável pelo menor desempenho do SCTP é a separação de mensagens.

O uso de soquetes estilo TCP ou UDP nos programas SCTP não influencia sensivelmente na performance. O melhor resultado do SCTP estilo UDP versão 2 é devido ao sincronismo que este impõe sobre as duas direções de transmissão. Os demais servidores SCTP tenderam a perder o sincronismo (i.e. uma direção terminava de transmitir muito antes que a outra).

Uma surpresa do teste, não relacionada com o SCTP, foi o TCP ter vazão maior que soquetes UNIX para mensagens pequenas.

No teste de latência, trocas de contexto têm de seguir uma ordem estrita: processo-cliente, *kernel*, processo-servidor, *kernel*, processo-cliente. Todos os *overheads* (*bytes* de controle, custo de CPU, ineficiências da implementação) ficam serializados e evidenciados.

Figura 5.3: Latência em conexões *loopback*

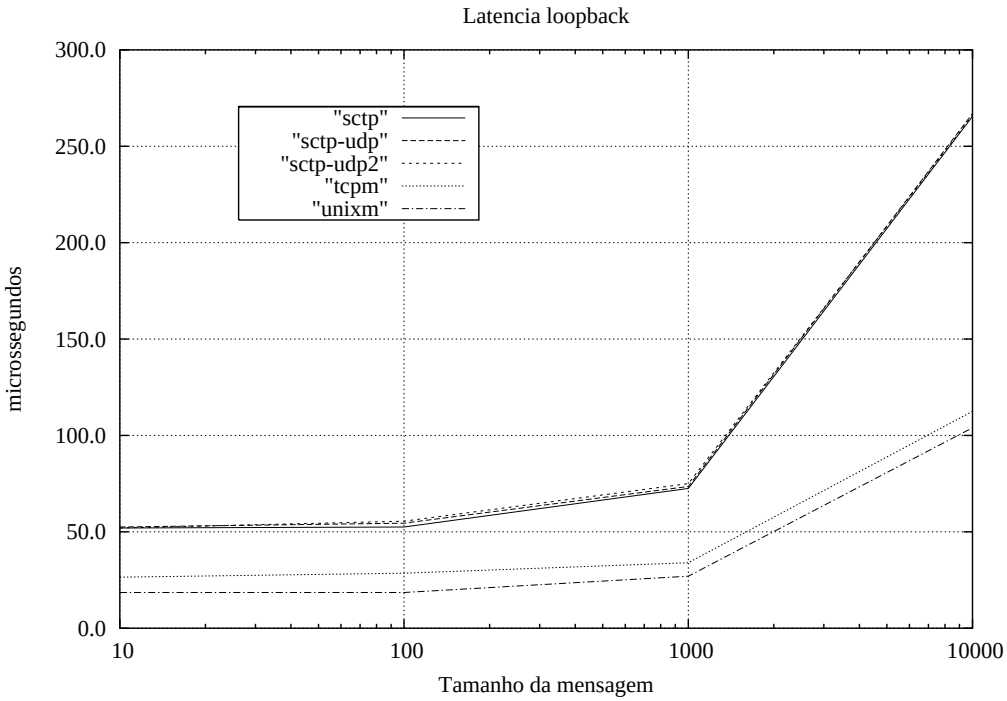


Tabela 5.2: Latência em μs, conexões *loopback*

Protocolo / tamanho msg. em *bytes*	10	100	1000	10000
UnixM	18,5	18,5	27,0	104,0
TCPM	26,5	28,5	34,0	112,5
SCTP	52,0	52,5	72,5	265,5
SCTP estilo UDP	52,5	54,5	73,5	267,0
SCTP estilo UDP v. 2	52,0	55,5	75,0	266,0

Tabela 5.3: Vazão e latência em conexões *loopback*, em função do número de fluxos

Número de fluxos	Vazão (kbps)	Latência (μs)
1	16140	51,0
10	16244	51,0
100	16128	53,0
1000	16136	53,5
10000	16036	51,5

Conforme a figura 5.3 e a tabela 5.2, o SCTP apresentou performance aproximadamente duas vezes menor que TCPM, não importa tamanho da mensagem.

A razão mais provável para esse desempenho inferior do SCTP é o grande número de cópias de memória executado pela implementação. Idealmente, um protocolo de transporte deveria fazer uma, ou até mesmo zero cópias de memória. Este ideal é mais difícil para o SCTP devido a suas características inerentes (CRC-32c, separação de mensagens etc.)

O uso de soquete estilo TCP ou UDP não fez qualquer diferença ao SCTP. Desta forma, a escolha de um ou outro estilo pode atender unicamente à conveniência do desenvolvedor, não havendo nenhum efeito colateral na performance.

Para testar a vazão e latência SCTP em função do número de fluxos, foram utilizadas mensagens de 100 *bytes* neste teste. A variação em função do número de fluxos é praticamente nula, como mostra a tabela 5.3.

Um teste levado a cabo por KANG & FIELDS demonstrou que a vazão do SCTP é maior quando utilizam-se diversos fluxos em paralelo, contrariando o resultado do teste deste trabalho.

Mas o teste de KANG & FIELDS pressupôs que o protocolo de aplicação utiliza simultaneamente todos os fluxos, e pode receber as mensagens de forma desordenada (i.e. sem poder prever de que fluxo viria a próxima mensagem). Isto seria equivalente a marcar todas as mensagens como “urgentes”, de modo que sua entrega não obedecesse qualquer ordem.

Os testes deste capítulo pressupõem que o protocolo de aplicação exige um transporte confirmado tal qual TCP, e deste modo não podemos aproveitar a entrega desordenada como estratégia de aumento de desempenho. Nos testes com o protocolo HTTP, onde diversos fluxos são usados para transportar uma única transação (página HTTP com os respectivos objetos acessórios), será então possível colher ganhos de desempenho com esta técnica.

Figura 5.4: Vazão, rede 100Mbps

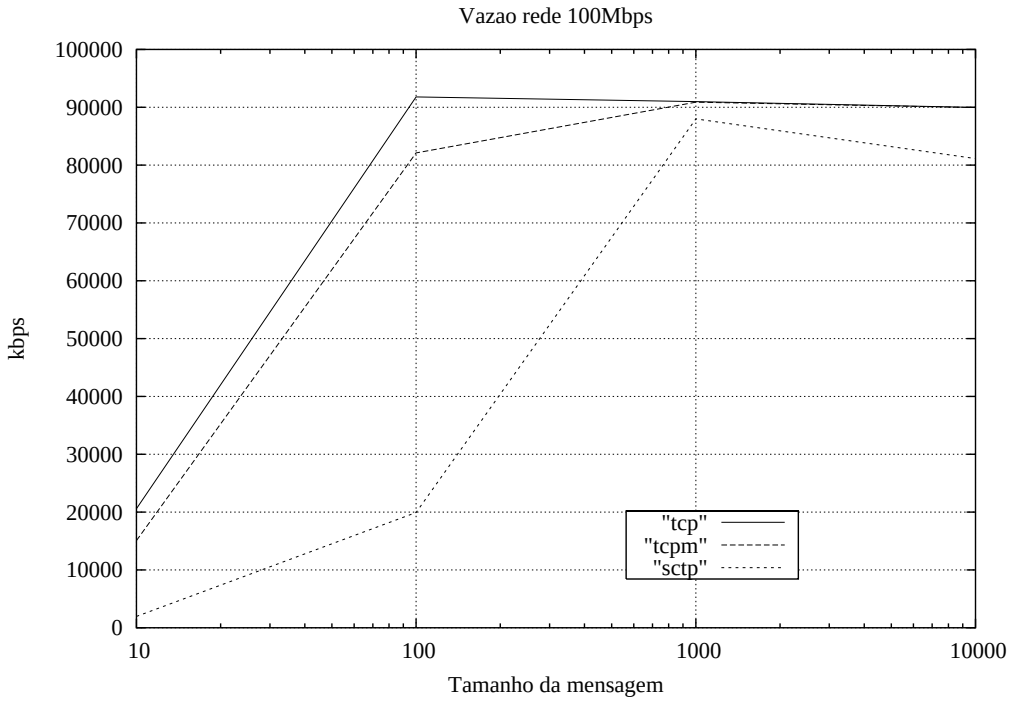


Tabela 5.4: Vazão em kbps, rede 100Mbps

Protocolo / tamanho msg. em *bytes*	10	100	1000	10000
TCP	20578	91782	90984	89972
TCPM	15076	82114	90886	89954
SCTP	1952	19926	87994	81118

5.7.2. Rede 100Mbps

Os resultados de vazão estão evidenciados na figura 5.4 e na tabela 5.4. Com mensagens grandes, a vazão tende ao máximo permitido pela rede em todos os protocolos.

A vazão do SCTP é menor pois o protocolo tem maior *overhead* inerente. Na medida em que as mensagens diminuem de tamanho, o *overhead* aumenta em importância relativa, e o desempenho do SCTP fica cada vez mais prejudicado.

Uma análise do tráfego de rede também revelou que o *chunk bundling* (combinação de diversos trechos de dados em um único pacote) não está otimamente implementado no LK–SCTP. A implementação empregou um pacote para cada mensagem, como se o algoritmo Nagle (NAGLE) estivesse desligado.

A grande diferença em desfavor do SCTP, no teste de latência com *loopback*, foi mascarada pela latência da rede, conforme mostram a tabela 5.5 e a figura 5.5. Inte-

Figura 5.5: Latência, rede 100Mbps

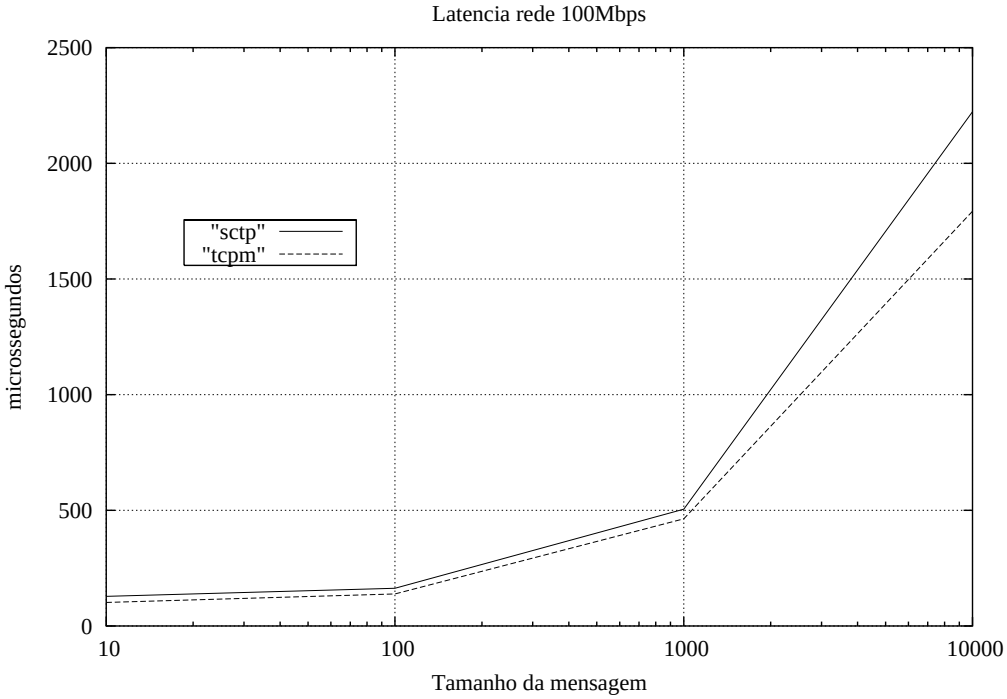


Tabela 5.5: Latência em μs, rede 100Mbps

Protocolo / tamanho msg. em *bytes*	10	100	1000	10000
TCPM	101.5	138.5	463.0	1793.0
SCTP	128.0	163.0	505.0	2223.0

Figura 5.6: Vazão em kbps, rede 100Mbps com perda de pacotes

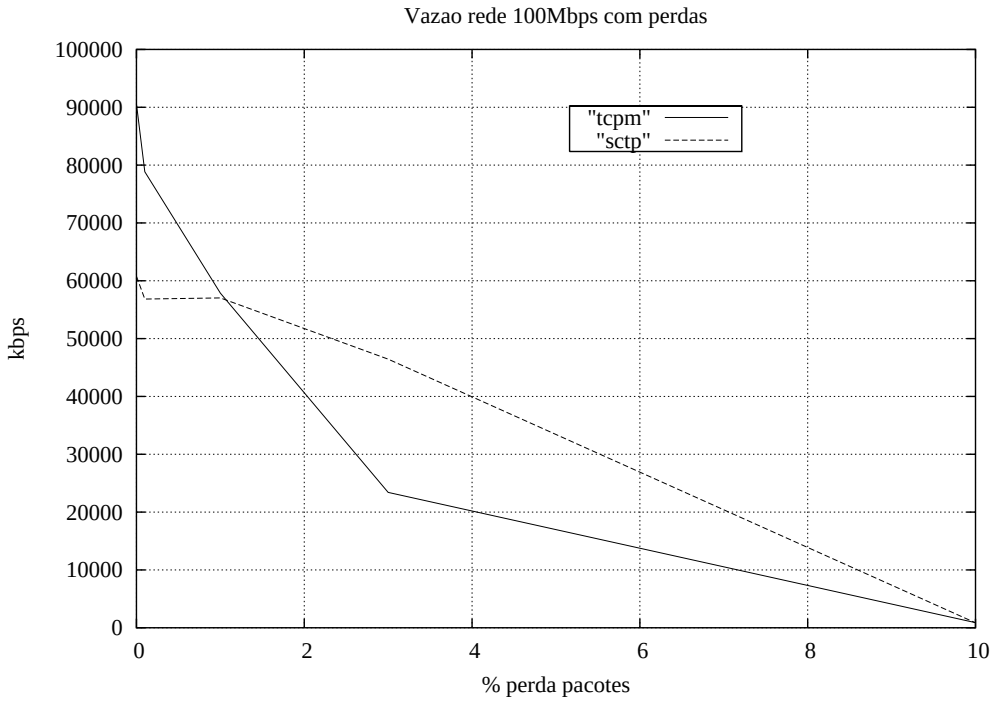


Tabela 5.6: Vazão (em kbps), rede 100Mbps com perda de pacotes

Protocolo / perda %	0%	0,1%	1%	3%	10%
TCPM	90596	78846	57876	23396	878
SCTP	60773	56834	57030	46438	852

ressante notar que a menor diferença ocorreu quando tamanho da mensagem tendeu ao tamanho do pacote de rede (máxima diluição da sobrecarga de dados).

5.7.3. Rede 100Mbps com perda de pacotes

A tabela 5.6 e a figura 5.6 evidenciam que, na ausência de perdas, a vazão do SCTP foi bastante menor que TCP, resultado congruente com a tabela 5.4 do teste anterior levando-se em conta que o tamanho de mensagem empregado foi de 500 *bytes*.

Na medida em que aumentam as perdas, a vazão naturalmente cai. E então o SCTP conseguiu manter uma vazão maior em perdas moderadas (1% a 3%).

Nenhuma otimização foi tentada nem em SCTP nem em TCP neste teste. As implementações de ambos no Linux permitem regular alguns parâmetros do controle de congestionamento, mas foram simplesmente utilizados os valores padrão. Possivelmente, o desempenho de ambos, em particular do TCP, podem ser melhorados se esses parâmetros forem atuados.

Figura 5.7: Latência, rede 100Mbps com perda de pacotes

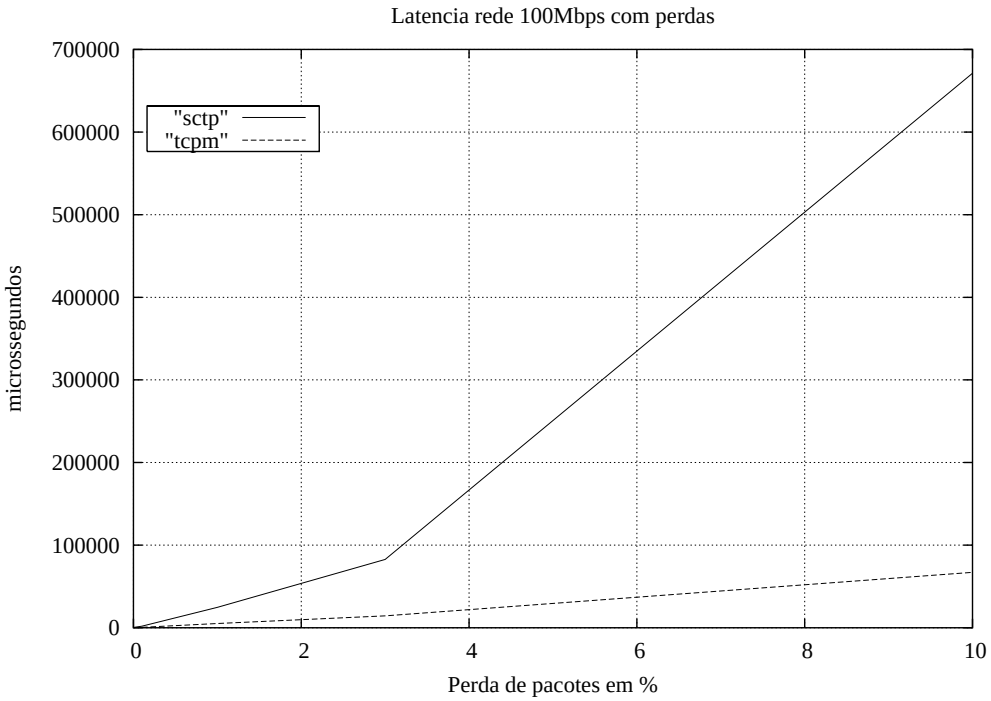


Tabela 5.7: Latência em μs, rede 100Mbps com perda de pacotes

Protocolo / perda %	0%	0,1%	1%	3%	10%
TCPM	281,0	483,9	5190,0	14418,5	67097,0
SCTP	318,5	1920,0	24662,0	82718,0	671100,5

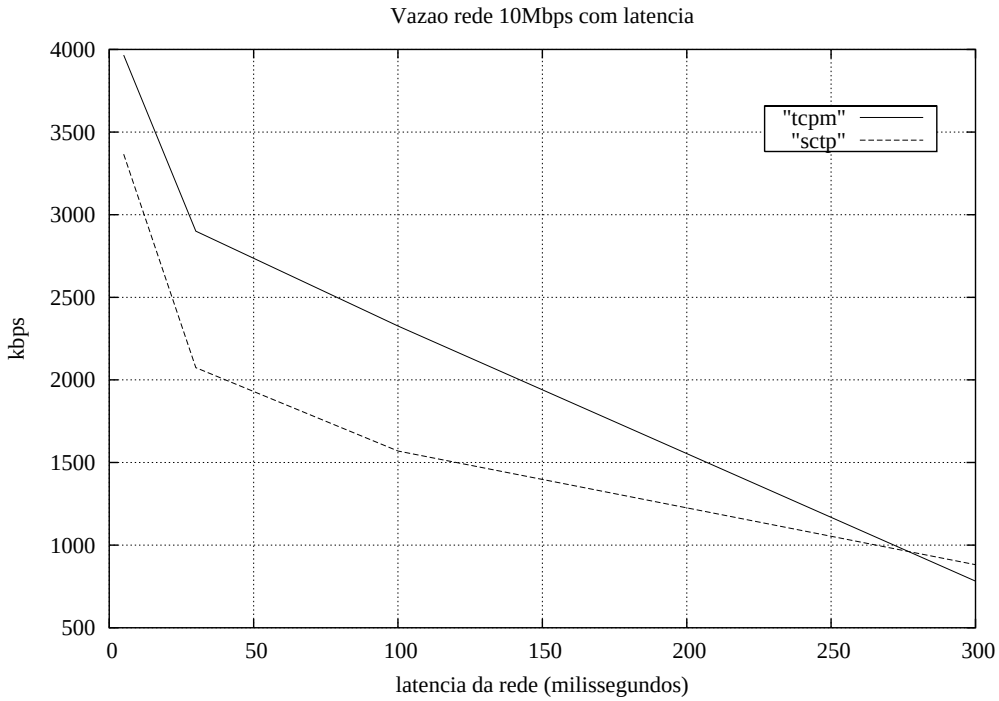
O SCTP beneficiou-se ainda de uma característica do aplicativo construído para os testes de vazão: pacotes são continuamente mandados em ambas as direções, o que permite ao SACK atuar com presteza.

Apesar de ter ido bem no teste de vazão deste cenário, o SCTP teve péssimo desempenho no teste de latência, conforme mostram a figura 5.6 e a tabela 5.7. O principal fator desse resultado ruim é a configuração *default* do controle de congestionamento do SCTP.

O RTO (*timeout*) mínimo padrão do LK–SCTP é de 1 segundo; isso é aproximadamente 5000 vezes o RTT da Ethernet 100Mbps. Portanto, toda perda causa um atraso de no mínimo 1 segundo na transação, elevando muito a média geral. O TCP demonstrou não ter limite mínimo embutido de RTO; ele será proporcional à latência da rede real, e portanto adaptar-se-à corretamente.

Quando o protocolo de aplicação troca pacotes continuamente nas duas direções, o SACK do SCTP tem a oportunidade de notificar a perda imediatamente, e o transmissor toma providências antes do RTO. Por esse motivo o SCTP foi bem melhor no teste de vazão que no teste de latência.

Figura 5.8: Vazão em kbps, rede 10Mbps com latência de rede



O problema detectado no teste de latência deste cenário em particular foi o mais grave encontrado no LK–SCTP. O SCTP tem uma extensão opcional para links de satélite, onde perdas severas e constantes são esperadas, mas o LK–SCTP não oferece essa extensão, e não foi possível testá-la.

O RTO mínimo pode ser regulado no LK–SCTP, porém das duas tentativas de resolver o problema por esse caminho, apenas uma foi bem-sucedida, apesar do cenário de teste ser rigorosamente o mesmo. Na tentativa bem-sucedida, a latência descia a patamares semelhantes ao TCP, mas na tentativa frustrada, a associação acabava estolando após um certo volume trafegado.

Mais testes são necessários para verificar se o controle de congestionamento do SCTP tem alguma instabilidade associado a perdas constantes de pacotes de rede.

De qualquer forma, com ou sem instabilidade, o RTO mínimo padrão de 1 segundo, apesar de estar em conformidade com a RFC 2960, é excessivo para uso no mundo real.

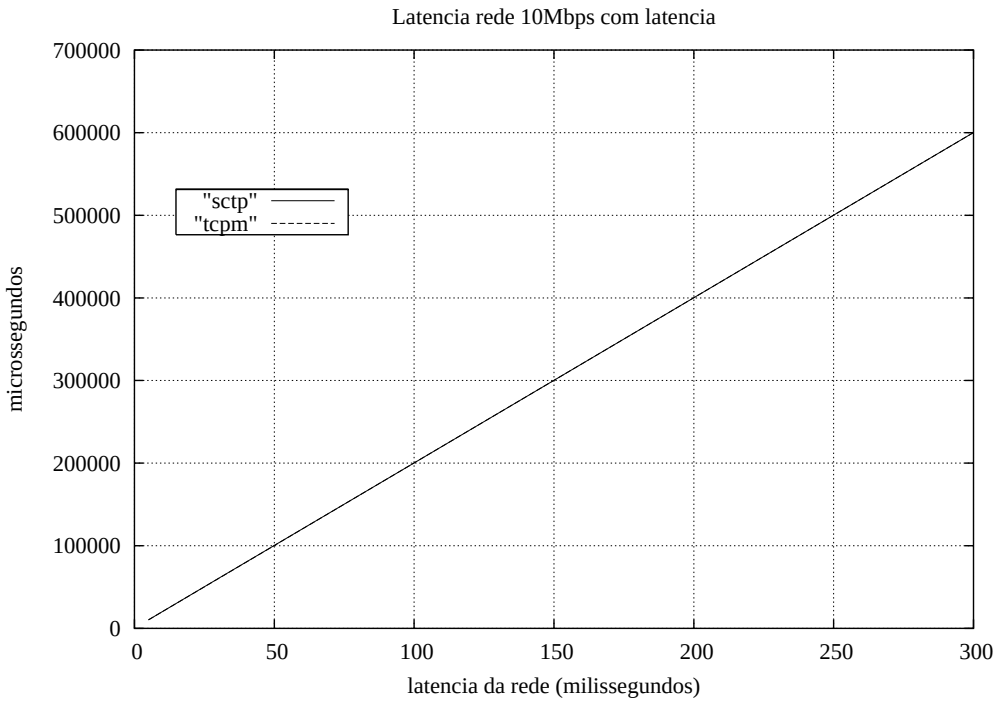
5.7.4. Rede 10Mbps com latência

Na ausência de perdas, a vazão de um protocolo confirmado deveria ser independente da latência da rede. Mas a figura 5.8 evidencia o contrário, tanto para TCP quanto para SCTP. A razão é que o aplicativo construído para o teste usa *buffers* de recepção

Tabela 5.8: Vazão em kbps, rede 10Mbps com latência de rede

Protocolo / latência de rede	5ms	30ms	100ms	300ms
TCPM	3966	2900	2326	782
SCTP	3366	2074	1570	882

Figura 5.9: Latência de transação, rede 10Mbps com latência de rede



no tamanho *default* do sistema operacional. Para obter performance ótima, esses *buffers* devem ser maiores que o produto latência de rede $\times$ largura de banda.

Os testes utilizaram sempre o tamanho padrão, pois é o que faz a maioria das aplicações na prática. E mesmo quando a aplicação permite a regulagem do *buffer*, dificilmente algum usuário atua sobre esse parâmetro.

Confome mostram a tabela 5.8 e a figura 5.8, a performance de vazão do SCTP foi congruente com os demais testes (menor que TCP), considerando-se o tamanho da mensagem de 500 *bytes*.

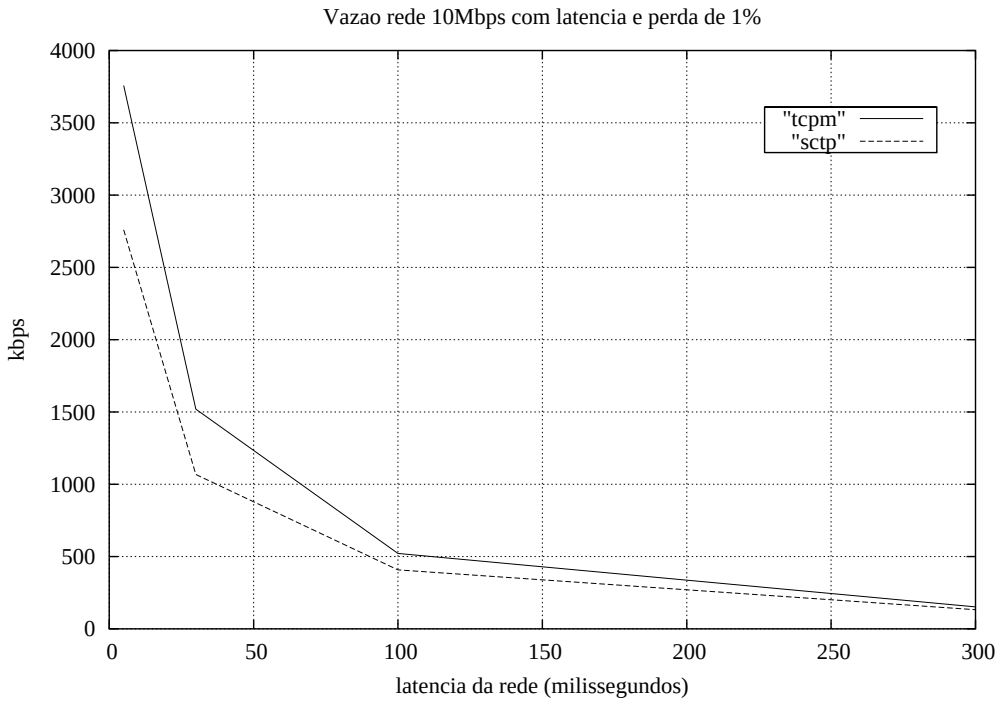
O desempenho do SCTP foi melhor que TCP quando o RTT de rede atingiu valores muito altos (600ms). Mas RTTs dessa ordem de grandeza são raros mesmo na Internet, sendo observados apenas em comunicações via satélite.

Conforme mostram a tabela 5.9 e a figura 5.9, a latência média da transação foi essencialmente o RTT da rede. A latência adicional do SCTP, demonstrada em outros

Tabela 5.9: Latência de transação em μs, rede 10Mbps com latência de rede

Protocolo / latência de rede	5ms	30ms	100ms	300ms
TCPM	10303,5	60286,0	200266,0	600183,5
SCTP	10355,0	60337,0	200298,5	600189,5

Figura 5.10: Vazão, rede 10Mbps com latência de rede e perda de pacotes



testes, aqui fica totalmente diluída.

5.7.5. 10Mbps com latência e perda de pacotes

Quanto à vazão, a figura 5.10 e a tabela 5.10 mostram valores congruentes com outros testes. O SCTP tende a estreitar a diferença com TCP na medida que a latência da rede alcança valores muito altos, devido ao RTO mínimo padrão ser de 1 segundo.

A latência, conforme o sucedido em outros testes, foi prejudicada na presença de perda constante. Mas a figura 5.11 e a tabela 5.11 mostram que, quanto maior a latência,

Tabela 5.10: Vazão em kbps, rede 10Mbps com latência de rede e perda de pacotes

Protocolo / latência de rede	5ms	30ms	100ms	300ms
TCPM	3758	1520	522	152
SCTP	2760	1068	408	132

Figura 5.11: Latência de transação, rede 10Mbps com latência de rede e perda de pacotes

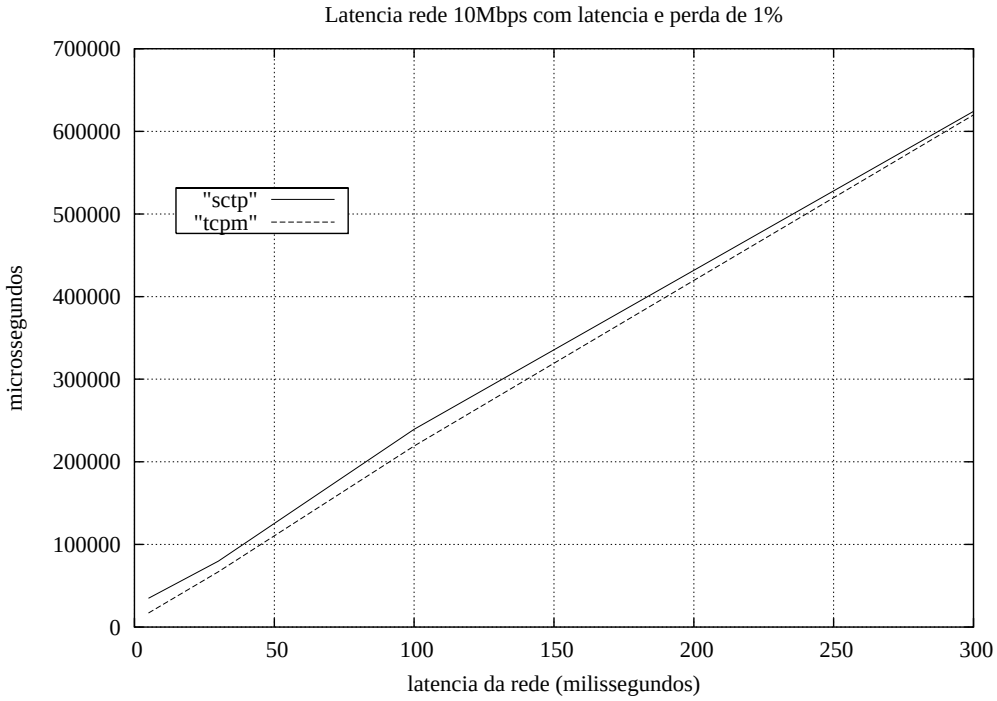


Tabela 5.11: Latência de transação em μs, rede 10Mbps com latência de rede e perda de pacotes

Protocolo / latência de rede	5ms	30ms	100ms	300ms
TCPM	17006,5	66986,0	219057,5	619908,5
SCTP	34844,5	79712,5	239280,5	624185,0

Tabela 5.12: Vazão e latência de transação, rede 1Mbps com de latência de rede e duplicação de pacotes

Protocolo	Vazão (kbps)	Latência (μs)
TCPM	710	100204,25
SCTP	652	108955,25

Tabela 5.13: Vazão e latência de transação, rede 1Mbps com latência de rede, perda e duplicação de pacotes

Protocolo	Vazão (kbps)	Latência (μs)
TCPM	740	105480,3
SCTP	682	109036,0

mais bem adaptado fica o SCTP. A razão é que o RTT foi se aproximando do RTO mínimo padrão (1 segundo).

5.7.6. Rede 1Mbps, latência e duplicação de pacotes

A vazão do SCTP, pouco menor que TCP, conforme a tabela 5.12, foi congruente com os demais testes. A latência do SCTP ficou sensivelmente mais alta que o RTT da rede, devido ao RTO mínimo padrão (1 segundo, 10 vezes maior que o RTT médio de 100ms).

5.7.7. Rede 1Mbps, latência, duplicação e perda de pacotes

A tabela 5.13, que contém os resultados deste teste, mostra uma aparente inconsistência: as vazões foram *maiores* na presença da perda constante, do que no teste de 1Mbps sem perda constante (tabela 5.12).

Tal diferença pode ser explicada pela influência desprezível da pequena perda constante sobre a vazão (pois a maior parte das perdas é variável e provocada pela simulação de congestionamento), e pela variância natural do teste.

A perda constante influenciou negativamente a latência do TCP. Interessante notar que a latência do SCTP ficou praticamente inalterada.

5.7.8. Rede 11Mbps *wireless*

O desempenho do SCTP, um pouco menor que o do TCP, foi congruente com os demais testes realizados, conforme mostra a tabela 5.14.

Tabela 5.14: Vazão e latência, rede 11Mbps *wireless*

Protocolo	Vazão (kbps)	Latência (μs)
TCPM	1221	6392,5
SCTP	916	7829,0

Tabela 5.15: Vazão e latência, rede 100Mbps – um servidor e dois clientes

Simultâneo	Cliente	Protocolo	Vazão (kbps)	Latência (μs)
Não	SOLDIER	TCPM	16584	421
		SCTP	15520	516
	COP	TCPM	(*) 9176	425
		SCTP	15448	523
Sim	SOLDIER	TCPM	4816	490
		SCTP	7736	681
	COP	TCPM	4792	490
		SCTP	7784	681

5.7.9. Dois clientes 100Mbps

O servidor é consideravelmente mais fraco que qualquer dos clientes, portanto os resultados serão diferentes de testes anteriores com 100Mbps. Para ter-se uma base de comparação, os testes foram refeitos entre cada cliente individual e o servidor.

Os resultados do teste estão na tabela 5.15. Para a vazão, o resultado ideal seria que a vazão total fosse a mesma para 1, 2 ou mais clientes. Foi o que aconteceu com SCTP. Já o TCP experimentou uma perda ao atender 2 clientes simultaneamente.

A razão não é exatamente um problema do TCP, e sim que os fluxos nas 2 direções ficaram descompassados (i.e. o tráfego num sentido acabava muito antes que o outro), baixando a média. Isso foi causado pela combinação servidor lento e cliente rápido. O SCTP também experimentou vazões baixas em outros testes pelo mesmo motivo. Mas por algum motivo o SCTP conseguiu, neste teste, manter o fluxo equilibrado em ambos os sentidos.

A tabela 5.15 ainda tem um ponto marcado com asterisco, assinalando um caso que o teste com TCP sofreu descompasso mesmo sem simultaneidade de clientes.

Nas latências de transação, as latências do TCP foram todas mais baixas, como de costume; e o TCP sofreu menos o efeito da simultaneidade (15%) que o SCTP (30%). A possível razão é o consumo de CPU inerentemente maior do SCTP (*checksum* CRC-32c) associado a um servidor fraco e ocupado.

Tabela 5.16: Vazão em kbps, conexões multicaminhos 10Mbps e 11Mbps

Protocolo	Rede 10Mbps	Rede 11Mbps	Ambas (multicaminhos)
TCPM	3746,0	1786,0	(*) 1064,0
SCTP	2586,0	1128,5	1750,0

Tabela 5.17: Latência em μs, conexões multicaminhos 10Mbps e 11Mbps

Protocolo	Rede 10Mbps	Rede 11Mbps	Ambas (multicaminhos)
TCPM	1773,5	5757,5	(*) 4319,0
SCTP	1886,0	6868,5	4505,5

5.7.10. Multicaminhos 10Mbps e 11Mbps

A associação SCTP é feita enquanto o *link* primário está ativo, pois a implementação LK–SCTP não possui a função *sctp_connectx()* que permitiria especificar diversos endereços de servidor. A descoberta dos endereços secundários do servidor é feita automaticamente na criação da associação (que só pode acontecer quando o *link* primário está ativo).

Em TCP no teste com "multicaminhos", ele simplesmente usa o *link* primário, e o trafego cessa nos períodos de bloqueio. Naturalmente, a performance será sofrível nessa situação. Seria difícil tornar esse teste mais justo para o TCP, já que TCP não tem nenhum recurso semelhante a multicaminhos.

Os principais objetivos do teste são determinar se o multicaminhos do LK–SCTP funciona, e se ele reage suficientemente rápido para funcionar sobre uma rede problemática.

O parâmetro ajustável *path_max_retrans* do SCTP foi calibrado para 1 (o valor padrão é 5), mais apropriado para a rede do teste, pois o tempo de retransmissão cresce exponencialmente. Numa rede "normal", onde as falhas sejam infreqüentes, o valor original pode ser adequado.

Os resultados dos testes de vazão estão listados na tabela 5.16. Como é naturalmente esperado, a vazão do SCTP com multicaminhos foi muito superior ao do TCP, ficando na média entre os links individuais. A tabela 5.16 também lista a vazão de cada caminho isolado, para verificar a congruência com os demais testes.

Quanto a latência. Apesar do uso de multicaminhos, a tabela 5.17 mostra que o desempenho do SCTP foi quase igual ao TCP. Verificou-se através de ferramentas de

diagnóstico de rede que o SCTP continuou usando o caminho secundário (cuja latência é muito maior) por 2 ou 3 segundos mesmo depois do caminho primário voltar a funcionar.

Para efeitos de comparação, o RTT da rede 10Mbps do teste era aproximadamente 0.36ms, e o RTT da rede 11Mbps era de aproximadamente 2.75ms. Os valores foram obtidos através do utilitário *ping*.

5.8. Validade dos testes

Os testes, na verdade, não avaliaram os protocolos TCP e SCTP em si, mas sim duas implementações particulares desses protocolos. Ainda assim, considera-se que os testes fornecem subsídios para a avaliação dos protocolos, pelas seguintes razões:

- demonstram que as promessas do protocolo SCTP podem ser entregues na prática;
- as duas implementações compartilham muitas características de implementação (estruturas de *kernel*, camada de rede etc.), de modo que as maiores diferenças observadas deverão ser *a priori* explicáveis por diferenças entre os protocolos, e não entre as implementações;
- a maturidade do TCP no Linux, pelo tempo de desenvolvimento e uso generalizado, fornece razoável certeza de que o desempenho do TCP é um patamar dificilmente superável por outros protocolos confirmados e/ou por outros sistemas operacionais, dado o mesmo *hardware*;
- a relativa imaturidade do LK–SCTP sugere que a performance obtida nos testes deste trabalho é atingível por qualquer outra implementação razoável (em particular se *kernel-level*) e o LK–SCTP tende a melhorar com o tempo (ou pelo menos, não piorará);
- os parâmetros ajustáveis do protocolo, como RTO mínimo, foram deixados intocados tanto quanto possível. Em alguns casos isolados foi necessário manipulá-los para determinar se um problema era causado pelo protocolo ou por uma falha da implementação. Tais ocorrências estão devidamente documentadas.

5.9. Conclusões

Os testes evidenciam que o desempenho do SCTP é próximo e inferior ao TCP, um resultado esperado pelas características inerentes a cada um.

Em algumas situações o SCTP teve desempenho melhor que TCP, mas foram situações-limite (RTT alto, percentagem específica de perda de pacotes) que não autorizam a dizer que o SCTP possa ter performance sustentada maior que TCP em uma rede real.

Uma fonte importante de sobrecarga do SCTP é a separação de mensagens. A maioria dos testes TCP utilizou um protocolo de aplicação, denominado TCPM, criado especialmente para os fins deste capítulo. Sua única função é impor a sobrecarga de separação de mensagens ao TCP, permitindo uma comparação mais justa de desempenho entre os dois protocolos de transporte.

Foram encontradas duas falhas algo graves do LK–SCTP: o RTO (*timeout* de pacote perdido) mínimo padrão configurado para 1 segundo (muito alto), e o algoritmo de controle de congestionamento que pode ter alguma instabilidade relacionada à perda constante de pacotes. Para tornar-se um protocolo de uso generalizado, o SCTP deverá absorver as melhorias recentes no controle de congestionamento que o TCP demonstrou possuir.

6. Comparação de desempenho com TCP e UDP em aplicações reais

6.1. Objetivos

Os testes deste capitulo objetivam verificar a performance do SCTP com protocolos de aplicação reais, usando implementações reais.

A escolha dos protocolos de aplicação levou em conta os seguintes critérios: relevância do protocolo na Internet, potencial de melhoria de performance no uso de SCTP como transporte, e existência de implementações de livre distribuição facilmente adaptáveis.

O protocolo HTTP é, de longe, a primeira escolha em todos os critérios. Praticamente todo texto introdutório sobre SCTP, como ARIAS-RODRIGUEZ, menciona o HTTP como candidato a adaptação.

O método de teste será a transmissão de objetos de tamanho fixo via HTTP, e o critério de avaliação do desempenho é a vazão em KBytes/s (que indiretamente indica o número de objetos transmitidos por unidade de tempo).

Dentre inúmeros protocolos cliente/servidor com mensagens de tamanho limitado, que tirariam proveito das mensagens indívisíveis do SCTP, o protocolo SMB (também conhecido pelos nomes NetBIOS e CIFS) foi escolhido por possuir implementações fora do *kernel*. A primeira escolha teria sido o NFS, mas tanto cliente quanto servidor NFS são tipicamente implementados dentro do *kernel*, o que torna a adaptação muito mais difícil de testar e depurar.

Assim como o HTTP, o SMB também usa originalmente o TCP como transporte. Dois métodos de teste serão empregados. O primeiro método mede a vazão mediante a transmissão de um único arquivo grande, assemelhando-se à experiência de um usuário copiando um arquivo ou pasta através da rede. O segundo método estima a latência, executando-se um grande número de operações de arquivo pequenas e serializadas, de forma semelhante a um banco de dados cujo arquivo está num servidor de rede.

Era desejável que este capítulo abordasse ao menos um protocolo de aplicação baseado em UDP. Para atender a esse objetivo, foi escolhido o protocolo RTP, que além de tudo proporciona uma excelente oportunidade de testar a extensão PR–SCTP, até aqui não abordada em nenhum teste. O método de teste do RTP é medir a capacidade de entregar com qualidade aceitável um conteúdo multimídia – áudio em formato MP3 – através de uma rede com banda estreita e perdas de pacotes.

O protocolo FTP foi cogitado como quarta opção, mas foi descartado pois sua adaptação ao SCTP exigiria extensões ao protocolo de aplicação para aproveitar os múltiplos fluxos, sem nenhuma perspectiva de uso prático. O FTP é um protocolo obsoleto; HTTP (em particular com a extensão WebDAV) e SFTP são alternativas melhores e prontamente disponíveis.

6.2. Adaptação dos protocolos de aplicação ao SCTP

O SCTP facilita a adaptação dos protocolos de aplicação originalmente concebidos para outros transportes. Em geral, nenhuma alteração no protocolo de aplicação em si é necessária. É o caso das adaptações do SMB e do RTP.

Mas, para aproveitar melhor as novas características do SCTP, uma adaptação mais profunda, com alguma modificação no protocolo de aplicação, se faz necessária. É o caso da adaptação do HTTP. Apesar de tudo, a adaptação proposta neste trabalho causa baixo impacto na definição do protocolo, e conseqüentemente baixo impacto nas implementações.

As adaptações aqui sugeridas podem ainda ser objeto de estudo mais aprofundado e/ou serem propostas ao IETF.

6.2.1. HTTP

A adaptação do HTTP ao SCTP procurou atingir os objetivos delineados em com o menor impacto possível sobre o protocolo definido na RFC 2616 (FIELDING, R. et al.).

Adaptação nível 1 – simples substituição de TCP por SCTP:

- Nenhuma alteração no protocolo trafegado, o que possibilita usar o mesmo interpretador HTTP sobre TCP e sobre SCTP;
- Associação SCTP com apenas um par de fluxos, um em cada direção, o suficiente para emular uma conexão TCP;
- Os dados têm de ser tratados como um fluxo contínuo, ignorando a atomicidade de mensagens proporcionada pelo SCTP;
- Clientes têm de solicitar associações com exatamente um par de fluxos, e servidores tëm de criar a associação também com exatamente um fluxo por direção. Isso sinaliza de parte a parte que a implementação é nível 1;
- O cliente HTTP não pode tentar colocar a associação em estado meio-fechado (através de *shutdown()* ou API equivalente) para sinalizar fim de requisição HTTP, pois o SCTP não suporta o estado meio-fechado.

Adaptação nível 2 – uso de múltiplos fluxos SCTP:

- Nenhuma alteração no protocolo trafegado, o que possibilita usar o mesmo interpretador HTTP sobre TCP e sobre SCTP;
- Os dados têm de ser tratados como um fluxo contínuo, ignorando a atomicidade de mensagens proporcionada pelo SCTP;
- O cliente HTTP não pode tentar colocar a associação em estado meio-fechado (através de *shutdown()* ou API equivalente) para sinalizar fim de requisição HTTP, pois o SCTP não suporta o estado meio-fechado;
- Os fluxos SCTP de mesmo número e direções opostas são agrupados em pares. Cada par é tratado como uma conexão bidirecional isolada das demais, tal qual TCP. A requisição HTTP transmitida pelo fluxo de número *n*, é respondida através do fluxo de número *n* de sentido oposto. Isso facilita a implementação e vem ao encontro da RFC 3436 (JUNGMAIER et al, 2002, define o protocolo de segurança TLS para SCTP) que agrupa os fluxos da mesma forma;
- O cliente HTTP deve requisitar tantos pares de fluxos quantos forem os arquivos que pretenda obter do servidor, até o limite do protocolo (65536) ou o limite imposto pela implementação SCTP;
- O servidor HTTP deve criar a associação com, no máximo, o número de fluxos requisitado pelo cliente. É facultado ao servidor criar a associação com menos fluxos que o requisitado, se por qualquer motivo não puder atender o pedido original do cliente (muito provavelmente implementações reais vão impor limites relativamente baixos, para minimizar o impacto de ataques de inundação);
- O cliente HTTP tem de adaptar-se graciosamente ao número de fluxos realmente aceito pelo servidor (geralmente abrindo outra(s) associação(ões) para requisitar os arquivos restantes);
- O servidor pode rejeitar como inválidas as associações onde o cliente tenha requisitado um número diferente de fluxos em cada direção. Se optar em aceitar a associação, deve considerar o menor dos dois números de fluxos como sendo o real número de conexões;
- O servidor HTTP não pode fechar a associação até que todos os fluxos tenham feito as últimas requisições (parâmetro `Connection:close`), e todas essas requisições estejam satisfeitas. O parâmetro `Connection:close` passa a indicar apenas que o par de fluxos não pode ser mais usado para nenhuma requisição;

- Todas as mensagens SCTP que compõem a resposta HTTP devem ter o código de protocolo igual a zero. Quando a transação HTTP for a última a acontecer em determinado fluxo (`Connection:close`), a última mensagem SCTP da resposta deve ter o código de protocolo diferente de zero;
- Mudança na seção 4.4 da RFC 2616, que define as estratégias de identificação de fim-de-arquivo bem como a prioridade de aplicação (mudança em negrito):
  1. Mensagem de resposta sem corpo termina sempre e linha vazia;
  2. Transferência em trechos (*chunked*) é auto-delimitada;
  3. Valor do parâmetro `Content-Length`;
  4. Mídia codificada (e.g. comprimida por gzip) é considerada auto-delimitada, se `Content-Length` não estiver presente;
  5. **Mensagem com código de protocolo diferente de zero, quando a transação é `Connection:Close`;**
  6. Fechamento da **associação** por parte do servidor indica fim de arquivo **para todos os fluxos**.
- O cliente HTTP deve dar preferência ao uso de múltiplos fluxos em lugar de *pipelining.*

Alocar um número arbitrariamente grande de fluxos, mas utilizar apenas alguns deles, contemplaria o caso de o cliente não saber exatamente, no momento da associação, quantos arquivos pretende requisitar. Essa possibilidade foi vetada pelos seguintes motivos:

- Consumiria muita memória de *kernel* por associação SCTP, em implementações que não otimizassem esse perfil de uso, como é o caso do LK–SCTP;
- O servidor HTTP típico aloca uma estrutura de controle de conexão no momento que a conexão TCP é criada. No caso do SCTP, todas as estruturas de conexão seriam criadas na criação da associação. Admitir que a maioria das conexões não seria usada implicaria em atrasar sua criação, complicando ainda mais o processo de adaptação;
- Na prática, todo servidor HTTP sobre SCTP nível 2 iria limitar o número de fluxos em um número bastante baixo, para dificultar ataques de inundação e diminuir o consumo de memória, tanto no *kernel* quanto no aplicativo. Aceitar 65536 fluxos significaria criar até 65536 estruturas de controle de conexão para uma única associação, facilitando o ataque de inundação.

6.2.2. SMB

O protocolo SMB também é conhecido pelas siglas CIFS e NetBIOS. É o protocolo de compartilhamento de arquivos e impressoras nativo da família de sistemas operacionais *Microsoft Windows*. A definição original do protocolo está nas RFCs 1001 (AGGARWAL, 1987) e 1002 (AGGARWAL, 1987a).

Desde sua criação em 1984, o SMB foi adotado e estendido por diversos fabricantes. Em geral, as extensões não foram formalizadas em RFCs ou livros. Alguns materiais como o de LEIGHTON tentam documentar as extensões conhecidas, mas o fato é que a versão do SMB em uso corrente não tem um documento público que descreva-o de forma completa e exata.

A implementação considerada "oficial", por ser a mais utilizada, é a do sistema operacional *Microsoft Windows*, com quem as demais implementações tentam manter-se interoperáveis lançando mão de engenharia reversa.

A adaptação do SMB consiste simplesmente na substituição do TCP por SCTP. O protocolo em si não sofre nenhuma modificação. Cada mensagem SMB será transmitida como uma mensagem indivisível SCTP, o que em tese desonera o receptor da separação de mensagens.

Os serviços SMB baseados em UDP, como a resolução de nomes, permanecem como estão, pois fazem uso de *broadcast*.

6.2.3. RTP

O protocolo RTP é definido pela RFC 1889 (SCHULZRINNE et al.). A adaptação para SCTP não exige nenhuma mudança no RTP em si. A implementação deve gerar mensagens indivisíveis SCTP da mesma forma que geraria pacotes UDP.

Se o RTP está sendo usado para transmissão em tempo real, as mensagens SCTP devem ter um prazo de validade finito – provavelmente baseado nos RTT e RTO da associação, cujos valores são consultáveis pelo aplicativo. O aplicativo também pode processar as notificações sobre mensagens não transmitidas, e com base nisto calibrar o prazo de validade. (Usando UDP, a única forma de obter *feedback* sobre perdas é através do protocolo auxiliar RTCP.)

A presença ou ausência da extensão PR–SCTP não altera em nada as aplicações (nem mesmo exige recompilação), mas melhora a performance em caso de congestionamento severo da rede.

6.3. Aplicativos adaptados para os testes

A metodologia de alteração dos programas foi, resumidamente, a seguinte:

- Busca das chamadas *socket()* e mudança para SCTP;
- Verificação das chamadas *setsockopt()*, verificando quais devem ser mudadas para adequação ao SCTP, e quais simplesmente não cabem no SCTP;
- Busca de chamadas *shutdown()* que criem o estado meio-fechado, eliminação da chamada (se isto não afeta o protocolo de aplicação e há uma chamada *close()* mais adiante que feche a associação);
- Verificação do tamanho da mensagem e dos *buffers* de transmissão/recepção (os *buffers* têm de ser maiores que a maior mensagem a ser trafegada);
- Decisão de uso do estilo TCP ou UDP, de acordo com o mais adequado ao estilo de programação do aplicativo. Na grande maioria dos casos o soquete TCP será mais adequado, pois a maioria dos programas adaptáveis utilizava TCP.

Se a adaptação pretende usar vários fluxos em lugar de várias conexões, alterações mais profundas são necessárias. A principal é identificar de que fluxo veio determinada mensagem, pois o manipulador de arquivo deixa de ser um identificador unívoco para a conexão. Tipicamente será inserido código entre a leitura e o despacho da mensagem, de forma a entregá-la à conexão correspondente.

Também é necessário verificar quando uma conexão coloca o manipulador de arquivo em estado somente-leitura ou somente-gravação; ao fazer isso sobre uma associação SCTP, estará afetando outras conexões. Todas estas chamadas deverão fazer testes adicionais (e.g. para colocar o soquete em estado somente-leitura, todas as conexões relacionadas devem estar nesse estado).

6.3.1. HTTP

Do lado servidor, a implementação escolhida foi o *thttpd* versão 2.25b. Este servidor posiciona-se como uma alternativa mais simples de configurar e mais leve que o Apache.

Sua principal característica é usar apenas um processo que atende a todas as requisições de páginas estáticas, proporcionando grande performance. Já páginas dinâmicas provocam forçosamente a criação de processos-filhos, no estilo CGI clássico; bibliotecas carregáveis dinamicamente não são suportadas. Portanto, o *thttpd* não é adequado para servidores de conteúdo majoritariamente dinâmico.

Outra característica do *thttpd* é ser portável apenas para sistemas POSIX, enquanto o Apache também suporta outras APIs como *Windows* 32 *bits* e *Novell Netware*. O *thttpd* também não implementa *pipelining*.

O *thttpd* sofreu as adaptações nível 1 e 2 para SCTP. A adaptação nível 2 foi consideravelmente mais trabalhosa e exigiu mais depuração. Cabe ressaltar que a arquitetura monoprocesso/monolinha facilitou a tarefa.

O suporte a CGI foi desligado na adaptação nível 2, pois o *thttpd* delega completamente o tratamento da conexão a um subprocesso, o que em SCTP não seria adequado, pois a cada soquete estão atreladas diversas conexões. Suportar CGI na adaptação nível 2 exigiria mudar mais profundamente o funcionamento do servidor, sem qualquer benefício para este trabalho.

Os clientes HTTP adaptados foram o Mozilla versão 1.6 e o *httperf* versão 0.8.

O Mozilla sofreu uma adaptação nível 1. A adaptação nível 2 exigiria um estudo profundo de seu funcionamento interno, o que fugiria ao escopo deste trabalho (o código-fonte do Mozilla tem 300 MB), e de qualquer forma não serviria para testar objetivamente o servidor.

O *httperf* é um *software* especificamente criado para testar a performance de servidores HTTP. Ele cria conexões (ou associações) na medida em que o servidor atende as anteriores, até chegar à saturação do servidor. Este *software* sofreu adaptação níveis 1 e 2, de modo que o *thttpd* sobre SCTP pudesse ser testado de forma objetiva. Assim como o *thttpd*, a adaptação nível 2 foi consideravelmente mais trabalhosa. Felizmente, o *httperf* também não é *multithreaded*.

6.3.2. SMB

O *software* escolhido para a adaptação foi o *Samba* versão 3.0.4, uma implementação de servidor SMB/CIFS para UNIX. O *Samba* também fornece utilitários-clientes para SMB, como o *smbclient* que será utilizado nos testes.

A adaptação do *Samba* foi a mais simples dentre todos os *softwares*, por diversas razões. O protocolo de aplicação não precisou de qualquer adaptação. O código-fonte do *Samba* é bem organizado e a criação de soquetes está concentrada em algumas poucas funções de biblioteca. Os soquetes SCTP estilo TCP funcionaram perfeitamente no lugar dos soquetes TCP originais. O *Samba* não utiliza chamadas como `shutdown()` ou recursos como *byte* urgente que implicariam em mudanças mais trabalhosas.

O separador de mensagens SMB poderia ter sido simplificado, pois em SCTP toda mensagem SMB será atômica. Esta alteração não foi feita por ser custosa, diminuir a

robustez (algum cliente problemático poderia mandar uma mensagem fragmentada), e não aplicar-se num *Samba* adaptado para TCP e SCTP simultaneamente (o separador continuaria atuando ao menos para os soquetes TCP).

Ainda assim, as mensagens indivisíveis SCTP poderiam trazer um ganho de perfomance ao SMB, pois o processo será acordado apenas quando for receber e processar uma mensagem SMB completa.

Foi necessário configurar os *buffers* de transmissão e recepção para valores acima de 64KBytes, pois este é o tamanho máximo de uma mensagem SMB, e o SCTP retorna erro para mensagens maiores que o *buffer* de transmissão. Isso pôde ser resolvido no arquivo de configuração, sem demandar mais mudanças no *software* em si. Uma adaptação mais robusta teria de verificar se os *buffers* têm o tamanho mínimo exigido.

6.3.3. RTP

O *software* `poc` versão 0.3.7 implementa clientes e servidores RTP para transferência de áudio no formato MP3. (Este *software* não implementa RTCP).

Apesar de suportar *bitrate* fixo e dividir a informação em quadros de tamanho fixo, o MP3 não foi projetado para transmissão com perdas. Um quadro pode referenciar dados de quadros anteriores; os quadros com sons mais “simples” emprestam o espaço livre para quadros futuros com sons mais “complexos”.

Esse mecanismo, conhecido como *reservoir*, permite oferecer uma qualidade sonora razoável com um *bitrate* baixo e constante. Mas os quadros deixam de ser *stateless*, o que desqualifica o MP3 para aplicações de tempo real como telefonia sobre IP. Também amplifica o efeito da perda de um pacote sobre a qualidade de áudio.

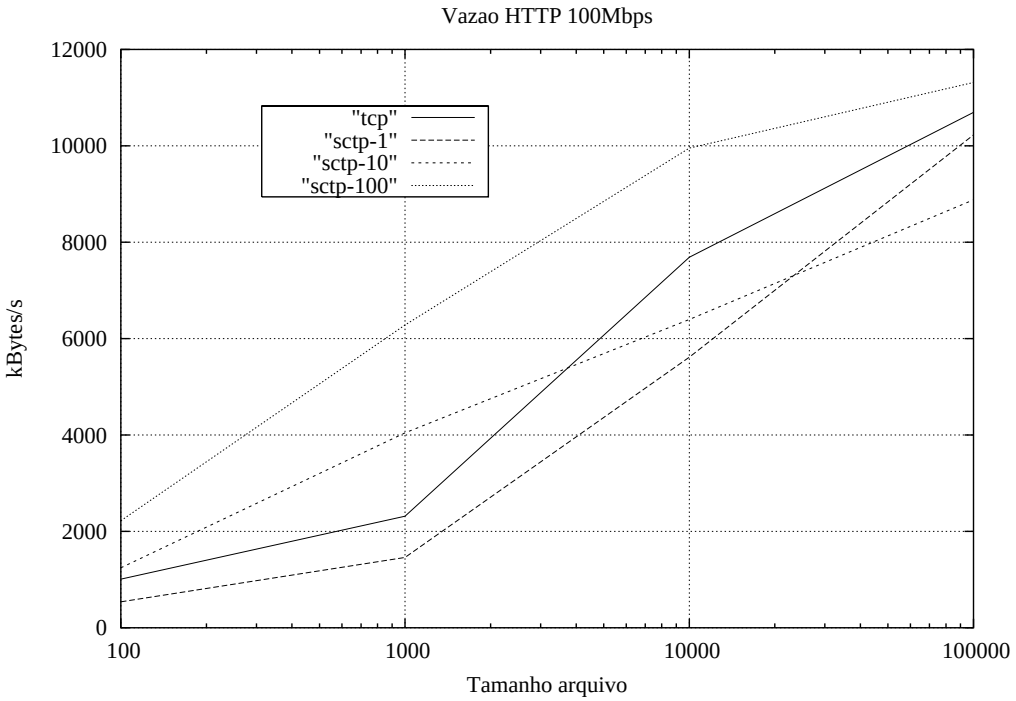
Para amenizar esse problema, o `poc` implementa três alternativas de mitigação de perdas para MP3: RFC 2250 (HOFFMAN et al.), RFC 3119 (FINLAYSON), e *Forward Error Correction* – FEC.

Este o único *software* dentre os adaptados que usa o soquete estilo UDP para transmissão de dados, pois neste o estilo UDP diminuiu o impacto da adaptação.

6.3.4. Outros aplicativos

Os *softwares* `trafshow` e `telnet` foram também adaptados para SCTP, como ferramentas de depuração de rede.

Figura 6.1: Vazão HTTP, rede de 100Mbps



6.4. Resultados

6.4.1. HTTP

Dois cenários foram criados para os testes HTTP:

- rede 100Mbps Ethernet;
- rede 1Mbps simulada (perde pacotes na saturação), sem perdas constantes, latência de 30ms, duplicação de 0,1%.

Os testes utilizaram as adaptações nível 2 para SCTP, que são capazes de utilizar vários fluxos por associação. A performance das adaptações nível 1 seria equivalente às adaptações nivel 2, desde que utilizando apenas um fluxo por associação.

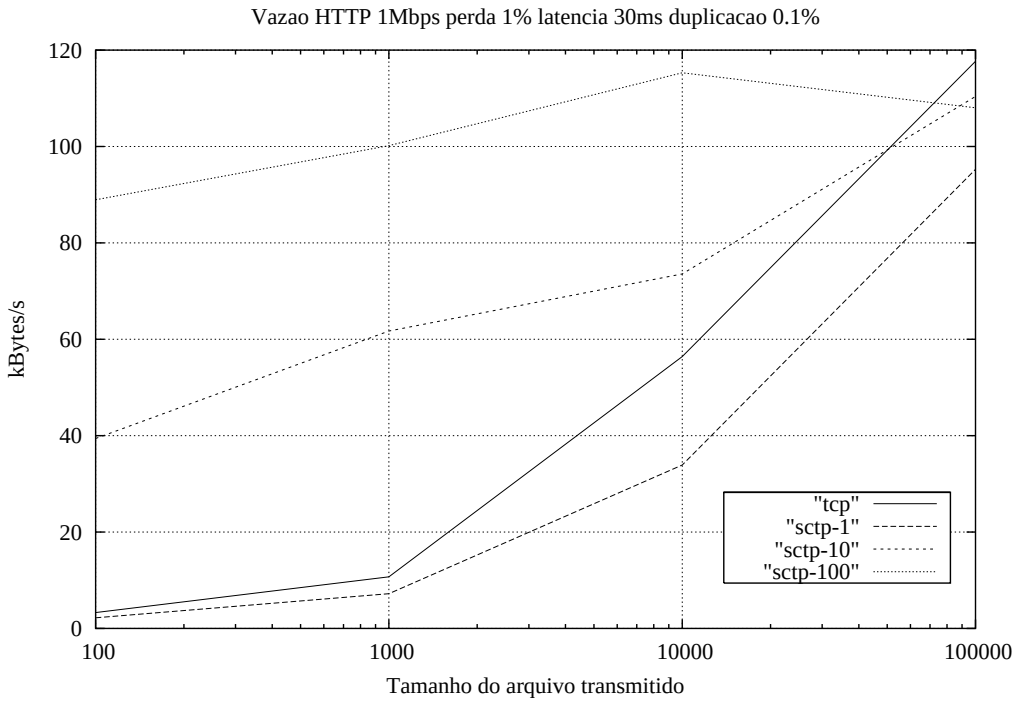
Nesses testes, a quantidade de arquivos tranferidos pelo *httperf* foi calibrada de forma a não saturar o servidor, pois o objetivo era medir a vazão e não o número máximo de conexões possíveis (que dependem unicamente de configurações e/ou variáveis de compilação do *thttpd*).

No cenário de rede 100Mbps, os resultados podem ser vistos na figura 6.1 e tabela 6.1. Para apenas um fluxo, a performance do SCTP é menor que TCP, como em todos os testes de vazão bruta que já foram feitos.

Tabela 6.1: Vazão HTTP em Kbytes/s, rede de 100Mbps

Tamanho arquivo (*bytes*)	TCP	SCTP - 1 fluxo	SCTP - 10 fluxos	SCTP - 100 fluxos
100	1008,7	537,6	1243,9	2221,3
1000	2315,0	1458,6	4043,1	6281,7
10000	7688,2	5613,4	6397,2	9954,1
100000	10693,3	10226,1	8880,5	11314,1

Figura 6.2: Vazão HTTP, rede de 1Mbps com perdas e latência



Ainda, na medida em que os arquivos aumentam de tamanho, a vantagem do SCTP diminui até desaparecer, pois a vazão bruta da rede torna-se o limitador principal, e nesse caso o SCTP leva sempre desvantagem com sua maior sobrecarga.

Um fator adicional detectado nesse teste, que prejudica os números finais do SCTP, é que a associação SCTP leva mais tempo que TCP para fechar a associação, tempo este que o *httperf* inclui no cômputo da vazão.

Já com o uso de vários fluxos por associação, a performance do SCTP foi melhor que TCP, em particular para arquivos pequenos. Este é um resultado esperado, pois a sobrecarga da criação e destruição de conexões é grandemente diluída, e o cliente pode requistar muito mais arquivos sem saturar o servidor (e a si mesmo).

Os resultados do cenário de rede 1Mbps, verificáveis na figura 6.2 e na tabela 6.2, são congruentes com aqueles obtidos nos testes com a rede 100Mbps. A diferença a favor

Tabela 6.2: Vazão HTTP, rede de 1Mbps com perdas e latência

Tamanho arquivo (*bytes*)	TCP	SCTP - 1 fluxo	SCTP - 10 fluxos	SCTP - 100 fluxos
100	3,3	2,2	39,4	88,9
1000	10,7	7,2	61,7	100,2
10000	56,4	33,9	73,6	115,3
100000	117,7	95,3	110,4	108,0

do SCTP aumenta ainda mais pois a perda de um pacote atrasa a entrega de apenas um fluxo, e não de todos.

Além deste trabalho, outros já testaram o HTTP sobre SCTP. Em particular o artigo de RAJAMANI et al. descreve um teste com HTTP sobre SCTP, e chegou a resultados análogos aos deste trabalho, embora tenha usado o sistema operacional BSD com uma implementação KAME relativamente imatura (o artigo é datado de 2002, e o KAME SCTP tinha então apenas sete meses de existência). O servidor HTTP não é especificado no artigo.

O método de teste foi essencialmente o mesmo – transmitir um certo número de arquivos de tamanho fixo. Mas a métrica utilizada para quantificar o desempenho naquele artigo foi a latência, ou seja, o tempo médio de entrega de um arquivo, ao invés da vazão.

6.4.2. SMB

Dois cenários básicos foram criados para os testes HTTP e SMB:

- rede 100Mbps Ethernet;
- rede 1Mbps simulada (perde pacotes na saturação), sem perdas constantes, latência de 30ms, duplicação de 0,1%.

Em 100Mbps, o teste de vazão consistiu da transmissão de um arquivo de 100 milhões de *bytes*. Já em 1Mbps, foi transmitido um arquivo de 1 milhão de *bytes*. Foram utilizados arquivos diferentes para que o teste não se alongasse demasiadamente em 1Mbps.

O teste de latência consistiu da execução repetida de dois comandos: transmissão de um arquivo de 100 *bytes*, e imediata remoção desse arquivo. O objetivo do teste é medir a agilidade no processamento de várias transações pequenas, comuns aplicações. Para 100Mbps, o par de comandos foi repetido 5000 vezes. Em 1Mbps, o par foi repetido 100 vezes.

Tabela 6.3: Vazão e latência SMB (valores em segundos, vide seção 6.4.2)

Rede / teste	TCP - vazão	SCTP - vazão	TCP - latência	SCTP - latência
100Mbps	11,54	12,39	6,12	7,13
1Mbps	8,53	11,51	30,02	30,09

Todos os valores dos testes, listados na tabela 6.3, são em segundos corridos (*wall clock*). Valores menores equivalem a melhor desempenho. O SCTP foi superado pelo TCP em todos os testes.

A simples substituição de TCP por SCTP não trouxe ganho de performance. O principal motivo parece ser o fato da implementação emitir duas mensagens para cada requisição, o que aumenta a desvantagem do SCTP pois as duas mensagens são entregues rigorosamente separadas no destino.

Seria necessária uma adaptação mais profunda, que explore melhor a separação de mensagens do SCTP, emitindo apenas uma mensagem por requisição e desligando a separação de mensagens na camada de aplicação, que o SCTP supre.

6.4.3. RTP

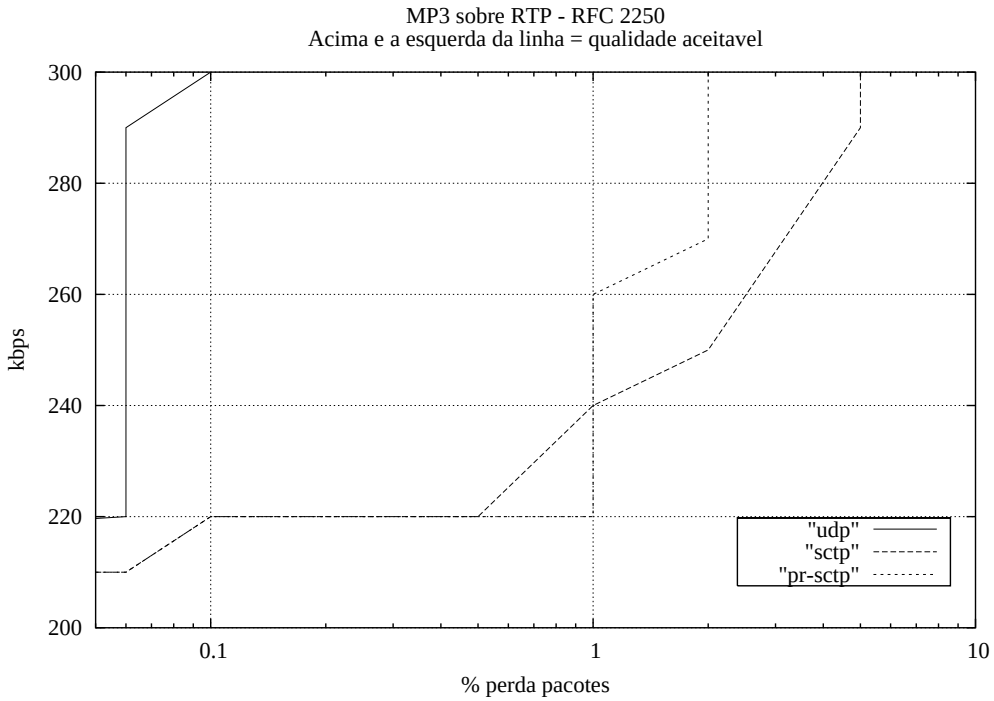
O teste de performance do RTP não mede a latência ou vazão, mas sim a capacidade de entregar satisfatoriamente um conteúdo multimídia através de uma rede com perdas. O conteúdo consiste de áudio MP3 com taxa contínua (CBR) de 192kbps. O critério de “entrega satisfatória” é subjetivo, pois é avaliado com base na saída de áudio por um ouvinte humano (o próprio autor).

Foi utilizada uma grade de cenários, com larguras de banda entre 200kbps e 300kbps em passos de 10kbps, e taxas de perda de pacotes de 0%, 0,1%, 0,2%, 0,5%, 1%, 2%, 5% e 10%. A latência de rede simulada é fixa em 30ms (RTT=60ms).

O *prebuffering* no cliente foi de 200ms, exceto no protocolo FEC onde o *prebuffering* é fixo e não pode ser dimensionado em termos de tempo. O consumo de banda do conteúdo foi em torno de 205kbps para RFC 2250 e RFC 3119, e 260kbps para FEC.

As mensagens SCTP foram remetidas sem garantia de reordenamento e com prazo de validade padrão de 200ms. Houve situações que exigiram a mudança desse prazo de validade. No SCTP padrão, a mensagem vencida é descartada somente se sua transmissão ainda não foi tentada. Com a extensão PR–SCTP, a mensagem vencida é descartada mesmo se já transmitida; o receptor é instruído a avançar a janela TSN sem esperar pelas mensagens perdidas.

Figura 6.3: Desempenho RFC 2250 sobre UDP, SCTP e PR–SCTP



Como o *buffer* do receptor é pequeno, a correta estimativa do RTO (*timeout* de pacote) torna-se muito importante para o SCTP reagir rapidamente aos erros. O RTO mínimo padrão do SCTP (*rto_min*), 1 segundo, seria totalmente inadequado para transporte de conteúdo multimídia em tempo real, de modo que este valor foi reduzido para 1 milissegundo. O RTO inicial (*rto_initial*) foi setado em 65ms, pouco mais que os 60ms do RTT da rede simulada. Assim, o SCTP adapta-se rapidamente à rede sem ser mais agressivo do que seria o TCP na mesma situação.

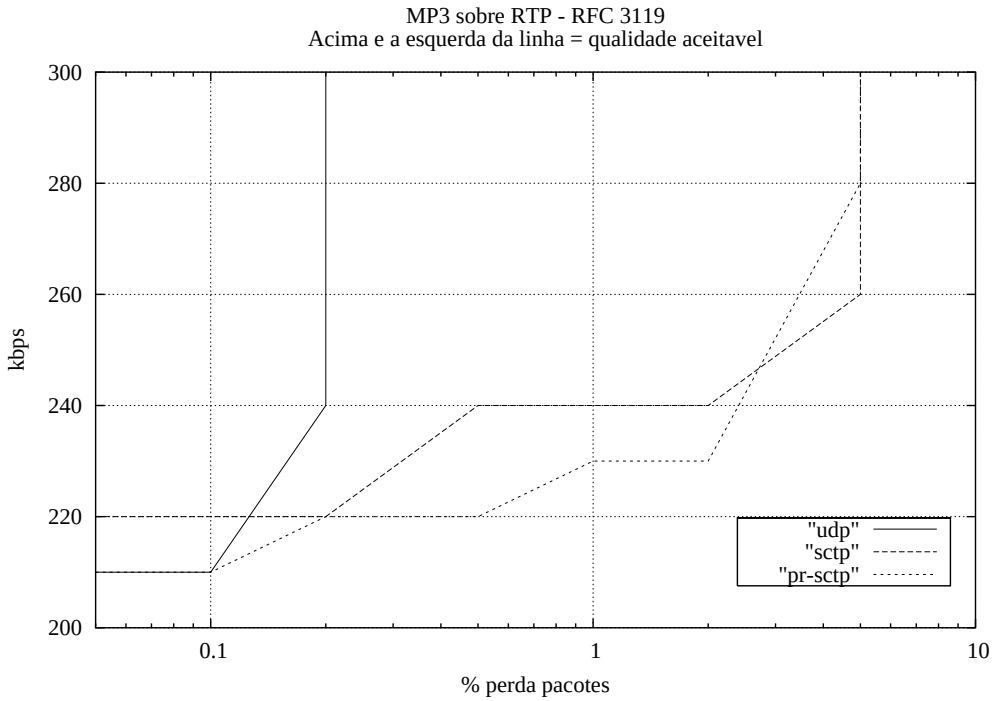
Os gráficos 6.3, 6.4 e 6.5 indicam a linha divisória entre cenários bem-sucedidos e mal-sucedidos. Os pontos mais acima e à esquerda das linhas representam cenários mais favoráveis (mais banda, menos perda), onde o conteúdo multimídia passou bem. Os pontos abaixo e à direita das linhas representam cenários piores, onde o conteúdo não conseguiu passar.

RFC 2250

O protocolo RFC 2250 é muito sensível a qualquer perda de pacotes, e foi incluído neste teste apenas por completeza, para apreciar o desempenho do SCTP. O desempenho de cada transporte pode ser verificado na figura 6.3.

Em UDP, qualquer falha faz o conteúdo apresentar ruído periódico da falha em diante, e dificilmente há recuperação. Com o SCTP, que adiciona confiabilidade ao tráfego,

Figura 6.4: Desempenho RFC 3119 sobre UDP, SCTP e PR–SCTP



o desempenho sob variadas perdas melhorou muito, viabilizando o RFC 2250. Além disso, as falhas conseguem recuperar-se – os ruídos periódicos tendem a desaparecer pouco depois da falha.

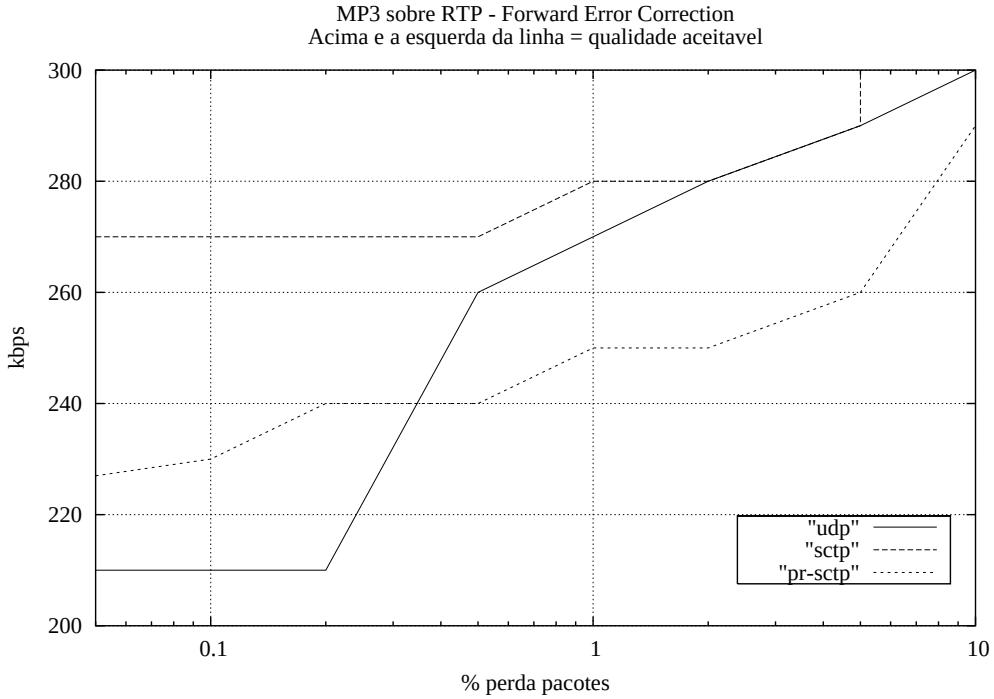
O SCTP normal apresentou desempenho mais linear, ou seja, aumentou a resistência a falhas conforme havia largura de banda de sobra, enquanto o PR–SCTP foi mais sensível à perda que à largura de banda, e não funcionou com perdas superiores a 2%.

RFC 3119

O protocolo RFC 3119 torna o conteúdo mais resistente a perdas, e evita o aparecimento de falhas periódicas na saída de som por conta de falhas isoladas. Seu comportamento foi muito bom tanto em UDP como em SCTP. Naturalmente, a resistência a perdas foi muito maior com SCTP.

Conforme mostra a figura 6.4, o PR–SCTP foi mais eficiente em situações de largura de banda estreita e perdas moderadas, porém foi mais sensível que o SCTP normal a situações de perda severa (5% e acima), mesmo com grande sobra de largura de banda. Essa sensibilidade poderia ser melhorada com a redução do prazo de validade das mensagens para um valor mais próximo do RTT.

Figura 6.5: Desempenho do protocolo FEC sobre UDP, SCTP e PR–SCTP



FEC – *Forward Error Correction*

A figura 6.5 evidencia os resultados. Como o protocolo FEC gera dados redundantes, a largura de banda mínima para tráfego é 260kbps. Desta forma, o SCTP padrão não funciona com largura de banda abaixo de 270kbps, pois é um protocolo confirmado.

O UDP conseguiu funcionar com taxas tão baixas quanto 210kbps, pois o FEC encarrega-se de reparar a maioria dos erros no receptor, e sem refletir os erros na saída de áudio.

O PR-SCTP teve, por sua vez, um desempenho bastante interessante, cobrindo praticamente as mesmas situações dos outros dois protocolos, e ainda apresentando desempenho superior que os demais em cenários de banda escassa e perda moderada.

O descarte de mensagens vencidas do PR–SCTP conseguiu manter o fluxo mesmo quando a banda da rede era inferior à taxa de dados gerada pelo transmissor. Assim como no UDP, as perdas são reconstruídas no receptor graças ao FEC.

Uma situação que exigiu ajuste do prazo de validade foi o SCTP e PR–SCTP com perdas de 5% e superiores. O tempo de validade teve de ser calibrado para abaixo de 100ms, do contrário o fluxo de mensagens estolava completamente.

Numa aplicação real, o prazo de validade teria de ser adaptativamente calibrado pelo transmissor, em função do RTT e das notificações de mensagens não transmitidas

por vencimento. Isto não é um defeito do SCTP frente ao UDP, e sim uma conseqüência de ele ser confirmado e possuir controle de congestionamento.

O receptor FEC ainda tem de sofrer ajustes para funcionar melhor com soquetes SCTP, pois ocorreram algumas falhas de audio não relacionadas a qualquer perda de dados (o próprio código-fonte alerta que soquetes não bloqueantes deverão ser empregados numa próxima versão).

7. Conclusões e trabalhos futuros

O objetivo do trabalho foi demonstrar a viabilidade do uso do protocolo de transporte SCTP em lugar de outros protocolos de transporte (principalmente TCP). Procurou-se fazer essa demonstração de forma gradativa, dos aspectos históricos aos testes reais.

Inicialmente foi demonstrada a necessidade da criação de um novo protocolo de transporte para a pilha TCP/IP, em função da intenção de usá-la em sinalização telefônica. Para esta demonstração, foi necessário introduzir conceitos de outra pilha de rede, a SS7, utilizada em telefonia. Conforme mostrado, nenhum protocolo preexistente na pilha TCP/IP era adequado ao transporte de mensagens de sinalização telefônica. O SCTP nasceu para esse fim, inicialmente como um protocolo de aplicação denominado MDTP, e posteriormente realocado para a camada de transporte recebendo o nome atual.

A seguir foram mostrados os novos recursos do SCTP em relação a TCP e UDP, com destaque para a atomicidade de mensagens, vários fluxos por associação, multicaminhos, extensibilidade e confiabilidade parcial.

Até este ponto, o trabalho versou sobre assuntos já abordados por outros documentos citados na bibliografia, com as devidas atualizações pois o protocolo SCTP ainda está em evolução. Concluiu-se, como nos demais trabalhos, que o SCTP fornece diversos recursos novos e interessantes para os protocolos de aplicação, e que o SCTP é uma novidade bem-vinda também para as redes TCP/IP de uso geral, como a Internet.

A seguir, estudou-se que tipos de protocolos de aplicação poderiam ter vantagens reais se portados para o SCTP. Dentre os protocolos TCP, aqueles que fazem uso de mensagens atômicas de tamanho máximo bem definido podem beneficiar-se do SCTP. Protocolos com mensagens arbitrariamente grandes, ou sem qualquer separação de mensagens, não aproveitam a atomicidade de mensagens do novo protocolo.

Outros protocolos baseados em TCP que, independentemente do tipo de mensagem, poderiam aproveitar o SCTP, são aqueles onde diversas conexões de transporte são utilizadas para realizar uma única transação lógica; tais conexões podem ser agrupadas numa única associação. O exemplo clássico é a busca de uma página HTTP completa com corpo e imagens.

Dentre os protocolos de aplicação baseados em UDP, cujo uso é cada vez mais reduzido, também são reduzidas as possibilidades de aproveitamento do SCTP. O UDP delega o controle das conexões lógicas à aplicação, possibilitando suporte a milhões de

clientes simultâneos (como nos servidores DNS globais) sem grandes problemas e com pouco *overhead*. O SCTP também não oferece suporte a *multicast* e *broadcast*.

Um dos poucos usos prováveis para SCTP em lugar de UDP é no transporte de conteúdo multimídia, graças aos recursos de confiabilidade parcial do SCTP (mensagens desordenadas e com prazo de validade). O SCTP também permite a presença simultânea de mensagens totalmente e parcialmente confiáveis na mesma associação, o que permite conteúdo e controle fluírem por um único canal (e.g. RTP e RTCP).

Três protocolos de aplicação, bem como implementações reais dos mesmos, foram adaptados para SCTP: HTTP, SMB e RTP. A adaptação proposta ao HTTP foi a mais elaborada pois previu o uso de diversos fluxos numa mesma associação, ou seja, diversas conexões HTTP fluindo por uma única associação SCTP. Nenhuma das três adaptações implicou em mudanças no protocolo de aplicação em si.

Foram realizados testes com a implementação LK–SCTP *(Linux Kernel SCTP)*, a mais avançada dentre as implementações SCTP de código-fonte aberto, suportando muitas extensões como PR–SCTP e AddIP. Foram medidas vazão e latência em variados cenários de rede. Os resultados foram em geral os esperados: o SCTP tem performance comparável, embora sempre um pouco menor, que TCP, devido ao maior *overhead* nos dados e no consumo de CPU.

Duas deficiências sérias na implementação do algoritmo de congestionamento do SCTP foram detectadas; os resultados serão devidamente repassados aos mantenedores do LK–SCTP.

Os programas utilizados nos testes de latência e vazão foram construídos especialmente para a tarefa, e não executam praticamente nenhum processamento, para evidenciar apenas as características dos protocolos de transporte. Uma medida de "justiça" entre SCTP e TCP foi impor a separação de mensagens na camada de aplicação para TCP, pois o SCTP sempre entrega mensagens atomicamente. Mas pelo menos dois grupos de testes, em rede *loopback* e Fast Ethernet, também foram feitos com TCP sem separação de mensagens, para que todas as possibilidades fossem contempladas.

Finalmente, as implementações dos protocolos HTTP, SMB e RTP, adaptadas para SCTP, foram postas à prova, com os bons resultados, na verdade esperados, para HTTP e RTP. A adaptação do SMB, que era simples mas esperava-se entregaria performance melhor que TCP, apresentou resultados piores que TCP. Essa adaptação teria de ser aprofundada para extrair-se a vantagem teoricamente prevista.

De tudo isso, conclui-se que o SCTP é um protocolo que merece a atenção dos desenvolvedores de aplicativos e de protocolos de aplicação, pois possui recursos atraentes, e ao menos uma implementação de qualidade muito próxima à de produção.

Como itens para trabalhos futuros que abordem o SCTP, sugere-se:

- Extensão para desligamento do *checksum* CRC-32c, que é redundante em conexões de rede local como Ethernet que já implementa CRC-32 na camada de enlace. Tal desligamento também serviria para mensurar o impacto real do CRC-32c sobre a latência, bem mais alta que a do TCP conforme mostram os testes deste trabalho. O desligamento do *checksum* deve ser acompanhado de mudanças no funcionamento interno do *driver* de *kernel*, visando diminuir o número de cópias dos dados, pois o simples desligamento feito num teste preliminar mostrou que o ganho de desempenho é relativamente pequeno;
- Alterações no algoritmo de controle de congestionamento, para melhor adaptação à perda constante de pacotes;
- Verificação da segurança do SCTP em diversas implementações, particularmente a geração de ISN e etiquetas de verificação para novas associações, que deve ser suficientemente imprevisível para evitar *blind spoof*;
- Adaptação ou elaboração de protocolos de aplicação que possam aproveitar de múltiplos fluxos SCTP para aumento da vazão, nos moldes do relatório de KANG & FIELDS;
- Adaptação de protocolos de aplicação de uso generalizado ao SCTP, e a proposição das adaptações como RFCs;
- Análise aprofundada do transporte de conteúdo multimídia em tempo real sobre SCTP, em particular a adaptação da taxa e do prazo de validade das mensagens, de acordo com o RTT e as perdas entre cliente e servidor.

Todos os *softwares* utilizados, tanto os construídos para o teste quanto os *softwares* de terceiros, estão disponíveis no CD encartado. O mesmo para o código-fonte deste documento e seus gráficos associados, várias RFCs, e planilha eletrônica com os resultados detalhados dos testes.

ANEXO 1: Características de segurança do SCTP

Os protocolos de transporte da pilha TCP/IP não foram inicialmente projetados com o objetivo de proporcionar segurança. O protocolo TCP tem alguma segurança, mais incidental que propositada, contra ataques "cegos" *(spoof)*, mas é sensível a ataques de negação de serviço *(DoS)*. As fraquezas do TCP são graves pois é o protocolo de transporte mais utilizado, e a maioria das aplicações não apresenta qualquer mecanismo adicional de segurança.

As duas principais fraquezas do TCP são:

- não autenticar as partes durante a criação da conexão;
- alocar recursos do servidor antes de a conexão estar completamente aberta.

A única proteção do TCP contra ataques *blind spoof* é confiar que cada terminal gere números iniciais de seqüência (ISN – *initial sequence number*) imprevisíveis. Essa imprevisibilidade foi prevista pelos criadores do TCP, porém como forma de distinguir mais facilmente encarnações diferentes da "mesma" conexão (i.e. conexão envolvendo mesmos endereços e portas). A implicação de segurança do ISN imprevisível foi percebida mais tarde.

Assim como uma conexão aberta, a conexão meio-aberta TCP ocupa uma estrutura TCB (*Transmission Control Block*) , pois lado passivo da conexão (tipicamente o servidor) precisa armazenar o ISN remetido pelo cliente no pacote SYN.

Como em geral o TCP é implementado no *kernel* do sistema operacional, e a memória do *kernel* é limitada, o número máximo de TCBs, e portanto o número de conexões TCP simultâneas, tem um limite bem definido.

Assim, um invasor pode simplesmente disparar um grande número de pacotes TCP SYN com origem falseada – é o ataque *SYN flood*. As conexões meio-abertas nunca são efetivadas e acabam descartadas por *timeout*, porém nesse meio tempo elas ocupam todos os TCBs disponíveis e impedem a realização de conexões legítimas (BERNSTEIN).

Felizmente, há uma técnica que praticamente elimina essa fraqueza do TCP: o *SYN cookie*. Consiste de um ISN gerado pelo servidor por um algoritmo criptográfico descrito por BERNSTEIN.

Depois de calcular o ISN e remeter o segundo pacote de *handshake*, o servidor não aloca TCB e não armazena nada sobre a conexão meio-aberta. Pode-se afirmar que o *SYN Cookie* transfere a responsabilidade do armazenamento da conexão meio-aberta para o cliente e para a rede. Se o cliente for legítimo e remeter o terceiro pacote de *handshake*, o servidor reconhece o seu ISN matematicamente, e só então atribui um TCB à conexão totalmente aberta.

Os *SYN cookies* funcionam, mas o número de seqüência (32 *bits*) é algo pequeno para fins de autenticação criptográfica. O uso de *SYN cookies* diminui a aleatoriedade do ISN. Por esse motivo, costuma ser ativado apenas quando há indícios de ataque e os TCBs estão escasseando.

Um "novo" ataque de negação de serviço contra o TCP, na verdade já conhecido há muito tempo, mas considerado apenas um risco teórico, tem se tornado provável com o aumento da velocidade da Internet, tanto nos *backbones* como nos terminais. É o simples disparo de pacotes RST (abortamento de conexão), com origem falseada, contra participantes de uma conexão legítima.

O agente hostil precisa conhecer os endereços IP e números de porta envolvidos, o que só é provável quando as vítimas usam padrões repetitivos de conexão. As principais vítimas em potencial são os roteadores globais da Internet, que utilizam o protocolo BGP, pois este último cria conexões TCP de longa duração e em portas fixas.

A proteção do TCP contra esse tipo de ataque é rejeitar pacotes cujo TSN esteja fora da janela atual. Com um TSN de 32 *bits* e uma janela típica de 4Kbytes, a chance de aceitar um RST falseado é de aproximadamente uma em um milhão (1 em 2^{32-12}) . Porém:

- a disponibilidade de enlaces de banda larga (v.g. ADSL) tornou possível a um usuário doméstico emitir grande quantidade de pacotes por unidade de tempo;
- as conexões TCP tendem a usar janelas cada vez maiores (64Kbytes ou maiores) para máxima performance, aumentando a probabilidade de aceitar um pacote falseado;
- os ataques podem ser (e são) desfechados de forma distribuída, inclusive por vírus.

Tudo isso pode trazer problemas muito sérios ao TCP em breve. Os protocolos MDTP e SCTP foram desde o início projetados com salvaguardas de segurança contra os ataques descritos (STEWART & XIE, 2002).

4-way handshake

Em primeiro lugar, a criação da associação usa 4 pacotes, 2 em cada direção, o que permite um *handshake* mais elaborado. Os últimos dois pacotes de *handshake* já podem transmitir mensagens, de modo a iniciar a transmissão de dados o mais rápido possível.

Em tese, o terceiro pacote de *handshake* TCP também pode transmitir dados. Mas a API *BSD Sockets* retorna de *connect()* apenas depois que o terceiro pacote já foi transmitido. Felizmente, a extensão do *Sockets* para SCTP não apresenta o mesmo problema pois abre as associações de forma diferente.

Os pacotes envolvidos na criação da associação, listados na seqüência normal de transmissão, têm os seguintes nomes (STEWART et al, 2000):

- **INIT**, do cliente para o servidor (”cliente” é quem toma a iniciativa da abertura);
- **INIT-ACK**, do servidor para o cliente em resposta a INIT;
- **COOKIE-ECHO**, do cliente para o servidor. Ao recebê-lo, o servidor considera estabelecida a associação;
- **COOKIE-ACK**, do servidor para o cliente, confirmando o recebimento de COOKIE-ECHO. Ao recebê-lo, o cliente considera estabelecida a associação.

Etiqueta de verificação (*verification tag*)

Na parte fixa da estrutura do pacote SCTP, existe um parâmetro de 32 *bits* denominado etiqueta de verificação (*verification tag*). Cada associação possui duas etiquetas, uma para cada direção. Não existe nada semelhante em TCP.

Nos pacotes INIT e INIT-ACK, cada lado calcula e transmite uma etiqueta de inicialização (*initiation tag*), e dali por diante deve repetir esse valor na etiqueta de verificação de todos os pacotes daquela associação. Todo pacote SCTP deve apresentar a etiqueta correspondente à sua associação e direção, sob pena de ser descartado (STEWART et al, 2000).

Essa etiqueta serve primariamente para identificar encarnações diferentes de uma mesma associação (i.e. entre os mesmos terminais e os mesmos números de porta). Mas também permite discriminar facilmente pacotes forjados, eliminando assim a possibilidade de *blind spoof*, tanto na tentativa de abertura de conexões anônimas quanto na tentativa de seqüestro ou abortamento de uma conexão existente.

O valor da etiqueta deve ser imprevisível para agentes externos, para que a proteção contra *blind spoof* seja efetiva. É o mesmo cuidado que deve ser tomado na geração do ISN do TCP, ou do ISN do SCTP, visto a seguir.

Número de seqüência de transmissão (TSN)

O TCP apresenta o parâmetro TSN (*transmission sequence number* – número de seqüência de transmissão) para indicar, em cada pacote, qual o segmento da seqüência de octetos que está sendo transmitido. Esse valor permite remontar a seqüência e detectar as lacunas. No TCP, o TSN é incrementado pelo número de octetos transmitidos.

O SCTP também apresenta o TSN nos trechos de dados, e sua utilidade é idêntica ao TSN no TCP, exceto pelo fato de ser incrementado por trecho, e não por octeto.

Da mesma forma que no TCP, o TSN inicial (ISN) é informado de parte a parte durante a criação da associação, e tal qual TCP ele deve ser imprevisível por parte de um agente externo. Mas o TCP conta *apenas* com esse recurso para detectar ataques *blind spoof* que visem abortar ou seqüestrar uma conexão.

Já o SCTP conta com o TSN e também com a etiqueta de verificação, tornando-se bem mais resistente.

A RFC 2960 (STEWART et al., 2000) sugere inicialmente que o ISN poderia ser uma duplicata da etiqueta de verificação, pois ambos têm o mesmo tamanho. Porém, STEWART & XIE (2002) recomendam em seu livro que a geração dos dois valores seja independente para aumentar a resistência a ataques.

Cookies

Eliminado o *blind spoof*, resta evitar que o lado passivo da associação SCTP aloque TCBs com associações meio-abertas e fique vulnerável a ataque análogo ao *SYN flood*. Assim como no *SYN Cookie* do TCP, a solução do SCTP é transferir para o cliente a responsabilidade total pelo armazenamento dos dados da associação meio-aberta, através dos *cookies*.

O *cookie* SCTP é uma estrutura de tamanho variável, opaca (i.e. apenas o criador conhece seu formato interno), que o servidor transmite ao cliente no pacote INIT-ACK. O cliente deve retransmitir o *cookie* ao servidor no pacote COOKIE-ECHO, como sugere o próprio nome do pacote.

Se o cliente for na verdade um agente hostil “cego” mandando pacotes forjados, o *cookie* nunca chegará de volta ao cliente, nem será devolvido ao servidor. Como o servidor não ocupa memória com associações meio-abertas, o ataque não satura a tabela de associações e não impede as associações legítimas. Ataques no estilo *SYN flood* não são viáveis contra o SCTP.

Após transmitir o pacote INIT-ACK, o servidor não conserva *nenhuma* informação sobre a associação, nem sobre o potencial cliente, e a associação só é efetivada com o pacote COOKIE-ECHO. Portanto, o servidor depende exclusivamente das informações contidas no *cookie* para criar a associação.

Logo, embora o formato do *cookie* seja de livre escolha, ele tem de conter obrigatoriamente os seguintes dados:

- os dados relevantes do pacote INIT. Normalmente será uma simples transcrição dos dados do pacote;
- os dados relevantes (gerados pelo servidor) do pacote INIT-ACK. Normalmente será também uma simples transcrição;
- duas etiquetas de 32 *bits* denominadas *tie tags*. Normalmente, essas etiquetas contém o valor zero, mas podem ser eventualmente preenchidas com as etiquetas de verificação do cliente e/ou do servidor. Elas servem para identificar retransmissões de pacotes de criação de associação.

Os dados acima são o minimo necessário para a implementação funcionar, porém mais alguns são necessários para garantir a segurança do SCTP:

- *timestamp* para que *cookies* velhos possam ser descartados, bem como para proteção contra ataques de repetição (*replay attacks*);
- assinatura digital que garanta a integridade dos dados, permita ao servidor reconhecer o *cookie* como seu, e atrele o *cookie* ao cliente.

KARN e KRAWCZYK et al. descrevem os mecanismos para a geração de um *cookie* seguro.

A assinatura digital é tipicamente obtida concatenando-se todos os dados do *cookie*, os endereços IP, as portas, uma chave secreta e calculando-se um *hash* de qualidade criptográfica. (KRAWCZYK et al. sugere MD5 ou SHA-1). A RFC 2960 recomenda que a chave secreta seja trocada de forma razoavelmente freqüente. Deve haver uma janela de aceitação da chave velha para evitar que associações em fase de criação sejam incorretamente ilegitimadas.

Obviamente, a chave secreta é concatenada apenas na memória do servidor para fins de cálculo do *checksum*, e não vai fazer parte do *cookie* transmitido.

A criptografia dos dados do *cookie* não é especialmente encorajada, pois não traz qualquer vantagem de segurança; os dados ali contidos poderiam ser obtidos facilmente de outras formas. O *hash* é suficiente para fins de autenticação.

Embora a implementação seja livre para escolher o tamanho do *cookie*, deve procurar escolher o menor tamanho possível para evitar problemas de interoperabilidade. Se considerarmos 16 octetos para os dados relevantes de INIT, mais 16 para os dados de INIT-ACK, 8 para os *tie-tags*, 8 para o *timestamp* e 16 para a assinatura digital, chegamos a um *cookie* de 64 octetos.

Somatório de verificação (*checksum*)

Segundo SHANNON, um *checksum* de n *bits* pode detectar, no máximo:

- 100% dos erros de até n *bits*;
- $2^n - 1$ em cada 2^n erros que afetem aleatoriamente mais de n *bits*.

Este é o melhor desempenho teoricamente possível (o trabalho de Shannon não descreveu *como* criar somatórios eficazes); algoritmos reais terão menor capacidade de detecção. O algoritmo CRC aproxima a expectativa da previsão teórica para erros em rajadas, desde que seu "polinômio gerador" seja bem escolhido.

Isso vale não só para números binários. O dígito verificador presente em números de conta bancária tem as mesmas propriedades: detecta 100% dos erros simples de digitação, e deixa passar 10% dos demais erros – se o algoritmo gerador for de boa qualidade, como o "Módulo 11".

O *TCP checksum*, somatório utilizado em TCP e UDP, é menos eficiente que o CRC na detecção de erros, mas ainda detecta 100% dos erros de 1 *bit* – que constituem a grande maioria dos erros – e é muito mais eficiente no cálculo por *software* que o CRC. (Deve-se levar em conta que essa escolha foi feita numa época em que as CPUs tinham velocidade muito menor que hoje.)

Em média, 1 em cada 5000 pacotes da Internet (0,2%) é entregue com algum erro (ARIAS-RODRIGUEZ *apud* PAXSON). Os meios de transmissão da Internet são muito confiáveis (v.g. Ethernet e FDDI usam CRC-32) e não justificam tamanha quantidade de erros. Os erros são na verdade provocados majoritariamente por problemas nos roteadores – desde memória de má qualidade, interferência eletromagnética até *bugs* na implementação dos respectivos *softwares*.

Um *checksum* de 16 *bits*, por melhor que seja seu algoritmo, deixa passar 1 em 2^{16} dos pacotes errados, para erros completamente aleatórios. Isso significa que 1 em 327.680.000 (5000×65536) dos pacotes da Internet entregues à camada de aplicação contém erros. É uma chance de erro pequena, porém observável, o que obriga a camada de aplicação a implementar algum mecanismo adicional de proteção. HTTP e FTP não possuem tal mecanismo, e portanto não são confiáveis para transmissão de dados sensíveis, se a integridade dos arquivos não é verificada de outra forma.

Como o SCTP tem a pretensão de transmitir mensagens de telefonia, o grau de confiabilidade tem de ser mais alto que o oferecido pelo *checksum* de 16 *bits*. E estabelecer essa confiabilidade no protocolo de transporte desonera as aplicações dessa tarefa.

O *checksum* inicialmente escolhido para o SCTP foi o Adler-32. Pesquisas posteriores determinaram que o poder de detecção de erros desse algoritmo é fraco para mensagens menores que 1 Kbyte (que é o caso do pacote SCTP típico).

O código CRC é antigo, mas ainda é dos mais poderosos na deteção de erros. Sua principal desvantagem é a lentidão no cálculo por *software* (embora seja rápido em *hardware*). Após muita deliberação, optou-se finalmente pelo CRC-32c. Levou-se em conta o grande poder de processamento dos dispositivos modernos. A mudança de *checksum* do SCTP está oficializada na RFC 3309 (STONE et al, 2002).

Quanto à escolha do polinômio gerador, a primeira opção foi o CRC-32 tradicional utilizado em Ethernet e FDDI, porém o polinômio CRC-32c protege melhor mensagens pequenas, de modo que acabou este último sendo o eleito para o SCTP.

Um *checksum* ótimo de 32 *bits* deixa passar apenas 1 em 2^{32} pacotes errados, para erros completamente aleatórios. Levando em conta a taxa de erros da Internet (1 em 5000), o SCTP entregaria à aplicação apenas 1 pacote errado a cada aproximadamente 21.474.836.480.000 trafegados, ou seja, a entrega de uma mensagem corrompida à aplicação é virtualmente impossível.

O somatório CRC não é uma assinatura digital e portanto não protege os dados contra fraude.

Sem garantias criptográficas

Os mecanismos de segurança do SCTP evitam apenas os ataques “cegos”. O SCTP não oferece por conta própria garantias criptográficas (confidencialidade, autenticidade, integridade), nem é resistente a ataques do “homem do meio”. Tais garantias de segurança podem ser providas por outros protocolos, e.g. IPSEC e SSL.

A utilização de SSL com SCTP está descrita na RFC 3436 (JUNGMAIER et al, 2002). As questões de utilização de IPSEC com SCTP em multicaminhos são descritas em BELLOVIN et al.

ANEXO 2: Detalhes de funcionamento do protocolo SCTP

Formato dos pacotes

O pacote SCTP possui um cabeçalho fixo de 4 parâmetros (12 octetos), mais um número variável de trechos (*chunks*) alinhados em 32 *bits*, conforme o esquema abaixo. Os parâmetros fixos estão em negrito.

Cabeçalho IP	
Porta de origem *(16 bits)*	**Porta de destino *(16 bits)***
Etiqueta de verificação *(32 bits)*	
Somatório de verificação *(32 bits)*	
Trecho 1 *(trechos alinhados em 32 bits)*	
Trecho 2	
Trecho n	

Toda a troca de informações de controle do SCTP (abertura de associação, fechamento de associação etc.) é feita através de trechos de tamanho variável, e não *bitmaps* como em TCP.

Os números de porta de origem e destino têm o mesmo fim que em TCP e UDP. A função dos demais campos é detalhada no ANEXO 1.

Notar que as portas estão nas mesmas posições relativas utilizadas por TCP e UDP, o que facilita a interpretação desses números por aplicativos de diagnóstico de rede. Os primeiros 8 octetos do cabeçalho SCTP contém todas as informações necessárias para identificar univocamente uma associação, consoante com as mensagens ICMP que incluem no mínimo 8 octetos do pacote que provocou o erro.

Como em todo protocolo da pilha TCP/IP, os dados que representam números inteiros devem ser representados em *network byte order* – o primeiro octeto é o mais significativo.

Os trechos são estruturas de dados TLV *(type, length, value* – tipo, comprimento e valor). São alinhados em 32 *bits* e podem apresentar até 3 octetos de enchimento. O valor do comprimento inclui os 4 octetos dos três primeiros campos, porém não inclui o enchimento.

Tipo *(8 bits)*	*Flags (8 bits)*	Comprimento *(16 bits)*
Dados úteis do trecho, mais enchimento ($32n$ *bits*)		

O parâmetro *"Flags"* é um mapa de *bits* interpretado de acordo com o tipo de trecho (não há nenhum *flag* com significado genérico).

Como o SCTP foi projetado para ser extensível, pode suceder de um tipo de trecho presente em uma implementação não ser suportado por outras implementações. A reação do receptor a um trecho não suportado depende dos dois *bits* MSB do tipo de trecho:

Tipo	*Ação tomada pelo receptor, se não suporta o trecho*
0x00 a 0x3F	descartar o pacote que contém o trecho
0x40 a 0x7F	descartar o pacote e remeter uma notifi cação de erro
0x80 a 0xBF	desprezar o trecho e continuar a processar o pacote
0xC0 a 0xFF	desprezar o trecho, continuar a processar o pacote, e remeter uma notifi - cação de erro

O código do tipo deve ser escolhido em função da reação desejada.

No espaço de dados úteis, os trechos predefinidos pelo protocolo têm parâmetros *permanentes* e *opcionais*.

Os parâmetros permanentes têm tamanho e ordem predefinidos para cada tipo, tal qual uma estrutura da linguagem C, e sempre vêm primeiro em relação aos parâmetros opcionais. O bloco de parâmetros permanentes deve ser alinhado em 32 *bits*.

Em seguida vêm os parâmetros opcionais, em número variável, que também são estruturas TLV:

Tipo de parâmetro *(16 bits)*	Comprimento *(16 bits)*
Dados úteis do parâmetro, mais enchimento ($32n$ *bits*)	

Assim como nas estruturas TLV dos trechos, o comprimento do parâmetro inclui os 4 octetos do tipo e do próprio comprimento, mas não inclui os octetos de enchimento.

Analogamente ao tipo de trecho, os dois *bits* MSB do tipo de parâmetro também especificam a reação a um parâmetro desconhecido pelo receptor:

Tipo de parâmetro	*Ação tomada pelo receptor, se não suporta o parâmetro*
0x0000 a 0x3FFF	descartar o trecho que contém o parâmetro
0x4000 a 0x7FFF	descartar o trecho e remeter uma notifi cação de erro
0x8000 a 0xBFFF	desprezar o parâmetro e processar o trecho
0xC000 a 0xFFFF	desprezar o parâmetro, processar o trecho, e remeter uma notifi cação de erro

Motivação de uso da estrutura TLV

A estrutura TLV é simples de entender, implementar e processar, e sem dúvida muito mais versátil que um *bitmap* fixo. Alguns testes realizados por Randall Stewart (um dos criadores do SCTP), citados por ARIAS-RODRIGUEZ, determinaram que o processamento de uma estrutura TLV é mais rápido que o de um *bitmap*.

O bitmap não é de todo desvantajoso. Ele pode ser muito pequeno, reduzindo a sobrecarga de rede. Protocolos como IP e TCP utilizam bitmaps, pois, à época de sua criação, as baixas velocidades dos enlaces justificavam qualquer esforço de diminuição da sobrecarga.

A situação atual, segundo TANENBAUM, é oposta - a vazão é limitada predominantemente pela latência de processamento dos terminais, e a redução dessa latência deve ser a principal preocupação dos projetistas de protocolos. Os protocolos devem ser simples, ágeis no processamento e extensíveis.

Desta forma, o SCTP utiliza majoritariamente estruturas TLV para transporte de dados e informações de controle. Até mesmo tarefas como abertura e fechamento de associação utilizam tais estruturas, e não *flags* (*e.g.* ACK, SYN, FIN do TCP).

Suporte ao SCTP em aplicativos-ferramentas

A alocação de todas as informações em estruturas TLV torna mais difícil a interpretação dos pacotes SCTP por parte de ferramentas de diagnóstico de rede. Não basta enquadrar o pacote numa estrutura C; a ferramenta precisa conhecer suficientemente o protocolo para separar e interpretar os TLVs.

Como o próprio *payload* está encapsulado em uma estrutura TLV, a figura didática da separação rígida entre cabeçalho de transporte e *payload*, não existe em SCTP. Um pacote SCTP pode ter dois ou mais trechos de dados, então haverá dois *payloads* incrustados dentro das estruturas de controle SCTP.

Em TCP e UDP, a tupla formada pelos pelos endereços e portas dos terminais identifica univocamente uma conexão, mesmo na Internet. Pode-se começar a monitorar uma conexão a qualquer momento, e individualizá-la imediata e facilmente. Em SCTP, o suporte a multicaminhos torna mais difícil a individualização da associação. Para uma identificação perfeita, o aplicativo tem de monitorar a associação desde sua criação, para conhecer os endereços alternativos informados de parte a parte.

Se o monitoramento começar com a associação multicaminhos já estabelecida, a individualização será imperfeita. As tuplas formadas pelas etiquetas de verificação podem

ser usadas como dado auxiliar de individualização, mas não são perfeitas pois existe sempre a (na verdade muito pequena) probabilidade de associações diferentes apresentarem as mesmas etiquetas.

Embora não sejam problemas insolúveis, certamente os fatores acima têm atrasado o suporte a SCTP em aplicativos-ferramentas, bem como em equipamentos ativos de rede.

Decisões de implementação

Conforme a RFC 2960 (STEWART et al, 2000), há algumas decisões deixadas à implementação do protocolo SCTP.

Maior mensagem suportada

Muito embora o tamanho máximo teórico da mensagem seja de $2^{32} - 1$ octetos, a implementação pode estabelecer um limite prático menor. Um dos limites é o tamanho máximo do *buffer* de recepção, que está na memória do *kernel* e sofre restrição relativamente severa de tamanho. Outro limite prático é imposto pelo PTMU da rede e pelos 16 *bits* do SSN – uma mensagem pode ser fragmentada em no máximo 2^{16} trechos, o que limita seu tamanho a $2^{16} \times PMTU$.

É também facultada à implementação limitar a mensagem ao PMTU, de modo que caiba integralmente em um datagrama e dispense o algoritmo de remontagem de mensagens.

Dispositivos de memória restrita como celulares podem determinar tais restrições. Mensagens SS7 podem ter, no nível de aplicação, no máximo 272 ou 4091 octetos, dependendo do meio de transmissão (ONG et al, 1999); dispositivos que utilizem SCTP exclusivamente para sinalização de telefonia podem ater-se a esses limites.

Formato do *cookie*

O tamanho e forma de geração do *cookie* de autenticação é de responsabilidade da implementação. As RFCs 2522 e 2104 sugerem algoritmos e fontes de dados para ele.

A implementação também pode restringir o tamanho máximo do *cookie* recebido, por limitações de memória etc. Isso limita sua interoperabilidade com outras implementações que usem *cookies* maiores.

Por esse motivo, e prevendo que o SCTP possa ser usado em dispositivos naturalmente limitados em capacidade de memória e processamento, a RFC 2960 recomenda que a implementação crie o menor *cookie* possível.

Suporte a multicaminhos

A implementação de multicaminhos é opcional em cada uma de suas modalidades (endereços IPv4, endereços IPv6 e nomes DNS).

Por exemplo, uma implementação pode suportar apenas multicaminhos com endereços IPv4. Isso é interessante se o sistema operacional subjacente não suporta IPv6 – se todas as modalidades fossem obrigatórias, sistemas sem IPv6 estariam impedidos de implementar SCTP.

Também é possível que algumas implementações optem por não suportar multicaminhos baseado em endereços DNS, por questões de segurança. Ainda, em alguns dispositivos, não faz nenhum sentido usar multicaminhos.

Suporte a confiabilidade parcial (PR–SCTP)

A extensão PR–SCTP modifica a semântica do prazo de validade das mensagens. Uma mensagem cujo prazo de validade estoure, é descartada mesmo que sua transmissão já tenha sido tentada. Ela é de implementação opcional.

Convivência do SCTP com outras tecnologias

A grande maioria dos problemas de convivência do SCTP com outras tecnologias de rede TCP/IP são derivadas do recurso de multicaminhos.

IPv6

As versões iniciais do SCTP previam apenas multicaminhos para IPv4, o que era uma grande limitação. Felizmente, o suporte a multicaminhos IPv6 foi introduzido numa das revisões do protocolo. O SCTP permite até mesmo mistura de endereços IPv4 e IPv6 para multicaminhos de uma mesma associação.

Uma observação importante, encontrável na RFC 2960, é que não se pode usar um parâmetro de endereço IPv6 contendo um endereço IPv4 mapeado em IPv6 **(**`::FFFF:0/96`**)**. Para endereços IPv4, deve-se utilizar o parâmetro de endereço IPv4, contendo o endereço IP de 32 *bits*.

Roteadores NAT

NAT para SCTP sem multicaminhos tem o mesmo grau de dificuldade que TCP. A priori não faz sentido fazer NAT para SCTP com multicaminhos, pois há muitas variáveis envolvidas:

- Haverá apenas um roteador NAT envolvido, ou um NAT para cada caminho?
- Se houver mais de um roteador NAT envolvido, como os NAT secundários ficarão sabendo da associação, se os pacotes INIT, INIT-ACK etc. trafegaram apenas pelo NAT primário?
- Quem é multicaminhos: o terminal ou o roteador NAT?
- Existe mistura de endereços válidos e inválidos?
- Quem define que endereços devem ou não ser traduzidos?

No caso específico do terminais “ocultos” monocaminhos, e um único roteador NAT *multi-homed*, existe um potencial interessante de aumentar a tolerância a falhas. Conforme a alternativa proposta por COENE, o protocolo SCTP foi estendido para aceitar endereços *multi-homed* em forma de nome DNS.

Os computadores "ocultos" podem tirar proveito do roteador NAT *multi-homed*, desde que informem como caminhos alternativos os nomes DNS do roteador NAT. Mais fácil ainda é associar um único nome DNS a todos os endereços IP dos caminhos.

É responsabilidade do terminal remoto fazer a resolução DNS do nome. Como o SCTP deve em geral ser implementado no *kernel* dos sistemas operacionais, e o protocolo DNS em geral está fora do *kernel*, é provável que as implementações escolham entre:

- não suportar multicaminhos com nomes DNS;
- criar um processo-agente que resolva nomes DNS para o *kernel*, com alguma provisão de segurança contra ataques de negação de serviço.

IPSEC

O IPSEC autentica (cabeçalho AH) e opcionalmente criptografa (cabeçalho ESP) cada pacote IP de modo que:

- o receptor tenha certeza de que o pacote veio da origem alegada, e que o pacote não foi forjado, adulterado ou danificado;
- a origem não possa negar que emitiu o pacote, nem negar a autoria dos dados contidos nele;
- opcionalmente, haja confidencialidade dos dados;

- opcionalmente, haja confidencialidade dos endereços IP dos terminais (possível com o modo túnel).

Para oferecer essas garantias, o IPSEC faz uso de chaves públicas, e há troca de chaves secretas entre as partes antes de qualquer comunicação de dados do usuário. As chaves secretas são atreladas aos endereços IP das partes.

O IPSEC opera em dois modos distintos:

- *Modo transporte*: o IPSEC empacota apenas o *payload* (dados úteis) do datagrama IP. Este modo não pode garantir a confidencialidade do cabeçalho IP, porém permite estabelecer um canal seguro de comunicação entre dois terminais IPSEC quaisquer da Internet;
- *Modo túnel*: o IPSEC reempacota inteiramente o datagrama IP, formando um túnel ou VPN. Este modo pode garantir a confidencialidade do cabeçalho IP, mas exige providências administrativas explícitas nas duas extremidades do túnel.

Nada disso interfere com os protocolos de transporte, nem impede que se use o SCTP com IPSEC.

O problema surge na interação do modo transporte com multicaminhos. Toda a negociação da associação SCTP (e a negociação IPSEC que a precede) é feita entre os IPs primários dos terminais. Se, durante a associação, o IP primário falhar, o SCTP tentará mandar dados para um IP secundário do terminal remoto, que rejeitará os pacotes pois há discrepância entre o código de autenticação e o endereço IP de destino.

A alternativa mais fácil, que funciona com o IPSEC sem modificações, é efetuar a negociação IPSEC entre todos os endereços envolvidos. Esse produto cartesiano pode não ser adequado se o número de endereços é grande – se ambos os terminais têm 5 endereços IP diferentes, teria de haver 25 negociações IPSEC, que envolvem busca de chave pública e cálculos relativamente demorados.

Outra alternativa é estender o IPSEC para tratamento de multicaminhos, possibilitando uma única negociação válida a todos os endereços da associação.

A solução definitiva para este problema ainda está em discussão (BELLOVIN et al.). O mais provável é que a escolha da estratégia seja delegada às implementações.

ANEXO 3: Extensões da interface *BSD Sockets* para suporte ao protocolo SCTP

O documento autoritativo das extensões à interface *BSD Sockets* para suporte a SCTP, é STEWART et al. (2004a). Este documento ainda é um rascunho *(draft)*, sujeito a revisões antes de ser promovido a RFC. Apesar disso, as extensões já foram implementadas em diversos sistemas operacionais.

A aplicação pode fazer uso do SCTP através de dois tipos de soquetes:

- **Estilo TCP:** cada soquete corresponde a uma associação, que funciona de forma semelhante a um soquete de conexão TCP.
- **Estilo UDP:** várias associações atreladas a um soquete. Existe certa semelhança com um soquete UDP pois este último permite trocar pacotes com qualquer terminal remoto.

Objetiva-se com isso oferecer conveniência para aplicativos novos, e facilidade de conversão de aplicativos existentes (a maioria deles simplesmente migrando de TCP para SCTP sem quaisquer mudanças no protocolo de aplicação).

Estilo TCP

O estilo TCP torna extremamente simples a conversão de programas que já utilizem TCP. Em geral, é preciso mudar apenas o terceiro parâmetro passado a *socket()*; nenhuma outra mudança é necessária.

```
// cliente
sockaddr_in servidor;
int fd = socket(AF_INET, SOCK_STREAM, IPPROTO_SCTP);
servidor.sin_family = AF_INET;
servidor.sin_addr.s_addr = inet_addr("10.0.1.1");
servidor.sin_port = 12345;
connect(fd, (sockaddr&) &servidor, sizeof(servidor));
// ... comunica-se com servidor com send() e recv() ...
close(fd);
```

```
// servidor
sockaddr_in serv, cliente;
int fd = socket(AF_INET, SOCK_STREAM, IPPROTO_SCTP);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = INADDR_ANY;
serv.sin_port = htons(12345);
bind(fd, (sockaddr*) &serv, sizeof(serv));
listen(fd, 5);
while (true) {
       int fd_conexao = accept(fd, (sockaddr*) &cliente,
                               sizeof(cliente));
       // ... trata cliente com send() e recv() ...
       close(fd_conexao);
}
```

A função *shutdown()* é suportada em SCTP porém simplesmente fecha a conexão sem liberar o manipulador de arquivo. SCTP não suporta o estado meio-fechado.

As funções *send()* e *write()* transmitem os dados pelo fluxo zero. As funções *recv()* e *read()* recebem mensagens vindas de qualquer fluxo. O aplicativo não tem como saber por que fluxo veio a mensagem. Se o terminal remoto utilizar múltiplos fluxos ou mensagens urgentes, a ordem de entrega dos dados é imprevisível.

Cada chamada a *send()* gera uma mensagem atômica, semelhante a *sendto()* em UDP. A atomicidade das mensagens é respeitada por *recv()*, desde que o *buffer* de recepção seja suficientemente grande, sob pena da mensagem ser entregue parcialmente (o que destrói sua atomicidade).

O aplicativo pode utilizar *select()* ou *poll()* para aguardar transmissão e recepção, tal qual faria em TCP.

É possível utilizar diversos fluxos e mensagens urgentes no estilo TCP, mas o aplicativo deverá utilizar as funções avançadas de I/O, abordadas mais adiante neste anexo. Mesmo utilizando tais funções, o aplicativo não receberá informações avançadas do estado da associação (e.g. estado de cada caminho). Tais informações são visíveis apenas no estilo UDP.

Para configurar o número de fluxos de uma nova associação neste estilo, é necessário recorrer a *setsockopt(SCTP_INIT)*.

Estilo UDP

O estilo UDP oferece um paradigma mais interessante para o tratamento de grande número de associações.

Uma característica interessante desse estilo é que o aumento na complexidade do código-fonte é aproximadamente proporcional ao nível de controle desejado sobre as associações e mensagens. As diferenças entre os soquetes “SCTP estilo UDP” e “UDP legítimo” também vão aumentando.

```
sockaddr_in local, remoto;
int fd = socket(AF_INET, SOCK_SEQPACKET, IPPROTO_SCTP);
local.sin_family = AF_INET;
local.sin_addr = INADDR_ANY;
local.sin_port = htons(54321);
bind(fd, (sockaddr*) &local, sizeof(local));
// aceitar associações passivamente (como servidor)
// um cliente puro não faria isto
listen(fd, 5);
// timeout de 2 minutos (120 segundos)
int timeout = 120;
setsockopt(fd, SCTP_AUTOCLOSE, &timeout);
remoto.sin_family = AF_INET;
remoto.sin_addr.s_addr = inet_addr(“10.0.1.1”);
remoto.sin_port = htons(12345);
char *mensagem = “a mensagem”;
sendto(fd, (sockaddr*) &remoto, mensagem, strlen(mensagem)+1);

// ... comunica-se com outros terminais
// usando sendto() e recvfrom() ...
close(fd);
```

Não há criação explícita de associação. As associações são abertas sob demanda, e ficam todas atreladas a um único soquete. Elas são fechadas automaticamente por desuso; o *timeout* é configurado através de *setsockopt(SCTP_AUTOCLOSE).*

O aplicativo fica livre de toda a burocracia de tratamento das associações, mesmo que faça uso de milhares delas. O uso de um único manipulador de arquivo também aumenta a escalabilidade do aplicativo.

Como não é necessário chamar *connect()* para criar a associação, a implementação pode usar os pacotes COOKIE-ECHO e COOKIE-ACK para transmitir dados, o que colabora para diminuir a latência. De qualquer forma, se o aplicativo deseja abrir uma associação sem mandar dados imediatamente, ainda pode usar *connect()* para esse fim.

O TCP também pode, em tese, transmitir dados no terceiro pacote de *handshake*, mas a chamada a *connect()* só retorna depois da conexão completamente aberta (STEVENS et al, 2004).

Se o aplicativo deseja fazer abertura passiva de associações (i.e. atuar como servidor), deve chamar *listen()* para o soquete. (Note que essa chamada não faria sentido para UDP.)

Em alguns aplicativos de servidor, em particular aplicativos *multithreaded*, pode ser válido destacar uma associação para seu próprio manipulador de arquivo. Para esse fim, foi criada uma nova função, *sctp_peeloff()*.

Uma vez destacada, a associação não mais pode ser acessada pelo soquete genérico, nem será automaticamente fechada por desuso. Apenas associações podem ser destacadas; não há forma de destacar um fluxo em particular.

A função *sendto()* transmite os dados pelo fluxo zero. A função *recvfrom()* recebe mensagens vindas de qualquer fluxo; o aplicativo não terá como saber de que fluxo veio a mensagem.

Cada chamada a *sendto()* gera uma mensagem, tal qual UDP. A atomicidade das mensagens é respeitada por *recvfrom()*, desde que o *buffer* de recepção seja suficientemente grande, sob pena de entrega parcial e perda da atomicidade.

O fechamento do soquete genérico fecha todas as associações atreladas. Para fechar uma associação em particular, o aplicativo deve usar o estilo UDP avançado, ou então fechar o manipulador destacado por *sctp_peeloff()*.

Como o soquete genérico está ligado a muitas associações, não faz sentido usar *select()* para aguardar o momento da transmissão. Não seria possível saber que associações estão prontas para transmitir. A solução mais elegante é usar I/O não-bloqueante.

Pode ser inclusive perigoso a um aplicativo *single-threaded* usar o soquete genérico em modo bloqueante, pois *sendto()* bloqueia se o *buffer* de transmissão da associação estiver cheio. E enquanto o aplicativo dorme, milhares de outras associações podem estar esperando para entregar mensagens (STEWART et al, 2004a).

Por essas razões, provavelmente toda aplicação deveria usar o soquete genérico em modo não-bloqueante.

Uma alternativa menos escalável, mas confortável ao desenvolvedor que não quer lidar com I/O não-bloqueante, é destacar todas as associações com *sctp_peeloff()* e fazer *select()* sobre todos os manipuladores (inclusive sobre o genérico, pois é através dele que criam-se novas associações ativa ou passivamente).

Interface avançada de I/O

Nos estilos TCP e UDP básicos, não existe forma de especificar o fluxo de transmissão, ou conhecer por que fluxo veio uma mensagem, ou ainda de marcar uma mensagem

como urgente Para ter acesso a todos os recursos do SCTP, o aplicativo enviar e receber dados através da “coleta de dados auxiliares” (funções *sendmsg()* e *recvmsg()*). Uma descrição pormenorizada do uso de coleta de dados auxiliares pode ser encontrada em STEVENS et al. (2004).. Essa interface é mais complexa, porém permite:

- tanto no estilo TCP quanto no estilo UDP, transferir mensagens através de estruturas, que (além dos dados) contém os parâmetros SCTP: fluxo, urgência, identificação de protocolo etc;
- no estilo UDP, receber e enviar eventos, como abertura e fechamento de associações, mudança no estado do enlace etc. para que o aplicativo possa ter, se quiser, controle total sobre o andamento das associações;
- no estilo UDP, abrir associações com número de fluxos, *timeout* de associação etc. escolhidos pela aplicação;
- no estilo UDP, fechar associações manualmente sem recorrer a *sctp_peeloff()*.

No estilo UDP o uso de *sendto()* e *recvfrom()* pode criar ambiguidades, pois estes identificam a associação pelo endereço e porta do terminal remoto, o que pode ser ambíguo no caso de associações com multicaminhos. No I/O avançado, cada associação tem um identificador opaco de 32 *bits*, o que resolve o problema. (Esse problema não diz respeito ao estilo TCP pois neste é o soquete quem identifica univocamente a associação.)

O I/O avançado também evita a destruição da atomicidade caso a mensagem seja maior que o *buffer* de recepção oferecido pelo aplicativo. A entrega parcial de mensagem é sinalizada ao aplicativo, que pode obter e tratar os diversos fragmentos.

Flags de mensagem

A coleta de dados auxiliares recebe/entrega uma variável em forma de *bitmap*, que em SCTP pode conter os seguintes *flags*:

MSG_NOTIFICATION	*Buffer* de transmissão/recepção contém um evento e não dados
MSG_EOR	Mensagem foi inteiramente entregue, ou último fragmento foi entregue
MSG_CTRUNC	Dados auxiliares truncados e perdidos (*buffer* é insufi ciente)
MSG_EOF	Indica que a associação deve ser fechada *(sendmsg())*

Mensagens auxiliares

As mensagens auxiliares são mandadas ou recebidas em paralelo com dados úteis. Pode haver diversas mensagens auxiliares, de diversos protocolos diferentes (e.g. tanto a camada IP quanto SCTP entregam mensagens auxiliares referentes a uma mensagem da aplicação). As mensagens próprias do SCTP são apenas duas:

SCTP_INIT

Essa mensagem serve apenas para criar associações explicitamente através de *sendmsg()*, e nunca é recebida. A estrutura da mensagem é:

```
struct sctp_initmsg {
        uint16_t sinit_num_ostreams;
        uint16_t sinit_num_instreams;
        uint16_t sinit_max_attempts;
        uint16_t sinit_max_init_timeo;
}
```

sinit_num_ostreams	Número de fluxos de transmissão desejado.
sinit_num_instreams	Número máximo de fluxos de recepção aceito pela aplicação.
sinit_max_attempts	Número máximo de tentativas para estabelecer a associação.
sinit_max_init_timeo	*Timeout* para o estabelecimento da associação.

SCTP_SNDRCV

Essa mensagem acompanha mensagens de dados úteis, tanto na transmissão quanto na recepção. A cada mensagem de dados transmitida ou recebida, corresponde uma e apenas uma mensagem auxiliar SCTP_SNDRCV. A estrutura da mensagem é:

```
struct sctp_sndrcvinfo {
        uint16_t sinfo_stream;
        uint16_t sinfo_ssn;
        uint16_t sinfo_flags;
        uint32_t sinfo_ppid;
        uint32_t sinfo_context;
        uint32_t sinfo_timetolive;
        uint32_t sinfo_tsn;
```

```
        uint32_t sinfo_cumtsn;
        sctp_assoc_t sinfo_assoc_id;
};
```

sinfo_stream	Número do fluxo de transmissão ou recepção.
sinfo_ssn	Número seqüencial da mensagem no fluxo (apenas recepção).
sinfo_flags	Mapa de *bits* com os seguintes valores possíveis: • *MSG_UNORDERED*: mensagem urgente (não ordenada); • *MSG_ADDR_OVER*: requisita à implementação para usar o endereço especificado em *sendmsg()/sendto()* ao invés do endereço primário; • *MSG_ABORT*: Faz a associação ser abortada; • *MSG_EOF*: Faz a associação ser fechada (*shutdown*); os dados em fila dos dois lados serão transmitidos antes do fechamento.
sinfo_ppid	Identificador de protocolo (passado e interpretado pelas aplicações, não é utilizado pelo SCTP em si).
sinfo_context	Valor passado pela aplicação, que será devolvido em caso de erro de transmissão.
sinfo_timetolive	Tempo de vida da mensagem em ms. Se a mensagem vencer ainda na fila de transmissão, não será mais mandada. O valor zero significa tempo de vida "infinito". Na extensão PR-SCTP, o tempo de vida muda ligeiramente de semântica; a mensagem não será mais transmitida mesmo que já tenha havido tentativas de transmissão.
sinfo_tsn	Valor do TSN do trecho de dados (apenas recepção).
sinfo_cumtsn	Valor do TSN cumulativo atual (apenas na recepção de mensagens urgentes).
sinfo_assoc_id	Identificador *(handle)* da associação.

O tipo *sctp_assoc_t* é um *handle* de 32 *bits* opaco (não deve ser interpretado como número). Toda mensagem, seja de dados, seja de evento, usa esse *handle* para identificar unicamente cada associação. A aplicação usuária do estilo UDP avançado deve prestar atenção às mensagens de criação de associações, e anotar o *handle* para futuras referências.

Eventos

Os eventos são mensagens especiais, lidas/gravadas no próprio *buffer* de transmissão/recepção de dados. Como são devidamente sinalizadas por MSG_NOTIFICATION e nunca ocorrem juntamente com dados da aplicação, podem ser interpretadas sem ambiguidade.

Todos os eventos têm um cabeçalho em comum em sua estrutura:

```
struct {
        uint16_t sn_type;
        uint16_t sn_flags;
        uint32_t sn_length;
} sn_header;
```

Através do valor de *sn_type*, sabe-se o tipo de evento e implicitamente a estrutura de dados correspondente a ele. Os tipos de evento possíveis são:

SCTP_ASSOC_CHANGE	Sinaliza que uma associação foi aberta ou fechada.
SCTP_PEER_ADDR_ CHANGE	O endereço que é parte de uma associação aberta sofreu mudança de estado (falha ou reabilitação).
SCTP_REMOTE_ERROR	O terminal remoto retornou um erro operacional, por conta de um tipo de trecho não suportado. Inclui o trecho que causou o erro.
SCTP_SEND_FAILED	Mensagem não pôde ser enviada. Inclui a estrutura *sctp_sndrecvinfo* que iria ser enviada.
SCTP_SHUTDOWN_ EVENT	O terminal remoto remeteu um pacote SHUTDOWN. Não se pode mandar mais dados para essa associação.
SCTP_ADAPTION_ INDICATION	O terminal remoto requisitou uma determinada camada de adaptação.
SCTP_PARTIAL_ DELIVERY_EVENT	Entrega parcial abortada – pode indicar uma associação que está para ser abortada.

Opções de soquete (*setsockopt*)

Toda opção de soquete envolve a passagem de um valor inteiro ou de uma estrutura, conforme a necessidade.

SCTP_RTOINFO	Alteração dos parâmetros de RTO (*timeout* de retransmissão). A implementação deve impor limites a usuários não-privilegiados na alteração desses parâmetros.
SCTP_ASSOCINFO	Obtenção e alteração de diversos parâmetros da associação e do terminal remoto e.g. janela de transmissão e recepção
SCTP_INITMSG	Alterações dos parâmetros padrão utilizados em associações automaticamente criadas
SO_LINGER	Utilizado em soquetes estilo TCP para abortar uma conexão ao invés do SHUTDOWN normal.

SCTP_NODELAY	Ligar/desligar o algoritmo Nagle. No caso do SCTP, esse algoritmo tem a função de empacotar diversos trechos pequenos em um datagrama. Com o Nagle desligado, todo trecho é mandado assim que solicitado.
SO_RCVBUF	Tamanho da janela de recepção *(rwnd)*.
SO_SNDBUF	Tamanho do *buffer* de transmissão.
SCTP_AUTOCLOSE	(Apenas para soquetes estilo UDP.) Controla o tempo de fechamento automático de associações em desuso, em segundos. O valor zero desliga o fechamento automático.
SCTP_SET_PRIMARY_ ADDR	Solicita que o terminal remoto considere o endereço passado como o primário. (Opcional – depende da extensão AddIP)
SCTP_SET_PEER_ PRIMARY_ADDR	Solicita que o sistema local use o endereço passado como o endereço primário do terminal remoto. (Opcional – depende da extensão AddIP)
SCTP_SET_ADAPTION_ LAYER	Requisita que o sistema local emita o trecho *"adaption layer indication"*, com o valor especifi cado, para novas associações. O terminal remoto receberá a mensagem SCTP_ADAPTION_INDICATION.
SCTP_DISABLE_ FRAGMENTS	Se habilitado, nenhuma mensagem será fragmentada em diversos trechos, e tem de caber no PMTU sob pena de um erro ser retornado à aplicação.
SCTP_SET_PEER_ ADDR_PARAMS	Manipulação dos *heartbeats* para os endereços remotos.
SET_DEFAULT_SEND_ PARAM	Parâmetros padrão da estrutura *sctp_sndrcvinfo* para as remessas que utilizem a interface simplifi cada *sendto()*.
SCTP_SET_EVENTS	Permite especifi car os eventos que a aplicação está interessada em receber.
SCTP_I_WANT_ MAPPED_ V4_ ADDR	Se esta opção estiver habilitada e o soquete for IPv6, os endereços IPv4 serão representados como mapeados em IPv6. Do contrário, recebe tanto endereços IPv4 quanto IPv6.
SCTP_MAXSEG	Limita o tamanho máximo de qualquer trecho. Trechos maiores serão fragmentados.
SCTP_STATUS	(somente leitura) Obtém estado e informações sobre uma associação.
SCTP_GET_PEER_ ADDR_INFO	Obtém informações sobre um endereço específi co do terminal remoto.

Atrelamento (*bind*) das interfaces de rede

O aplicativos-clientes TCP (e alguns aplicativos-clientes UDP) optam geralmente em não chamar a função *bind()*. O sistema decide pela tabela de roteamento qual é a

interface de rede mais “próxima” do destino; e atribui uma porta de origem efêmera, aleatória, que não conflite com outras conexões e com serviços bem conhecidos.

Na maioria dos clientes UDP, e na quase totalidade de servidores de TCP e UDP, é mais comum chamar-se *bind()* com o pseudo-endereço *INADDR_ANY* (IPv4) ou *in6addr_any* (IPv6), que atrela o soquete a todas as interfaces de rede da máquina.

Porém, no caso do SCTP, não chamar *bind()*, ou chamá-lo com *INADDR_ANY* ou *in6addr_any* tem uma semântica diferente: é facultada à implementação utilizar automaticamente *todas* as interfaces de rede disponíveis para fins de multicaminhos.

O sistema pode optar em usar todas as interfaces, ou um subconjunto ótimo. STEWART et al. (2004a) não define o que é “ótimo”, ou seja, não define os critérios de escolha das interfaces de rede; isso fica a cargo da implementação.

Essa mudança de semântica vale tanto para clientes como para servidores. O desenvolvedor deve ficar atento para eventuais problemas com redes com endereços privados (e.g. 10.0.0.0/8) ou inválidos.

O livro de STEWART & XIE (2002) sugere que as implementações permitam ao administrador do sistema definir as faixas proibidas para multicaminhos. As faixas proibidas padrão seriam as faixas de endereçamento privado para IPv4, e as faixas de escopo local para IPv6.

A função *bind()* suporta apenas um endereço (ou absolutamente todos, via *INADDR_ANY*), e não é permitido chamar *bind()* repetidas vezes para especificar vários endereços. Se uma aplicação SCTP quer utilizar um subconjunto específico de interfaces de rede para multicaminhos, deve usar a função *sctp_bindx()*, que faz as vezes de *bind()* e aceita uma lista de endereços.

No protocolo UDP, que não tem suporte a multicaminhos embutido, as aplicações contornam essa limitação criando um soquete para cada interface de rede. Um aplicativo servidor SCTP que deseja atender em todas as interfaces de rede, mas não queira usar multicaminhos, deve adotar a mesma estratégia.

Relação de interfaces completamente novas

Quase todas as novas funções têm relação com o suporte a multicaminhos.

sctp_peeloff()	No estilo UDP, separa do soquete genérico uma determinada associação, dando-lhe um manipulador de arquivo exclusivo
sctp_bindx()	Uma versão de *bind()* que permite especifi car uma lista de endereços, para controle fi no do multicaminhos.

sctp_connectx()	Uma versão de *connect()* que permite especifi car uma lista de endereços, para controle fi no do multicaminhos.
sctp_getpaddrs()	Obtém todos os endereços remotos de uma associação. Retorna uma matriz dinamicamente alocada.
sctp_freepaddrs()	Libera da memória a matriz retornada por *sctp_getpaddrs()*.
sctp_getladdrs()	Obtém todos os endereços locais de uma associação. Retorna uma matriz dinamicamente alocada.
sctp_freeladdrs()	Libera da memória a matriz retornada por *sctp_getladdrs()*.
sctp_sendmsg()	Função oferecida opcionalmente como alternativa a *sendmsg()*. Não deve ser usada para associações multicaminhos.
sctp_recvmsg()	Função oferecida opcionalmente como alternativa a *recvmsg()*. Não deve ser usada com associações multicaminhos.
sctp_opt_info()	Certas chamadas *getsockopt()* SCTP implicam em passagem de informações através do *buffer* destinado ao retorno. Em algumas implementações, essa passagem de informações não é permitida. A chamada *sctp_opt_info()* deve então ser usada.

Macros

Algumas macros identificam a presença do protocolo SCTP e suas extensões. O recurso existe se a macro é definida e tem o valor 1.

HAVE_SCTP	Existe implementação do SCTP no sistema.
HAVE_KERNEL_SCTP	A implementação do SCTP está no *kernel* e pode ser acessada pela interface *BSD Sockets*. (Uma implementação *user-level* tem de ser acessada por funções de biblioteca).
HAVE_SCTP_PRSCTP	A extensão PR−SCTP é suportada.
HAVE_SCTP_ADDIP	A extensão AddIP é suportada.
HAVE_SCTP_ CANSET_PRIMARY	A extensão de mudança dinâmica do IP primário é suportada.
HAVE_SCTP_SAT_ NETWORK_CAPABILITY	A extensão de suporte a redes por satélite é suportada.
HAVE_SCTP_MULTIBUF	No estilo UDP, implementa um *buffer* por associação ao invés de um *buffer* único. O *buffer* único pode causar problemas em aplicações que remetem quantidade excessiva de dados sem aguardar confi rmação.
HAVE_SCTP_ NOCONNECT	No estilo TCP, permite que a função *sendto()* inicie automaticamente a associação.

Fatores determinantes na escolha do estilo TCP ou UDP

O estilo UDP fornece mais ferramentas de controle da associação. Aplicações de sinalização telefônica, que motivaram a criação do SCTP, têm no estilo UDP uma interface mais apropriada.

O estilo TCP facilita muito a adição de suporte ao SCTP em aplicações que já usam o protocolo TCP. Ele também fornece uma interface mais conhecida e mais fácil de usar, permitindo que o desenvolvedor mantenha inalterado o uso de I/O bloqueante, *select()*, *poll()*, subprocessos, *threads*, passagem de soquetes para outros processos etc.

A adaptação de um aplicativo existente ao estilo UDP é muito mais difícil, e o suporte simultâneo de TCP e SCTP impõe duplicação de código. O estilo UDP é claramente voltado para aplicativos novos, e que usem exclusivamente SCTP como transporte.

Para a maioria das aplicações em uso na Internet, o nível de controle oferecido pelo estilo UDP não é necessário, sendo suficiente a interface apresentada pelo estilo TCP, que dá acesso aos recursos interessantes do SCTP *e.g.* multicaminhos, fluxos e mensagens com prazo de validade.

O estilo UDP faz uso de um único soquete para todas as associações, o que alivia as aplicações com grande número de associações simultâneas. No entanto, a substituição de *select()* por *poll()* e */dev/epoll* permite também ao estilo TCP suportar de forma escalável um número virtualmente ilimitado de soquetes.

O estilo UDP não permite o uso confiável de *select()/poll()* para transmissão; é necessário usar I/O não bloqueante. Aplicações que transmitam conteúdos de grande volume e sem estrutura (e.g. arquivos) são muito mais fáceis de implementar com I/O bloqueante, e tendem a preferir o estilo TCP.

Associações que façam uso de confiabilidade parcial (como por exemplo multimídia em tempo real) utilizarão mais provavelmente o estilo UDP. O I/O não-bloqueante é adequado para servidores de conteúdo multimídia.

Referências Bibliográficas

AGGARWAL, Avnish et al. **RFC 1001** – Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and methods. IETF Network Working Group, 1987.

AGGARWAL, Avnish et al. **RFC 1002** – Protocol standard for a NetBIOS service on a TCP/UDP transport: Detailed specifications. IETF Network Working Group, 1987.

ALBITZ, Paul; LIU, Cricket. **DNS and BIND.** 2.ed. O'Reilly, 1997.

ALLMAN, M.; PAXSON, V.; STEVENS, W. **TCP Congestion Control.** IETF Network Working Group, 1999.

ARIAS-RODRIGUEZ, Iván. **Stream Control Transmission Protocol – The design of a new reliable transport protocol for IP networks.** Helsinki University of Technology, Electrical and Communications Engineering Department. Networking Laboratory 2002/02/12.

BELLOVIN, S. M.; IOANNIDIS, J.; KEROMYTIS, A. D. et al. **On the use of SCTP with IPSec** (draft-ietf-ipsec-sctp-06.txt). IETF Network Working Group, 2003.

BERNSTEIN, Dan J. **SYN Cookies.** http://cr.yp.to/. 1997.

CAMARILLO, Gonzalo. **SIP Demystified.** McGraw-Hill, 2002.

COENE, L. **RFC 3257** – Stream Control Transmission Protocol Applicability Statement. IETF Network Working Group, 2002.

CUSTÓDIO, Felipe. **Segurança em Computação.** UFSC/PPGCC, 2002.

DANTAS, Mário. **Tecnologia de rede de comunicação e computadores**. Axcel Books, 2002.

ECKSTEIN, Robert; COLLIER-BROWN, David; KELLY, Peter. **Using SAMBA.** O'Reilly, 2000.

FIELDING, R.; GETTYS, J.; MOGUL, J. et al. **RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1.** IETF Network Working Group, 1999.

FINLAYSON, R. **RFC 3119 – A More Loss-Tolerant RTP Payload Format for MP3 Audio**. IETF Network Working Group, 2001.

FLOYD, Sally; HANDLEY, Mark; KOHLER, Eddie. **Problem Statement for DCCP.** IETF Network Working Group, 2002.

GEORGE, Tom; BIDULOCK, Brian; DANTU, Ram et al. **SS7 MTP2-User Peer-to-Peer Adaptation Layer.** IETF Network Working Group, 2003.

HOFFMAN, D.; FERNANDO, G.; GOYAL, V. et al. **RFC 2250 – RTP Payload Format for MPEG1/MPEG2 Video.** IETF Network Working Group, 1998.

JUNGMAIER, A.; RATHGEB, E. P.; SCHOPP, Michael et al. SCTP – A Multi-link End-to-end Protocol for IP-based Networks. **AEÜ – International Journal of Electronics and Communications.** 55 (2001) No. 1, 46-54, 2001.

JUNGMAIER, A.; RESCORLA, E.; TUEXEN, M. **RFC 3436** – Transport Layer Security over Stream Control Transmission Protocol. IETF Network Working Group, 2002.

KAME: *http://www.kame.net*. Sítio da implementação do SCTP e do IPSEC no FreeBSD.

KANG, Sukwoo; FIELDS, Mattew. **Experimental Study of the SCTP compared to TCP**. Texas A&M University. Computer Communications and Network Fall, 2003.

KARN, P.; SIMPSON, W. **RFC 2522** – Photuris: Session-Key Management Protocol. IETF Network Working Group, 1999.

KRAWCZYK, H.; BELLARE, M.; CANETTI, R. **RFC 2104 –** HMAC: Keyed-Hashing for Message Authentication. IETF Network Working Group, 1999.

LEIGHTON, Luke Kenneth Casson. **DCE/RPC over SMB. Samba and Windows NT Domain Internals.** MTP, 2002.

LEWIS, Bil; BERG, Daniel J. **Multithreaded Programming with Pthreads.** Sun Microsystems Press, 1998.

LKSCTP: *http://lksctp.sourceforce.net*. Sítio do LK-SCTP *(Kernel-Level SCTP)*, a implementação do SCTP no Linux.

MAZZOLA, Vitório Bruno. **Arquitetura de Redes de Computadores**. UFSC/PPGCC, 2002.

MENTAT. *http://www.mentat.com/xtp/xtp-overview.html*. Site da empresa Mentat sobre o protocolo XTP.

MORNEAULT, K.; RENGASAMI, S.; KALLA, M. et al. **RFC 3057** – ISDN Q.921-User Adaptation Layer. IETF Network Working Group, 2001.

MORNEAULT, K.; DANTU, R.; SIDEBOTTOM, G. et al. **RFC 3331** – Signaling System 7 (SS7) Message Transfer Part 2 (MTP2) – User Adaptation Layer. IETF Network Working Group, 2002.

NAGLE, John. **RFC 896** – Congestion Control in IP/TCP Internetworks. IETF Network Working Group, 1984.

ONG, L.; RYTINA, I.; GARCIA, M. et al. **RFC 2719** – Framework Architecture for Signaling Transport. IETF Network Working Group, 1999.

ONG, L.; YOAKUM, J. **RFC3286** – An Introduction to the Stream Control Transmission Protocol (SCTP). IETF Network Working Group, 2002.

POSTEL, J.; REYNOLDS, J. **RFC 959 – File Transfer Protocol (FTP)**. IETF Network Working Group, 1985;

RAJAMANI, Rajesh; KUMAR, Sumit; NIKHIL, Gupta. **SCTP versus TCP: Comparing the performance of transport protocols for Web traffic.** Computer Sciences Department, University of Wisconsin-Madison, 2002.

RAMAKRISHNAN, K.; FLOYD, S.; BLACK, D. **RFC 3168** – The Addition of Explicit Congestion Notification (ECN) to IP. IETF Network Working Group, 2001.

RIVUS: *http://rivus.sourceforge.net*. Sítio da implementação do SCTP em sistema operacional BSD, com extensão de *"Load Sharing"*.

RUSSELL, Travis. **Signaling System #7.** 4. ed. McGraw-Hill, 2002.

SCHULZRINNE, H.; CASNER, S.; FREDERICK, R. et al. **RFC 1889 – RTP: A Transport Protocol for Real-Time Applications**. IETF Network Working Group, 1996.

SCTP.ORG: *http://www.sctp.org*. Sítio dos principais autores do SCTP.

SHANNON, Claude. **A Mathematical Theory of Communication.** The Bell System Technical Journal, Vol. 27, pag. 379-423, 623-656, Julho/Outubro de 1948.

SHREMMINGER: *http://developer.osdl.org/shremminger/netem/example.html*. *Emulating wide area network delays.* Sítio da ferramenta de simulação de rede *Netem*.

SIDEBOTTOM, G.; MORNEAULT, K.; PASTOR-BALBAS, J. **RFC 3332** – Signaling System 7 (SS7) Message Transfer Part 3 (MTP3) User Adaptation Layer (M3UA). IETF Network Working Group, 2002.

STEVENS, Richard W. **RFC 2001** – TCP Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery Algorithms. IETF Network Working Group, 1997.

STEVENS, Richard W. **TCP/IP Illustrated Volume 1.** The Protocols. Addison-Wesley, 1994.

STEVENS, Richard W.; Fenner, Bill; Rudoff, Andrew M. **Unix Network Programming Volume 1.** Networking APIs: Sockets and XTI. 3.ed. Prentice-Hall, 2004.

STEWART, R.; XIE, Q.; MORNEAULT, K. et al. **RFC 2960** – Stream Control Transmission Protocol. IETF Network Working Group, 2000.

STEWART, Randall R.; XIE, Qiaobing. **Stream Control Transmission Protocol (SCTP).** Addison-Wesley, 2002.

STEWART, R.; RAMALHO, M.; XIE, Q. et al. **SCTP Partial Reliability Extension** (draft-stewart-tsvwg-prsctp-03.txt). IETF Network Working Group, 2003.

STEWART, R.; RAMALHO, M.; XIE, Q. et al. **Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration**. IETF Network Working Group, 2003.

STEWART, R., ONG, L., ARIAS-RODRIGUEZ, I. et al. **Stream Control Transmission Protocol (SCTP) Implementer's Guide**. IETF Network Working Group, 2003.

STEWART, R.; XIE, Q.; YARROLL, L. et al. **Sockets API Extensions for Stream Control Transmission Protocol – SCTP**. IETF Network Working Group, 2004.

STEWART, R; RAMALHO, M. et al. **RFC 3758 – Stream Control Transmission Protocol (SCTP) Partial Reliability Extension.** IETF Network Working Group, 2004.

STONE, J.; STEWART, R.; OTIS, D. **RFC 3309 – Stream Control Transmission Protocol (SCTP) Checksum Change.** IETF Network Working Group, 2002.

TANENBAUM, Andrew S. **Redes de computadores**. 4.ed. Rio de Janeiro: Campus, 2002.

USAGI: *http://www.linux-ipv6.org*. Sítio do projeto USAGI, que implementa o IPv6 no Linux, coligado com o projeto KAME.

WILLRICH, Roberto. **Sistemas multimídia distribuídos.** UFSC/PPGCC, 2002.

XIE, Q.; STEWART, R.; SHARP, C. et al. **SCTP Unreliable Data Mode Extension** (draft-ietf-tsvwg-usctp-00.txt). IETF Network Working Group, 2001.

Este trabalho foi editado no LyX e tipografado pelo LaTeX .