UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE TELEMÁTICA

Propostas de implementação de qualidade de serviço na arquitetura VPN MPLS, utilizando linguagem de especificação formal SDL orientada a objetos e análise de desempenho utilizando o simulador OPNET.

Autor:

Marcel Cavalcanti de Castro

Orientador:

Prof. Dr. Walter da Cunha Borelli

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.

Banca Examinadora:

Prof. Dr. Walter da Cunha Borelli (Presidente)	FEEC/UNICAMP
Prof. Dr. Paulo Cardieri	FEEC/UNICAMP
Prof. Dr. Rodrigo Pinto Lemos	Universidade Federal de Goiás

Campinas, Dezembro de 2004.

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

C279p

Castro, Marcel Cavalcanti de
Propostas de implementação de qualidade de serviço na arquitetura VPN MPLS, utilizando linguagem de especificação formal SDL orientada a objetos e análise de desempenho utilizando o simulador OPNET / Marcel Cavalcanti de Castro. --Campinas, SP: [s.n.], 2004.

Orientador: Walter da Cunha Borelli
Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. SDL (Linguagem de programação de computador). 2. Interconexão de redes (Telecomunicações). 3. Redes de computação - Protocolos. 4. Redes de computação – Protocolos. 5. Simulação (Computadores). 6. Software – Validação. 7. Desempenho. I. Borelli, Walter da Cunha. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Resumo

Este trabalho apresenta propostas de implementação de qualidade de serviço na arquitetura *VPN MPLS* e análise de desempenho destas propostas. São desenvolvidos sistemas com base na arquitetura *VPN MPLS* e sugerido uma proposta de expansão da arquitetura *VPN MPLS* para mapeamento dinâmico das prioridades dos clientes *VPN* na rede do provedor de serviço através da inserção do valor de prioridade de rota na tabela *vrf* e modificações realizadas no protocolo *MP-BGP*. As propostas foram especificadas utilizando a linguagem de especificação formal *SDL* orientada à objetos a partir da ferramenta *SDL TAU Suite*. Esta ferramenta permite simular os sistemas especificados, a partir de diagramas *MSC*, e validar estes sistemas para detecção e correção dos erros de lógica e de especificação. A análise de desempenho das propostas foi realizada com o uso do simulador *Opnet Modeler*.

Palavras Chave: VPN, MPLS, MP-BGP, Qualidade de Serviço, Sinalização, Provedor de Serviço, SDL - *Specification and Description Language*, SDT - *SDL TAU Suite*, MSC - *Message Sequence Chart*, Simulação, Validação, Análise de Desempenho, Opnet Modeler.

Abstract

This work describe proposals for the implementation of quality of service (QoS) in VPN MPLS architecture, and their performance analysis based on simulations are presented. New systems are developed based on the VPN MPLS architecture, and it is being proposed an extension for the VPN MPLS architecture to construct a dynamic mapping of VPN clients priorities into service provider network through the insertion of priority field at *vrf* table and MP-BGP protocol modification. All these systems were specified using the SDL object-oriented formal language with the SDL TAU Suite tool (Telelogic, Sweden). This tool allows the simulation with MSC diagrams of the specified systems, and the validation of these systems by detecting and correcting logical and specification errors. The performance analysis of these proposals were realized through the Opnet Modeler simulator.

Keywords: VPN, MPLS, MP-BGP, Quality of Service, Signaling, Service Provider, SDL - Specification and Description Language, SDT - SDL TAU Suite, MSC - Message Sequence Chart, Simulation, Validation, Performance Analysis, Opnet Modeler.

Publicações no Período

M. C. Castro, M. A. Siqueira, M. F. Magalhaes, L. Farias. A BGP/MPLS PPVPN Management Information Model and a J2EE-based Implementation Architecture for Policy and Web-based Configuration Management Systems. In: *IEEE 3rd International Conference on Networking - ICN'04*, Guadeloupe - French Caribbean, Fevereiro 2004.

M. C. Castro, N. A. Nassif, W. C. Borelli. QoS Performance Evaluation in BGP/MPLS VPN. In: *International Information and Telecommunication Technologies Symposium - I2TS'2003*, Florianópolis - SC, Novembro 2003.

M. C. Castro, T. L. Resende, T. B. Pereira. Improving NGN with QoS Strategies. In: *Opnetwork 2003*, Washington - EUA, Agosto 2003.

M. C. Castro, E. T. Nakamura, M. A. Siqueira. Análise dos Aspectos de Segurança das VPNs MPLS. In: *Simpósio Brasileiro de Redes de Computadores - SBRC 2003*, Natal - RN, Maio 2003.

Siglas e Acrônimos

afi - *Address Family Identifier*

atm - *Asynchronous Transfer Mode*

asn - *Autonomous System Number*

bgp - *Border Gateway Protocol*

case - *Computer Aided Software Engineering*

ce - *Customer Edge*

cpqd - *Centro de Pesquisa e Desenvolvimento em Telecomunicações*

diffserv - *Differentiated Service*

dscp - *Diff-Serv Code Point*

ebgp - *Exterior BGP*

exp - *Experimental Use*

e-lsp - *EXP-inferred-PSC LSP*

fec - *Forwarding Equivalent Class*

fifo - *First In First Out*

fr - *Frame Relay*

ftp - *File Transport Protocol*

gui - *Graphical User Interface*

ietf - *Internet Engineering Task Force*

ip - *Internet Protocol*

ipv6 - *Internet Protocol version 6*

ipx - *Internetwork Packet Exchange*

ipsec - *IP Security Protocol*

itu - *International Telecommunications Union*

ler - *Label Edge Router*

lsp - *Label Switching Path*

lsr - *Label Switching Router*

l2vpn - *Layer 2 VPN*

l3vpn - *Layer 3 VPN*

l-lsp - *Label-only-inferred-PSC LSP*

mp-bgp - *Multiprotocol BGP*

mpls - *Multiprotocol Label Switching*

msc - *Message Sequence Chart*

nlri - *Network Layer Reachability Identifier*

ngn - *Next Generation Networks*

ospf - *Open Shortest Path*

pe - *Provider Edge*

phb - *Per Hop Behavior*

pid - *Process Identifier*

ppvpn - *Provider Provisioned VPN*

psc - *PHB Scheduling Class*

qos - *Qualidade de Serviço*

rd - *Route Distinguisher*

rip - *Routing Information Protocol*

safi - *Subsequent Address Family Identifier*

sdl - *Specification and Description Language*

sdt - *SDL TAU Suite*

sla - *Service Level Agreement*

spf - *Shortest Path First*

tcp - *Transport Control Protocol*

tos - *Type of Service*

vpn - *Virtual Private Network*

vrf - *Virtual Routing and Forwarding*

voip - *Voice over IP*

wfq - *Weight Fair Queue*

rede privada virtual - É uma rede privada que utiliza uma rede pública como *backbone*, e que é acessada somente pelos usuários ou grupos devidamente registrados nesta VPN.

vpn gerenciada pelo cliente - *vpn* onde a criação, manutenção e gerenciamento da rede esta a cargo do próprio cliente.

vpn gerenciada pelo provedor - *vpn* onde a criação, manutenção e gerenciamento da rede do cliente esta a cargo do provedor de serviço.

site vpn - um conjunto de sub-redes que fazem parte da rede do cliente *vpn*, mas que não estão localizadas no mesmo local geográfico.

par mp-bgp - roteadores que implementam o protocolo *mp-bgp*, e estabelecem uma relação de "vizinhança", na forma de pares.

roteador ce - roteador de borda do cliente *vpn*. Este roteador não implementa a tecnologia *mpls*, e representa o ponto da rede em que os pacotes entram e saem da rede do cliente *vpn*, atravessando a rede *mpls* do provedor.

roteador pe - roteador de borda do provedor de serviço usado para conectar os *sites vpn* dos clientes. Esta localizado na borda da rede do provedor, e deve suportar a tecnologia *mpls*.

roteador p - roteador de núcleo do provedor de serviço. Este roteador não necessita manter informações de clientes *vpn*, sendo responsável apenas pela comutação rápida de pacotes na rede, e deve suportar a tecnologia *mpls*.

tabela vrf - tabela de roteamento e encaminhamento independente criada para cada *site vpn* que esta conectado ao roteador *pe*. Esta tabela é usada pelo provedor de serviços, para distinguir as informações dos clientes *vpn* nos roteadores *pe*.

família de endereço vpn-ipv4 - formato de endereço criado para tornar único os endereços dos clientes *vpn*. O endereço *vpn-ipv4* é composto pelo campo para distinção de rotas (*rd - route distinguisher*) seguido do endereço *ipv4* do cliente *vpn*.

sistema autônomo - rede que esta sobre o controle de uma mesma entidade administrativa.

sessão mp-ibgp - sessão *mp-bgp* do tipo interna. Representa uma sessão *mp-bgp* estabelecida entre roteadores pertencentes ao mesmo sistema autônomo, ou seja, sessão entre os roteadores *pe* da rede do provedor.

sessão mp-ebgp - sessão *mp-bgp* do tipo externa. Representa uma sessão *mp-bgp* estabelecida entre roteadores pertencentes à sistemas autônomos diferentes, ou seja, sessão entre os roteadores *pe* e *ce*.

Sumário

Lista de Figuras

1

Introdução

1.1 Motivação e Objetivos

Corporações e empresas estão se tornando cada vez mais dependentes de suas redes para comunicação de dados e serviços de telecomunicações. Atualmente a necessidade de interconectar redes corporativas em lugares geograficamente distribuídos tornou-se cada vez mais importante.

A conectividade de redes corporativas vem sendo realizada pelos provedores de serviço, principalmente através de conexões *fr* (*frame relay*) ou *atm* (*asynchronous transfer mode*), e recentemente através de *ethernet* e túneis *IP* (*internet protocol*). Este tipo de rede, que interconecta vários *sites* (redes corporativas) através de uma infraestrutura de rede compartilhada é chamada de *vpn* (*Virtual Private Network*) ou *rede privada virtual* [19].

Atualmente os provedores de serviço estão a procura de maneiras mais eficientes de ofer-

ecer serviços *vpn* a seus clientes, combinando diferentes requisitos, como: otimização no uso do *backbone* da rede, automação na criação e gerenciamento de *vpn* e habilidade em oferecer serviços diferenciados sobre a mesma infra-estrutura. Fabricantes começaram a propor implementações proprietárias diferentes a fim de sanar estas necessidades, basicamente com o uso do *mpls* (*multiprotocol label switching*) como tecnologia de *backbone* convergente.

Com a iniciativa de alguns provedores de serviço, as organizações *itu-t* (*international telecommunications union - telecommunication standardization sector*) e *ietf* (*internet engineering task force*) reconheceram a importância de se iniciar um trabalho de padronização da tecnologia *vpn*. O grupo de estudo 13 do *itu-t* (*itu-t study group 13*) iniciou o processo de definição de requisitos para *vpn* [22] e a classificação das propostas técnicas de *vpn* para serviços de nível 3 sobre tecnologia *mpls* [21] em Maio de 2000. Alguns meses depois, o *ietf* iniciou a discussão que levou a criação do grupo de trabalho *ppvpn* (*provider provisioned VPN*)[34] [5] no início de 2001, incumbido de padronizar as propostas de *vpn* para serviços de nível 2 e 3.

O termo *rede privada virtual* é muito abrangente e tem significados diferentes para diferentes pessoas. Baseado na classificação usada pelos órgãos padronizadores citados, as propostas de *vpn* são classificadas de acordo com a responsabilidade de gerenciamento; *vpn* gerenciada pelo cliente (*customer edge - ce based vpn*) ou *vpn* gerenciada pelo provedor (*provider edge - pe based vpn*). Este trabalho é focado nas *vpns* gerenciadas pelo provedor, também conhecidas como *network based vpn*.

Em 2003 o grupo de trabalho *ppvpn* do *ietf* foi dividido em dois grupos de trabalho, o *l3vpn* (*layer 3 vpn*) responsável por padronizar as propostas de *vpn* para serviços de nível 3 e o *l2vpn* (*layer 2 vpn*) responsável por padronizar as propostas de *vpn* para serviços de nível 2. A partir do grupo *l3vpn*, três propostas surgiram como padronização para *vpn*. São elas; *vpn ce-based ipsec* [24], *vpn virtual router ip* [32] e *vpn bgp/mpls ip* [12].

Este trabalho está baseado na arquitetura *vpn bgp/mpls ip*, também denominada arquitetura *vpn-mpls*, onde o provedor de serviço usa o protocolo *mp-bgp* (*multiprotocol border gateway protocol*) padronizado pelo *ietf* [37] no *backbone* da rede. O protocolo *mp-bgp* troca informações de roteamento de cada *vpn* entre todos os *sites vpn* pertencentes à aquela *vpn*. A arquitetura também faz uso dos rótulos *mpls* para identificar e separar o tráfego de diferentes *vpns*.

Comparadas as estratégias de *vpn* nível 2 (exemplo: *vpn* com uso da tecnologia *frame relay* ou *atm*), as *vpns* nível 3 herdam a flexibilidade e a simplicidade das redes *IP*, refletindo-se em uma arquitetura escalável [25], devido ao grande potencial de crescimento das redes *IP*. Deve-se ressaltar que este não é o tipo ideal de rede com suporte a qualidade de serviço [26], devido à característica implícita de melhor esforço (*best effort*) das redes *IP*.

Segundo o IETF [12], a implementação de qualidade de serviço na arquitetura *vpn-mpls* pode ocorrer com o uso da arquitetura de serviços diferenciados (arquitetura *diffserv* [36]) na rede do cliente *vpn*, combinado ao uso do *mpls* no *backbone* do provedor. Através desta combinação, as aplicações dos clientes *vpn* são tratadas na borda da rede do provedor de serviço pelos mapeamentos de prioridades da arquitetura *diffserv* no cabeçalho *mpls* [14], de acordo com o nível de qualidade de serviço contratado pelo cliente. Alguns trabalhos como [27], [15], [17] e [26] discutem esta forma de provimento de qualidade de serviço para arquitetura *vpn-mpls*. É importante ressaltar que estes trabalhos levam em consideração o mapeamento entre as prioridades do cliente e os acordos de nível de serviço contratados, implementados de forma estática pelo provedor de serviço, ou seja, o cliente *vpn* não consegue estabelecer novos níveis de qualidade de serviços em tempo real.

Neste contexto, este trabalho desenvolve uma proposta em *SDL* de expansão da arquitetura *vpn-mpls*, para criação de um mapeamento dinâmico entre as prioridades dos clientes *vpn* e os acordos de nível de serviço contratados do provedor. Este mapeamento é implementado através da inserção de um novo parâmetro na tabela *vrf*, que representa a prioridade da rota do cliente *vpn*, e a modificação do protocolo *mp-bgp* para o transporte dos valores de prioridade de rota entre o cliente *vpn* e provedor de serviço. A expansão da arquitetura tem como objetivo a troca de sinalização entre o cliente *vpn* e o provedor de serviço, a fim de se estabelecer uma implementação de qualidade de serviço fim-a-fim, através do estabelecimento de novos níveis de qualidade de serviço em tempo real.

A linguagem *SDL* foi escolhida por ser uma linguagem de especificação formal, e por disponibilizar de uma ferramenta *case* (*computer aided software engineering*) como o *SDL TAU Suite* [1]

[1]Pacote Telelogic SDL TAU Suite 4.2 : adquirido pelo DT/FEEC/UNICAMP através do Projeto Temático - FAPESP (Proc. 91/3660-0)

que utiliza essa linguagem para construção de especificação formal de *software* de comunicação, permitindo a realização de simulações, validações e geração de códigos, criando dessa forma sistemas mais confiáveis. Os modelos *SDL* propostos conferem uma formalização dos sistemas, permitindo a validação de suas principais funcionalidades, bem como a simulação e análise dos diagramas *MSC* de casos críticos envolvidos nos sistemas, auxiliando a identificação e correção de quaisquer erros existentes antes de uma eventual implementação.

Este trabalho também desenvolve uma análise de desempenho do mapeamento de prioridades da arquitetura de serviços diferenciados na rede *mpls* do provedor de serviços, usado na proposta de expansão da arquitetura *vpn-mpls*. Esta análise de desempenho foi realizada em conjunto com o projeto de pesquisa *NGN* (*Next Generation Network*) desenvolvido pelo *CPqD* (Centro de Pesquisa e Desenvolvimento em Telecomunicações) sendo validada em ambiente de simulação (*Opnet Modeler* [2] [3]).

1.2 Organização da Dissertação

Este trabalho está dividido em seis capítulos e três apêndices.

O **Capítulo 2** descreve uma introdução à arquitetura *vpn-mpls*, considerando os pontos mais relevantes para o trabalho desenvolvido. Ao final do capítulo são apresentadas as propostas dos sistemas para o capítulo 3.

O **Capítulo 3** apresenta as propostas de especificação formal orientada a objetos da arquitetura *vpn-mpls* e da proposta de expansão da arquitetura *vpn-mpls*. No capítulo são apresentados diagramas em *SDL* dos sistemas desenvolvidos a partir dos conceitos da arquitetura apresentada no capítulo 2. É mostrado também a possibilidade de reuso das especificações através do conceito de herança. Desta forma, é possível simplificar a construção dos novos blocos e processos *SDL* a partir das especificações já existentes.

O **Capítulo 4** apresenta os resultados da validação dos sistemas desenvolvidos e exemplos de simulações dos comportamentos funcionais e casos críticos através de diagramas *MSC* (*Message Sequence Chart*). O capítulo apresenta também técnicas para identificar e corrigir erros

[2]Pacote Opnet Modeler v9.1 : adquirido pelo *CPqD* através do Projeto de Pesquisa Redes de Próxima Geração(Projeto *NGN*)

existentes de lógica e especificação destes sistemas através da validação e simulação, garantindo maior sucesso em futuras expansões e implementações.

O **Capítulo 5** apresenta os resultados de análise de desempenho do mapeamento de prioridades da arquitetura de serviços diferenciados na rede *mpls* do provedor de serviços, usado na proposta de expansão da arquitetura *vpn-mpls*. Estes resultados são obtidos com o uso da ferramenta de simulação *Opnet Modeler*.

No **Capítulo 6** são apresentadas as conclusões finais e sugestões para eventuais trabalhos futuros.

O **Apêndice A** apresenta uma introdução à linguagem *SDL* com as características principais e relevantes para o trabalho desenvolvido. O apêndice também apresenta a ferramenta *SDL TAU Suite* que possui os módulos *Simulator* e *Validator* utilizados para gerar resultados das simulações dos casos críticos e validação dos sistemas respectivamente, para o desenvolvimento das especificações dos sistemas em *SDL*.

O **Apêndice B** apresenta uma introdução ao simulador *Opnet Modeler* com as características principais e relevantes para o trabalho desenvolvido. O objetivo deste apêndice é mostrar os aspectos básicos da plataforma de simulação utilizada.

O **Apêndice C** apresenta os diagramas das especificações gráficas em *SDL*, referentes às propostas dos sistemas especificados da arquitetura *vpn-mpls* (sistema *basicarquitecture-scenario1* - Apêndice C.1) e da expansão da arquitetura *vpn-mpls* (sistema *basicarquitecture-scenario2* - Apêndice C.2). Por motivo de organização, e para facilitar o entendimento dos sistemas, este apêndice apresenta todos os diagramas usados nas especificações, inclusive os já apresentados no capítulo 3.

2

Arquitetura VPN MPLS e a proposta de expansão para provimento de qualidade de serviço

As *redes privadas virtuais* (*VPN - Virtual Private Network*) possibilitam a conectividade em níveis globais a custos relativamente baixos, permitindo assim que as comunicações das corporações sejam realizadas de modo a tornar possível a adoção de um modelo de negócio mais rápido e mais dinâmico. Além disso, a *VPN* possui também importantes aspectos econômicos, principalmente com relação à não necessidade de se manter uma infra-estrutura de comunicação própria.

Tradicionalmente, *VPNs* eram estabelecidas através do uso de circuitos dedicados, ou

através do uso de *softwares* de segurança estabelecendo-se túneis de comunicação entre as redes. Estas tecnologias não apresentavam soluções economicamente viáveis para empresas pequenas e ao mesmo tempo não eram escaláveis às grandes corporações. Em seu lugar, as empresas podem hoje utilizar o potencial dos serviços oferecidos pelos provedores de serviço, tendo vantagens na redução dos custos e experiência na manutenção dos serviços.

Baseando-se na necessidade dos provedores de serviço em oferecer serviços *VPN* a seus clientes de maneira eficiente, através do uso otimizado do *backbone*, da automação na criação e gerenciamento de *VPNs* e da habilidade em oferecer serviços diferenciados sobre a mesma infra-estrutura, o IETF iniciou o trabalho de padronização da arquitetura *vpn-mpls*[12].

O oferecimento de serviços *VPN* com qualidade (QoS) é um aspecto importante a ser levado em consideração pela arquitetura *vpn-mpls*(seção 2.1). De acordo com [12] a implementação de qualidade de serviço nesta arquitetura pode ocorrer com a combinação da arquitetura de *serviços diferenciados* junto à arquitetura *mpls*, através do uso de mapeamentos de prioridades. Este mapeamento sugerido pelo IETF, é um mapeamento estático, onde o cliente ao contratar o serviço *VPN* só poderá alterar este mapeamento via provedor de serviço. Neste contexto, este trabalho desenvolve uma proposta de expansão da arquitetura *vpn-mpls*(seção 2.2) para mapeamento dinâmico entre as prioridades dos clientes *vpn* e os acordos de nível de serviço contratados do provedor.

Para a compreensão de todos os sistemas propostos neste trabalho, este capítulo apresenta a arquitetura *vpn-mpls* e os principais conceitos relacionados, bem como uma introdução a linguagem *SDL* utilizada para especificar os sistemas.

A seção 2.1 descreve a arquitetura *vpn-mpls* com os respectivos componentes, sinalizações e funcionalidades. A seção 2.2 descreve a proposta de expansão da arquitetura *vpn-mpls* e uma breve descrição das características da linguagem *SDL*.

2.1 Arquitetura VPN MPLS

A arquitetura *vpn-mpls*, também denominada arquitetura *vpn bgp/mpls ip*, define mecanismos que permitem o provedor de serviço utilizar seu *backbone IP* para prover serviços *VPN* a seus

clientes. Nesta arquitetura, o protocolo *mp-bgp* (*multiprotocol bgp*) [37] é usado para distribuir informações de roteamento dos clientes *vpn*, e o *mpls* para envio do tráfego dos clientes através do *backbone* do provedor de serviço.

Nesta arquitetura, o IETF padronizou através da RFC 2858 [37] o *mp-bgp* como uma extensão do *bgp* [41] para transportar informações de roteamento de múltiplos protocolos de rede, como; IPv6 ou IPX.

- **Componentes da arquitetura *vpn-mpls***

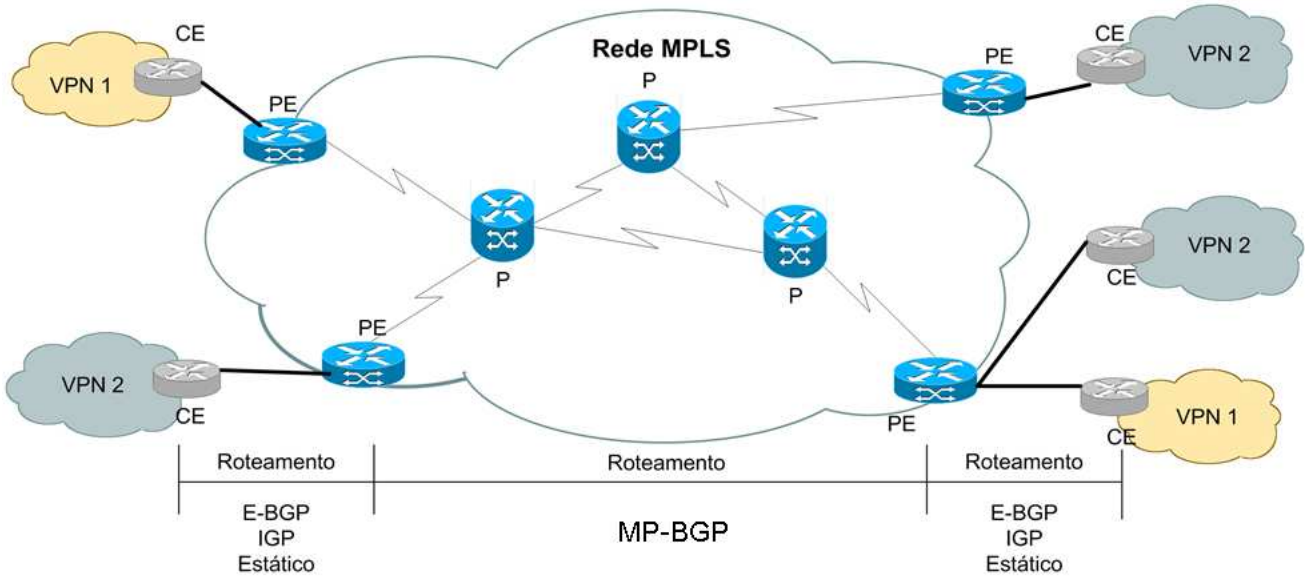


Figura 2.1: Componentes da arquitetura *vpn-mpls* [31]

A figura 2.1 apresenta os componentes de uma arquitetura *vpn-mpls*. Nesta figura, o equipamento de borda do cliente, também conhecido como roteador *ce* (*customer edge*) proporciona aos clientes acesso à rede do provedor de serviço. Através da conexão entre os roteadores *ce* e o provedor de serviço, o cliente *vpn* divulga suas informações de rotas à toda rede, e através desta conexão o roteador *ce* é informado de novas rotas adicionadas à rede por outros *sites vpn*. O roteador *ce* não implementa a tecnologia *mpls*, e representa o ponto da rede em que os pacotes entram e saem da rede do cliente *vpn*, atravessando a rede *mpls* do provedor.

O equipamento de borda do provedor, também conhecido como roteador *pe* (*provider edge*), troca informações de roteamento com o cliente *vpn* conectado ao roteador *ce* e com a rede do

provedor de serviço. A troca de informações com o roteador *ce* (cliente *vpn*), ocorre através de roteamento estático ou com o uso de protocolos de roteamento, como; *rip* (*routing information protocol*), *ospf* (*open shortest path*) ou *bgp*(*exterior border gateway protocol*). Já a troca de informações com a rede do provedor de serviços ocorre através do uso de dois protocolos de roteamento, o primeiro para troca de informações de rota da rede do provedor, como: *rip* ou *ospf*, e o segundo para troca de informações de roteamento dos clientes *vpn*, através do protocolo *mp-bgp*.

Na arquitetura *vpn-mpls*, o roteador *pe* mantém somente as informações de roteamento dos clientes *vpn* diretamente conectados a ele, com isso há um aumento da escalabilidade da rede, pois elimina a necessidade dos roteadores *pe* manterem todas as rotas *vpn* informadas ao provedor de serviço [25].

Cada roteador *pe* mantém uma tabela de roteamento independente para cada cliente *vpn* diretamente conectado, denominada tabela *vrf*(*virtual routing and forwarding*). Após a instalação das rotas locais do cliente *vpn* na tabela *vrf*, o roteador *pe* troca estas informações de roteamento com outros roteadores *pe* que possuem *sites* pertencentes à mesma *VPN*.

Na figura 2.1, os equipamentos de núcleo do provedor, denominados roteadores *p* (*provider*), são roteadores que não estão ligados diretamente aos roteadores *ce*. Roteadores *p* exercem a função de roteadores comutadores de rótulos *mpls*, (*lsr - label switching routers*), comutando os tráfegos de dados dos clientes *vpn* entre os roteadores *pe*. Como o tráfego é enviado através do *backbone mpls* usando hierarquia de rótulos, os roteadores *p* somente necessitam manter rotas para os roteadores *pe*, não necessitando manter informações de roteamento *vpn*, aumentando assim a escalabilidade no *backbone mpls* [25].

- **Sobreposição do Espaço de Endereçamento do Cliente**

Redes corporativas usam espaço de endereçamento privado[40], sendo que um mesmo espaço de endereçamento pode ser usado por diferentes clientes *vpn* para identificar redes diferentes. Havendo a necessidade de se interligar estas redes corporativas através do *backbone* do provedor de serviço, se torna difícil para o protocolo de roteamento do *backbone* assumir que cada endereço *ipv4* (*internet protocol version 4*) que ele transporta é único na rede.

Para eliminar o problema do uso do mesmo espaço de endereçamento privado em redes diferentes, a arquitetura *vpn-mpls* criou um mecanismo que possibilita o protocolo de roteamento *mp-bgp* distinguir os prefixos destas redes, de tal forma que possa instalar rotas completamente diferentes para cada cliente *vpn*, mesmo que mais de um cliente faça uso do mesmo espaço de endereçamento privado. O mecanismo criado pela arquitetura *vpn-mpls* para suportar esta capacidade é denominado *família de endereço vpn-ipv4*, que representa a adição do campo para distinção de rotas (*rd - route distinguisher*) ao endereço *ipv4* dos clientes *vpn*, criando assim endereços *vpn-ipv4* únicos na rede.

É importante ressaltar que o uso do endereço *vpn-ipv4* se restringe ao *backbone* do provedor de serviço, ou seja, os clientes *vpn* não necessitam saber do uso de endereços *vpn-ipv4*. Os endereços *vpn-ipv4* são transportados somente durante a sinalização de roteamento dos clientes *vpn*, não sendo necessário o transporte deste endereçamento no cabeçalho dos pacotes de tráfego de dados do cliente ao passar pela rede do provedor de serviço.

- **Tabelas de roteamento independentes**

Na arquitetura *vpn-mpls*, o isolamento do tráfego dos clientes *vpn* é realizado através do uso de tabelas de roteamento independentes, também conhecidas como tabelas *vrf*. Nesta arquitetura, cada roteador *pe* mantém uma ou mais tabelas *vrf*, dependendo da quantidade de clientes *vpn* conectados a ele. Cada tabela *vrf* é associada a uma ou mais portas (interfaces ou sub-interfaces) ligadas aos roteadores *ce*, sendo criadas tabelas de roteamento individuais para cada *site vpn* [29].

A figura 2.2 apresenta um exemplo de criação de tabelas *vrf* em uma rede baseada na arquitetura *vpn-mpls*. Existe uma tabela *vrf* para cada *site vpn* conectado ao roteador *pe*, cujas informações básicas são: interface ligada ao cliente, campo de distinção de rotas, atributos de exportação e importação de rotas, e o rótulo *mpls*, garantindo assim a diferenciação entre os tráfegos dos clientes *vpn*. Neste exemplo, os clientes da *vpn 1* estão ligados à *vrf* da *vpn 1* localizada nos roteadores *pe-1* e *pe-2*. Os clientes da *vpn 2* estão ligados à *vrf* da *vpn 2* também localizada nos roteadores *pe-1* e *pe-2*. De acordo com a figura 2.2, os roteadores *pe-1*, *pe-2* e *pe-3* trocam informações de rotas dos clientes através do protocolo *mp-bgp*. Neste cenário

diferentes *sites vpn* conectados ao mesmo roteador *pe* podem usar a sobreposição do espaço de endereçamento, ou seja, clientes distintos podem adotar a mesma faixa de endereçamento privado sem causar distúrbios para o roteamento no *backbone* do provedor de serviço.

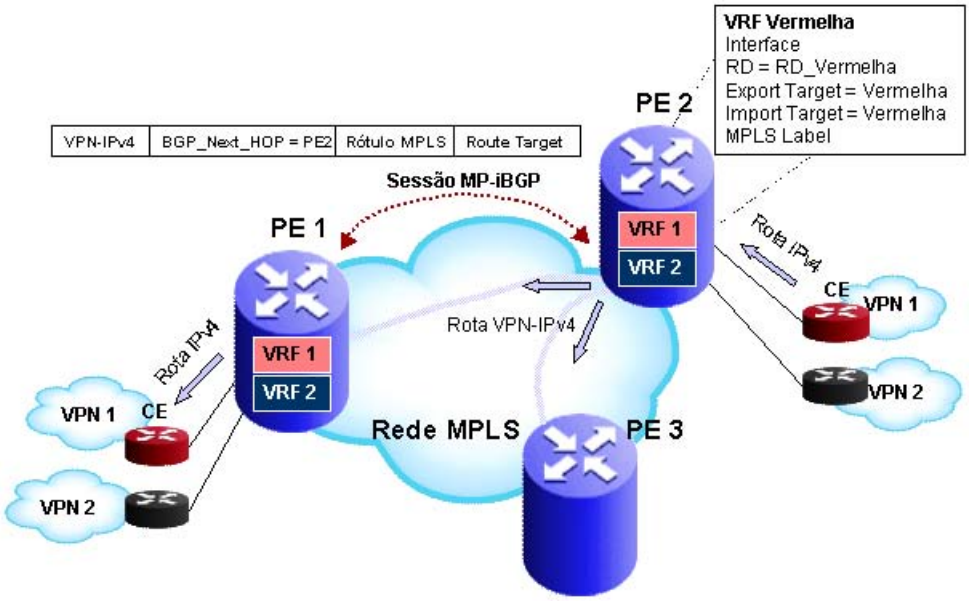


Figura 2.2: Criação de tabelas de tabelas *vrf*

- **Distribuição de rotas dos clientes *vpn* entre os roteadores *pe***

Roteadores *pe* distribuem rotas *vpn-ipv4* via protocolo *mp-bgp*, e fazem uso de seu endereço como informação de próximo *hop* da rota (campo *next hop* do protocolo *mp-bgp*). O endereço do roteador *pe* é codificado como um endereço *vpn-ipv4*, com valor de *rd* igual a zero devido a necessidade do *mp-bgp* em transportar a informação de próximo *hop* da rota pertencendo a mesma família de endereço *vpn-ipv4*. Junto a informação de rota, o roteador *pe* distribui um rótulo *mpls* que identifica a tabela *vrf*. Quando um roteador *pe* processa um pacote recebido com a informação do rótulo *mpls*, o roteador verifica a qual tabela *vrf* pertence o rótulo *mpls* e envia o pacote ao roteador *ce* conectado a esta tabela *vrf*.

A troca de informações de rotas dos clientes *vpn*, é feita através da troca de mensagens *update* entre roteadores *pe* do *backbone* do provedor de serviço, que transporta a informação de rota pertencente à família de endereço *vpn-ipv4* junto com o valor do rótulo *mpls*.

Máquina de estado do protocolo *mp-bgp*

A máquina de estado do protocolo *mp-bgp* proposta por [37] de forma textual, é apresentada na figura 2.3. De acordo com esta figura, a máquina de estado é composta por eventos e estados.

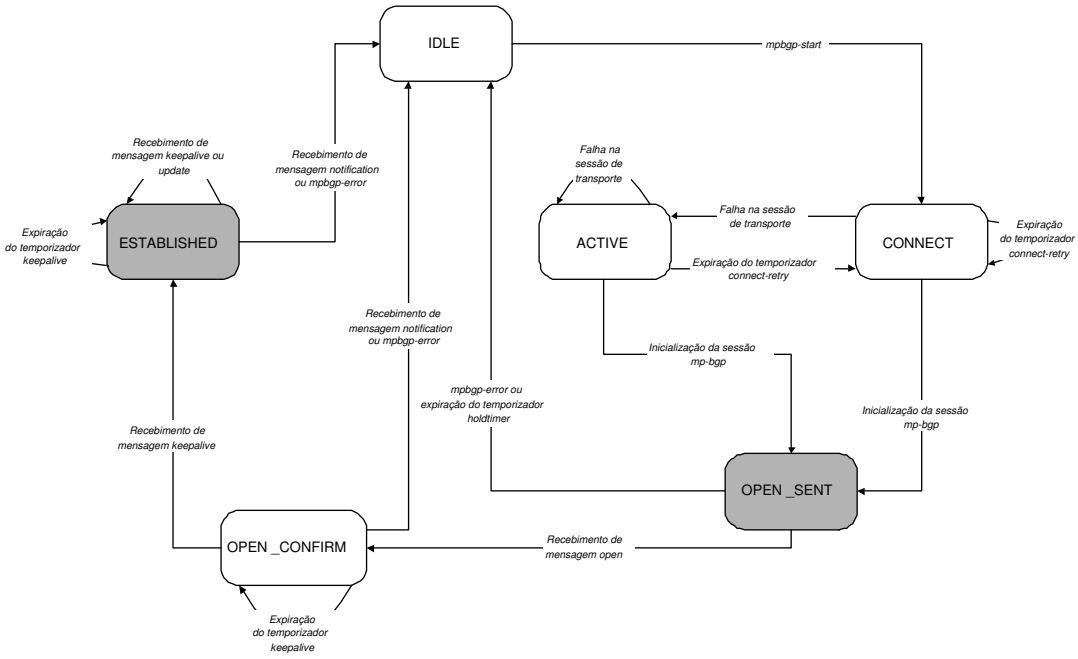


Figura 2.3: Máquina de estado do protocolo *mp-bgp*

Eventos:

- ***open***; utilizada pelo protocolo *mp-bgp* para o estabelecimento de conexão com o par *mp-bgp*.

- ***keepalive***; utilizada pelo protocolo *mp-bgp* para troca de informações periódicas com o objetivo de informar ao par *mp-bgp* que a conexão ainda está ativa.

- ***notification***; mensagem utilizada pelo protocolo para informar a ocorrência de erros na troca de informação ou execução da máquina de estado.

- ***update***; mensagem trocada pelo protocolo *mp-bgp* para informar ao par *mp-bgp* sobre atualizações de rotas ocorridas na tabela *vrf*.

Estados:

- ***Idle***: Estado inicial do protocolo *mp-bgp*. Neste estado o *mp-bgp* recusa todas as solicitações de conexão. Em resposta ao evento de inicialização (*mpbgp-start*) enviado pelo sistema, o protocolo inicializa seus recursos, inicia o temporizador *connect-retry* usado para delimitar o tempo de espera para o estabelecimento de uma sessão *mp-bgp*, e vai para o estado *connect*.

- ***Connect***: Neste estado o protocolo *mp-bgp* aguarda o estabelecimento da conexão do nível de transporte(conexão *tcp*). Se a conexão de transporte ocorrer com sucesso, o protocolo envia uma mensagem *open* para o par *mp-bgp* com o qual esta se estabelecendo a sessão, e o protocolo vai para o estado *open-sent*. Se a conexão de transporte falhar, o protocolo continua à espera de novas conexões, reinicializa o temporizador *connect-retry*, e muda seu estado para *active*. Se o temporizador expirar, o protocolo reinicializa o temporizador, inicia uma conexão de transporte com o par *mp-bgp*, e permanece no estado *connect*.

- ***Active***: Neste estado o protocolo *mp-bgp* tenta estabelecer uma conexão com seu par através da tentativa de inicialização da sessão *mp-bgp*. Se a conexão de transporte for estabelecida com sucesso, o protocolo envia uma mensagem *open* para o par *mp-bgp* com o qual esta se estabelecendo a sessão, e o protocolo vai para o estado *open-sent*. Se o temporizador *connect-retry* expirar, o protocolo reinicializa o temporizador, inicia uma conexão de transporte com o par *mp-bgp*, e muda para o estado *connect*.

- ***Open-sent***: Neste estado o protocolo *mp-bgp* aguarda por uma mensagem *open* do seu par e faz uma verificação de conformidade dos dados da mensagem. A verificação de conformidade leva em consideração a versão do protocolo *mp-bgp* suportada pelo roteador, o número do sistema autônomo do par *mp-bgp* que enviou a mensagem, e parâmetros de autenticação, se for utilizado a opção de autenticação das mensagens *mp-bgp*. Se a mensagem estiver em conformidade, o protocolo *mp-bgp* envia uma mensagem *keepalive*, inicializa o temporizador *hold-timer* usado para limitar o tempo máximo de espera entre

o recebimento de mensagens *keepalive* e/ou mensagens *update*, e vai para o estado *open-confirm*. Se o temporizador expirar, ou o sistema enviar uma notificação de erro, o protocolo vai para o estado *idle* e encerra a conexão *mp-bgp* e todos os recursos desta conexão.

- ***Open-confirm***: Neste estado o protocolo *mp-bgp* aguarda o recebimento de mensagens *keepalive* ou *notification*, para que o par possa trocar informações de roteamento. Se o protocolo receber uma mensagem *keepalive*, ele inicializa o temporizador *keepalive* usado para delimitar o tempo de espera para o recebimento da mensagem *keepalive* de seu par *mp-bgp*, e vai para o estado *established*. Se o temporizador *keepalive* expirar, o protocolo envia uma mensagem *keepalive* e reinicia o temporizador. O recebimento de uma mensagem *notification* representa a ocorrência de erro informado na mensagem, e o protocolo retorna ao estado *idle*.

- ***Established***: Neste estado o protocolo *mp-bgp* troca mensagens *update*, *keepalive* e *notification* com seus pares. Se ocorrer o recebimento de uma mensagem *update*, o protocolo processa as informações da mensagem e permanece no estado *established*. O mesmo ocorre com o recebimento das mensagens *keepalive*. Se o temporizador *keepalive* expirar o protocolo envia uma mensagem *keepalive* e reinicia o temporizador *keepalive* permanecendo no estado *established*. Caso o sistema envie uma notificação de erro, o protocolo vai para o estado *idle* e encerra a conexão *mp-bgp* e todos os recursos desta conexão.

2.1.1 Qualidade de serviço na arquitetura *vpn-mpls*

A provisão de qualidade de serviço (QoS) é parte intrínseca dos serviços emergentes, como a arquitetura *vpn-mpls*. De acordo com [12], uma solução para provimento de qualidade de serviço na arquitetura *vpn-mpls* é a implementação da combinação entre à arquitetura de serviços diferenciados (arquitetura *diffserv* [36]) e o *mpls* [13]. A arquitetura *diffserv* e o *mpls* podem ser vistos como tecnologias complementares na busca pela qualidade de serviço fim-a-fim na arquitetura *vpn-mpls*. Estas duas tecnologias possuem vários pontos em comum, realizando classificação de pacotes nos nós de ingresso da rede, e uso de agregação de tráfego em classes,

sendo o *mpls* com *fecs* (*forwarding equivalent classes*) e o *diffserv* com *phb* (*per hop behavior*) [30]. Rótulos de tamanho fixo são usados em ambas as arquiteturas, chamados: rótulo *mpls* para o *mpls* e *dscp* (*diffserv code point*) para o *diffserv*. Os roteadores *p* tratam os pacotes de acordo com seus rótulos, e o campo *dscp* mapeado no rótulo *mpls* determina o comportamento do nó com relação ao mecanismos de escalonamento e gerenciamento de filas. Este mapeamento de prioridades, sugerido pelo IETF, é um mapeamento estático, onde o cliente ao contratar o serviço *VPN* só poderá alterar este mapeamento via provedor de serviço. Com isto se define a prioridade e a ordem de descarte dos pacotes na rede.

A engenharia de tráfego, sugerida por [9], também é aplicável à arquitetura *vpn-mpls*. A engenharia de tráfego pode ser usada no estabelecimento de caminhos comutados por rótulos (*lsp - label switching path*) com características particulares de qualidade de serviço entre sites de clientes *vpn*, conforme é apresentado no capítulo 5, através da análise de desempenho do cenário de simulação com engenharia de tráfego.

2.2 Proposta de expansão da arquitetura *vpn-mpls*

Este trabalho propõe uma expansão da arquitetura *vpn-mpls* para provimento em tempo real de qualidade de serviço fim-a-fim. A proposta de expansão da arquitetura *vpn-mpls* introduzida neste trabalho de tese é baseada na possibilidade de estabelecimento de níveis de qualidade de serviços em tempo real, ou seja, estabelecimento de mapeamento dinâmico de prioridades (mapeamento da arquitetura de *serviços diferenciados* no *mpls*) de acordo com a necessidade do cliente *vpn*.

A criação deste mapeamento dinâmico de prioridades entre os clientes *vpn* (roteador *ce*) e o provedor de serviço (roteador *pe*) é realizado através da inserção de parâmetros de prioridade na tabela *vrf* implementada no roteador *pe*, representando as prioridades das rotas dos clientes *vpn*, e através de uma modificação do protocolo *mp-bgp* para o transporte dos valores de prioridade das rotas. Estas modificações são apresentadas no capítulo 3 através da especificação em *SDL* da expansão da arquitetura *vpn-mpls* (seção 3.2).

De acordo com a proposta, a troca de informações de prioridade entre o roteador *ce* e o

roteador *pe*, possibilita o estabelecimento de novos níveis de qualidade de serviço através da inserção de novos mapeamentos ou modificação de mapeamentos já existentes. Os valores de prioridade das rotas a serem informados pelo cliente *vpn* são obtidos a partir do campo *dscp* da arquitetura de serviços diferenciados, e transportados pela modificação realizada no protocolo *mp-bgp*, dentro do estado *established* (figura 2.3), através do uso do campo *route target* da mensagem *update*. Com as informações de rotas e prioridades das rotas, o roteador *pe* monta a tabela de mapeamento de prioridades do cliente *vpn*. De acordo com a necessidade do cliente, cada mapeamento contido nesta tabela pode ter seus valores de prioridade alterados através da troca de mensagens *update* entre o roteador *ce* e o roteador *pe*.

O capítulo 3 apresenta as propostas em *SDL* da arquitetura *vpn-mpls* definida por [12](sistema *basicarquitecture-scenario1*, seção 3.1), e da expansão da arquitetura sugerida nesta tese(sistema *basicarquitecture-scenario2*, seção 3.2). Os sistemas propostos apresentam as suas estruturas em *SDL* com orientação à objetos através do uso da ferramenta *SDL TAU Suite* [6], que permite simplificar as especificações, além de validar e simular alguns casos críticos apresentados no capítulo 4, garantindo a especificação correta destes sistemas e possibilitando possíveis modificações, expansões e a implementação dos sistemas. No capítulo 5 são sugeridos alguns cenários de simulação para análise de desempenho da proposta de qualidade de serviço na arquitetura *vpn-mpls*, e comparações dos resultados de vazão, atraso fim-a-fim e perda de pacotes das aplicações multimídias são realizados, com o uso do simulador *Opnet Modeler*[3].

2.2.1 A linguagem SDL e a ferramenta SDL TAU Suite

A linguagem formal *SDL* (*Specification and Description Language*), começou a ser desenvolvida em 1972 pela CCITT, hoje ITU-T, para a especificação e a descrição de sistemas de telecomunicações. Sua primeira versão foi lançada em 1976, seguida por novas versões sendo lançadas de quatro em quatro anos. A partir da versão *SDL-92*, a linguagem incorporou conceitos de orientação à objetos, como; tipos, reuso e herança. A linguagem *SDL* foi criada visando a especificação e descrição de sistemas telefônicos, sendo depois utilizada também na especificação de protocolos em redes de telecomunicações. Atualmente sua aplicação é muito mais abrangente, sendo utilizada na especificação de sistemas de informação [8], sistemas distribuídos

[35], sistemas de tempo real, etc.

A linguagem *SDL* serve tanto para representar o comportamento do sistema, como sua estrutura, usando diferentes níveis de abstração, desde o nível mais alto até o mais detalhado. Um sistema *SDL* é dividido em blocos, que se comunicam uns com os outros e com o ambiente. Cada bloco é composto por um conjunto de processos que também se comunicam entre si e até mesmo com outros blocos. O processo, por sua vez, trata-se do nível de abstração onde é especificado o comportamento do sistema através da definição de estados e transições. Sempre que necessário, podem ser especificados procedimentos (dentro de processos) que realizam alguma tarefa específica.

Os processos se comunicam entre si de duas formas diferentes: através da troca de sinais (modo assíncrono) ou através da chamada a procedimentos exportados (modo síncrono). Através destes sinais, e procedimentos exportados, os processos podem trocar dados e trabalhar cooperativamente, dando a funcionalidade desejada ao sistema especificado. Os processos podem ser definidos como tipos instanciáveis, ou seja, é especificado um comportamento padrão para um determinado processo e podem ser criadas inúmeras instâncias deste processo. Estas instâncias são executadas concorrentemente e podem se comunicar umas com as outras.

Os blocos e processos *SDL* podem herdar suas características de um ancestral, através da construção *inherits*. Neste caso, as funcionalidades do ancestral também estão presentes no bloco/processo herdeiro, além de novas funcionalidades que podem ser incorporadas. Além disso, algumas funcionalidades podem ser redefinidas através da construção *redefined*, desde que isso seja permitido pelo ancestral através do uso da construção *virtual*. Blocos, processo, procedimentos e transições de estado, devido a recepção de sinais podem ser declarados como *virtual* e, portanto, redefinidos.

Além da definição de tipos, instâncias, herança e redefinição, outra funcionalidade importante da linguagem *SDL* é representada pelo mecanismo de *package*, que permite reusar todas as definições de um sistema já especificado, ou apenas algumas delas, num propósito similar às bibliotecas de funções. A inclusão de um *package* num sistema é feita através da cláusula *use*.

Uma das características fortes do *SDL* é que, diferentemente de outras linguagens de especificação formal (como *Estelle* e *Lotos*, por exemplo), ela apresenta uma representação gráfica,

denominada *SDL-GR*, que facilita bastante o seu entendimento pelo usuário, já que a descrição do sistema é feita de uma maneira muito mais próxima àquela a que se esta acostumado. Além disso, por ser uma linguagem formal, possui uma sintaxe e uma semântica bem definidas, possibilitando a simulação e a validação automática do sistema sendo especificado, mesmo nas fases iniciais do desenvolvimento, através de uma ferramenta *case* como o ambiente *SDL TAU Suite* [6] [7].

Neste trabalho, todo o desenvolvimento em *SDL* foi realizado fazendo-se uso da ferramenta *case SDL TAU Suite*, que permite o desenvolvimento das especificações em *SDL* através de sua representação gráfica (*SDL-GR*), facilitando assim a compreensão do sistema como um todo. A ferramenta *SDL TAU Suite* também disponibiliza módulos capazes de realizar a validação da especificação desenvolvida e simulações de suas funcionalidades e casos críticos. Estas facilidades foram utilizadas para a detecção e correção de erros (ou falhas de especificação), o que garante a corretude das especificações geradas.

O apêndice A apresenta maiores detalhes das principais construções *SDL* e da estruturação da linguagem. É também apresentada a ferramenta *SDL TAU Suite* e suas principais facilidades, utilizadas neste trabalho.

3

Especificação formal da arquitetura *vpn-mpls* para provimento de qualidade de serviço

3.1 Introdução

Com base nas descrições apresentadas nas seções anteriores do capítulo 2, referentes a arquitetura *vpn-mpls* padronizada pelo *IETF* (*Internet Engineering Task Force*) [12] e a proposta de expansão desta arquitetura para provimento de qualidade de serviço, neste capítulo são apresentadas as propostas de especificações formais em *SDL* (*Specification and Description Language*) da arquitetura *vpn-mpls* e da expansão da arquitetura *vpn-mpls*.

A arquitetura *vpn-mpls* foi proposta pelo grupo de trabalho *l3vpn* (*layer 3 VPN*) do *IETF* em 2000, também denominada de arquitetura de *vpn bgp/mpls IP* pois utiliza a extensão do

protocolo *bgp* , denominado *mp-bgp*[37], para troca de informações de sinalizações e o *mpls* [13] para encaminhamento do tráfego.

A provisão de qualidade de serviço na arquitetura *vpn-mpls* descrita por [12], [27], [15], [17] e [26] propõem a implementação da arquitetura de serviços diferenciados no cliente, também denominada arquitetura *diffserv* [36], interoperando com a rede *mpls* do provedor através do mapeamento das prioridades do cliente no campo *exp* dos rótulos *mpls* [14]. Também conhecido como *e-lsp* (*exp-inferred-psc lsp*) [16], este mapeamento das prioridades do cliente é feito pelo provedor de serviço nos roteadores de borda, denominados roteadores *pe* (*provider edge*). É importante ressaltar que estes trabalhos citados levam em consideração o mapeamento entre as prioridades do cliente e os acordos de nível de serviço contratados, sendo implementado de forma estática pelo provedor de serviço, ou seja, o cliente *vpn* não consegue estabelecer novos níveis de qualidade de serviços em tempo real.

A proposta de expansão da arquitetura *vpn-mpls* introduzida nesta tese é baseada na necessidade de estabelecimento de níveis de qualidade de serviços em tempo real, ou seja, estabelecimento de novos mapeamentos de prioridades de acordo com a necessidade do cliente *vpn*. Estes novos mapeamento são informados ao provedor de serviço através das modificações realizadas na estrutura do protocolo de roteamento *mp-bgp* (*multiprotocol - border gateway control protocol*) apresentadas na seção 3.3 deste capítulo. De acordo com a proposta, a troca de informação de prioridade entre o roteador *ce* e o roteador *pe* possibilita o estabelecimento de novos níveis de qualidade de serviço, através da inserção de novos mapeamentos ou modificação de mapeamentos já existentes. Os valores de prioridade das rotas a serem informados pelo cliente *vpn* são obtidos a partir do campo *dscp* da arquitetura de serviços diferenciados, e transportados pelo protocolo *mp-bgp* através do uso do campo *route target* da mensagem *update*. Com as informações de rotas e prioridades das rotas, o roteador *pe* monta a tabela de mapeamento de prioridades do cliente *vpn*. De acordo com a necessidade do cliente, cada mapeamento contido nesta tabela pode ter seus valores de prioridade alterados através da troca de mensagens *update*, feita na modificação do protocolo *mp-bgp*, entre o roteador *ce* e o roteador *pe*.

Na seção 3.2 é apresentada a especificação formal em *SDL* da arquitetura *vpn-mpls* baseada em [12]. As entidades e as sinalizações são especificadas utilizando a estrutura hierárquica

da linguagem *SDL*. Na seção 3.3 é apresentada a especificação da expansão da arquitetura *vpn-mpls*, com ênfase nas modificações realizadas no sistema ancestral.

3.2 Especificação formal da arquitetura *vpn-mpls*

A proposta da especificação deste sistema é detalhar o comportamento dos componentes da arquitetura *vpn-mpls* e sinalizações trocadas pelo protocolo de roteamento *mp-bgp* para o estabelecimento das Redes Privadas Virtuais (*VPN*). Este sistema é a base para o desenvolvimento da proposta de expansão da arquitetura *vpn-mpls* descrito na seção 3.3.

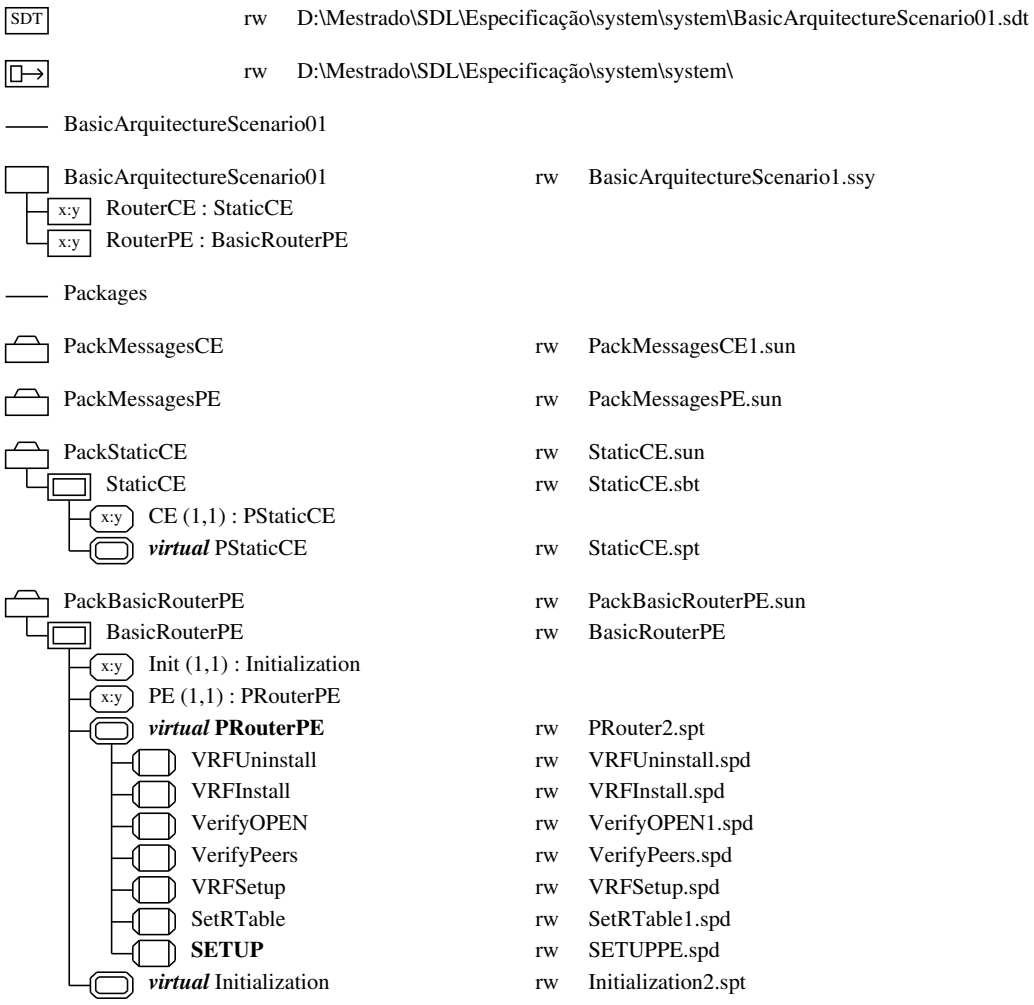


Figura 3.1: Visão do Organizer (*SDL TAU Suite*) do sistema *basicarquitecture-scenario1*

Especificou-se neste trabalho o comportamento de um roteador *ce* e um roteador *pe* com o propósito de caracterizar de forma detalhada a interação entre o cliente *vpn*(roteador *ce*) e provedor de serviço (roteador *pe*). Durante a especificação optou-se por utilizar os quatro níveis de hierarquia do *SDL*: sistema, bloco, processo e procedimento com o objetivo de facilitar o entendimento do sistema e para que uma possível evolução neste trabalho, ou em outros trabalhos de pesquisa, pudesse ser realizada utilizando-se dessas estruturas através dos conceitos de herança e de orientação a objetos.

Estrutura *SDL*	**Especificação**
Pacote	*Package packmessage-ce*[a], *packmessage-pe*[a], *packstatic-ce* e *packbasicrouter-pe*
Sistema	*System basicarquitecture-scenario1*
Bloco	*Block static-ce* e *basicrouter-pe*
Processo	*Process pstatic-ce*, *prouter-pe* e *initialization*[a]
Procedimentos	*Procedure setup*, *set-rtable*[a], *vrf-setup*[a], *verify-peers*[a], *verify-open*[a], *vrf-install*[a] e *vrf-uninstall*[a]

Tabela 3.1: Estruturas *SDL* da especificação da arquitetura do sistema *basicarquitecture-scenario1*

[a]Estes diagramas estão presentes no apêndice C.

A figura 3.1 apresenta a organização hierárquica em *SDL* do sistema *basicarquitecture-scenario1* especificado. Nesta figura os dois primeiros ícones simbolizam o arquivo principal de especificação que contém o sistema, e o diretório raiz onde se apresentam todos os arquivos utilizados na especificação. Abaixo destes ícones, por motivos de organização da especificação, duas áreas de trabalho foram definidas, denominadas *basicarquitecture-scenario1* e *packages*. Na primeira área de trabalho encontra-se a especificação do diagrama raiz que é o sistema *basicarquitecture-scenario1*(figura 3.2), contendo os blocos *static-ce*(figura 3.6) e *basicrouter-pe*(figura 3.8). Na segunda área de trabalho encontra-se a especificação dos pacotes

(*packmessage-ce*, *packmessage-pe*, *packstatic-ce* e *packbasicrouter-pe*) utilizados pelo sistema.

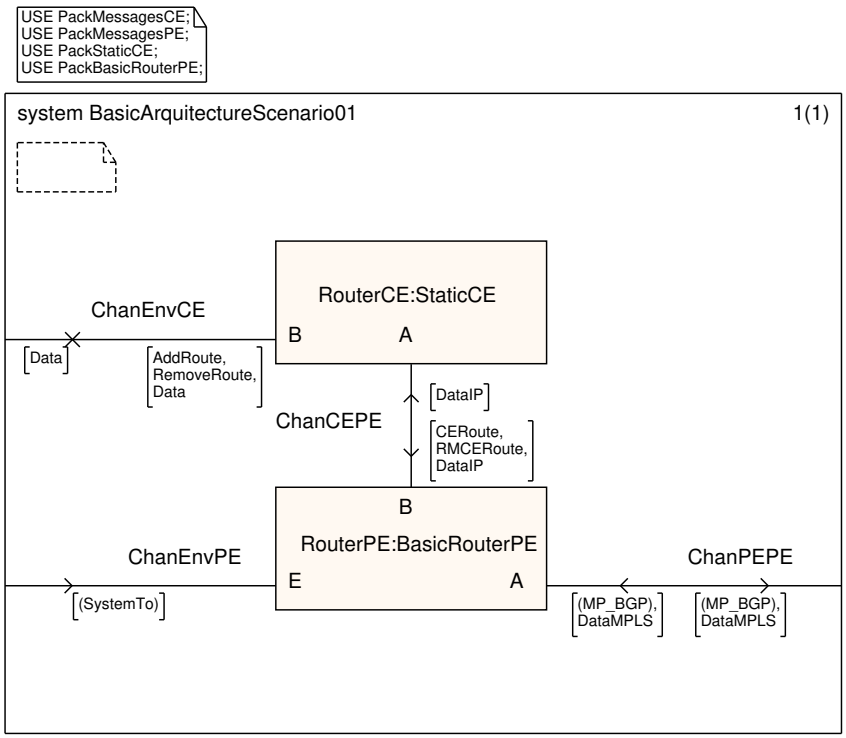


Figura 3.2: Sistema *basicarquitecture-scenario1*

As estruturas *SDL* que foram criadas para a especificação deste sistema são apresentadas na tabela 3.1 e podem ser observadas no diagrama de hierarquia da figura 3.1. De acordo com a tabela 3.1, a estrutura do sistema *basicarquitecture-scenario1* especificado faz uso de quatro pacotes(*packages*); *packmessage-ce*, *packmessage-pe*, *packstatic-ce* e *packbasicrouter-pe*. Os pacotes *packmessage-ce* e *packmessage-pe* apresentam a descrição em *SDL* das mensagens e listas de sinais utilizadas na especificação do sistema. A especificação das mensagens e lista de sinais em *packages* facilita a especificação em *SDL* orientada a objetos, visto que podem ser reutilizadas por outros *packages* e por outros sistemas, como é o caso do sistema *basicarquitecture-scenario2* (figura 3.12) descrito na seção 3.3 que irá utiliza-las na especificação da proposta de expansão da arquitetura *vpn-mpls*.

O pacote *packstatic-ce*(figura 3.3) apresenta a especificação do bloco *static-ce* (figura 3.6), que representa o comportamento do roteador *ce* do cliente *vpn*. De acordo com a figura 3.1, este pacote é composto pelo bloco *static-ce*, processo *pstatic-ce* e instância *ce*. O pacote

packbasicrouter-pe(figura 3.4) apresenta a especificação do bloco *basicrouter-pe* (figura 3.8), que representa o comportamento do roteador *pe*. De acordo com a figura 3.1, este pacote é composto pelo bloco *basicrouter-pe*, processos *initialization* e *prouter-pe*, e pelas instâncias *init* e *pe*. Os diagramas das especificações que não forem mostrados neste capítulo, podem ser consultados no apêndice C, que apresenta todas os diagramas utilizados pelas especificações.

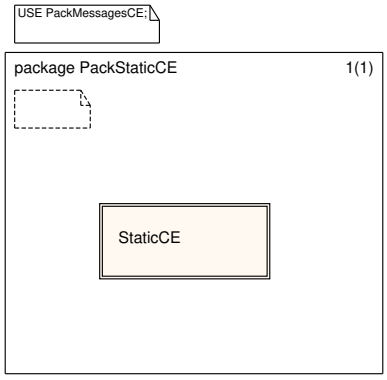


Figura 3.3: Pacote *packstatic-ce*

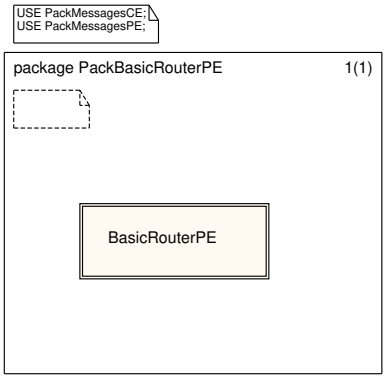


Figura 3.4: Pacote *packbasicrouter-pe*

A figura 3.2 apresenta o sistema *basicarquitecture-scenario1*, que representa a estrutura em *SDL* da arquitetura *vpn-mpls*. Os dois componentes principais da arquitetura (roteador *ce* e roteador *pe*) são representados em *SDL* pelos blocos *static-ce* e *basicrouter-pe*. Através do retângulo com uma borda dobrada localizado na parte superior esquerda da figura 3.2, o sistema *basicarquitecture-scenario1* faz uso dos pacotes (*packmessage-ce*, *packmessage-pe*, *packstatic-ce* e *packbasicrouter-pe*) especificados através da cláusula *USE*.

Na figura 3.2, a interação entre o bloco *static-ce*(figura 3.6) e o bloco *basicrouter-pe*(figura 3.8) é representado pelo canal de comunicação *chan-cepe*, e as interações entre estes blocos e o ambiente externo são representadas pelos canais de comunicação; *chan-envce, chan-envpe* e *chan-pepe*. O canal de comunicação, *channel*, é usado para troca de sinais entre dois componentes do sistema especificado em *SDL*. Cada canal pode trocar informações em um único sentido, como é o caso do canal *chanenv-pe* que troca informações do ambiente externo para o bloco *basicrouter-pe*, ou em ambos os sentidos, como é o caso dos demais canais usados na especificação da figura 3.2.

Os blocos *static-ce* e *basicrouter-pe* usados pelo sistema *basicarquitecture-scenario1* (figura

3.2) foram especificados dentro de uma estrutura de *packages*, que permite estes blocos serem instânciados pelo sistema através dos pacotes *packstatic-ce* (figura 3.3) e *packbasicrouter-pe* (figura 3.4) ou herdados por outra especificação, como será descrito na seção 3.3.

Os pacotes *packstatic-ce* (figura 3.3) e *packbasicrouter-pe* (figura 3.4), através da cláusula *USE* (presente no retângulo com uma borda dobrada no canto superior das figuras 3.3 e 3.4), faz uso dos sinais e listas de sinais especificados nos pacotes *packmessage-ce* e *packmessage-pe*.

A figura 3.5 representa as listas de sinais usadas para definir os sinais que trafegam nos canais do sistema *basicarquitecture-scenario1* (figura 3.2). A lista de sinais *mp-bgp*, representa os sinais (*open, keepalive, notification, update e tcp*) que podem ser enviados e recebidos pelo bloco *basicrouter-pe* (figura 3.8) durante as sessões *mp-bgp*. A lista de sinais *system-to*, representa os sinais (*start-sys* e *error-sys*) que podem ser enviados do ambiente externo(*environment*) para o sistema a fim de inicializá-lo. A lista de sinais *init-to*, representa os sinais (*mpbgp-start* e *mpbgp-error*) enviados pelo processo *initialization* (figura 3.8) a fim de inicializar a maquina de estado do protocolo *mp-bgp*.

A declaração do bloco *static-ce*(figura 3.6) no pacote *packstatic-ce* (figura 3.3) e do bloco *basicrouter-pe*(figura 3.8) no pacote *basicrouter-pe*(figura 3.4) como *type*, representados nas figuras 3.3 e 3.4 pelos retângulos com bordas duplas, permite que estes blocos sejam herdados e instânciados pelo sistema *basicarquitecture-scenario1* (figura 3.2).

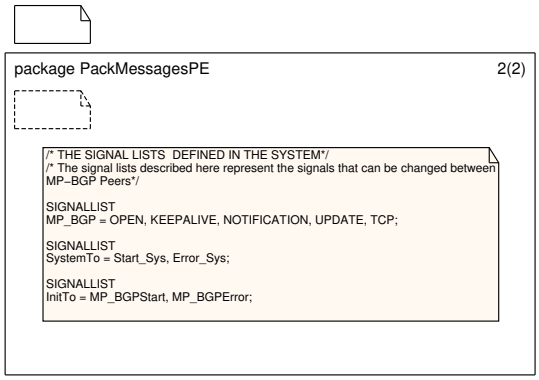


Figura 3.5: Definição das listas de sinais do sistema *basicarquitecture-scenario1* (figura 3.2)

Bloco *static-ce*

O bloco *static-ce*(figura 3.6) é a especificação do bloco que representa o roteador *ce* da arquitetura *vpn-mpls*, onde a troca de informações de roteamento entre o roteador *ce* e roteador *pe* é feita de forma estática, ou seja, não se utiliza nenhum protocolo para troca de informações de roteamento entre estes roteadores.

A declaração do bloco *static-ce* como *block type* (figura 3.6), permite que o bloco seja herdado e instanciado pelo sistema *basicarquitecture-scenario1* (figura 3.2), através da cláusula *USE packstatic-ce*. A troca de sinais do bloco *static-ce* com o bloco *basicrouter-pe*(figura 3.8) e com o ambiente externo, ocorre através dos *gateways* A e B respectivamente.

O bloco *static-ce* é composto pelo processo *pstatic-ce* (figura 3.7), que é responsável por armazenar as informações de rotas do cliente *vpn* e trafegar informações de dados entre a rede do cliente *vpn* e do provedor de serviço. Este processo possui uma instância (*ce*) representada pela notação (1,1). A declaração do processo *pstatic-ce*(figura 3.7) como *virtual process type*, torna o processo possível de ser instanciado pelo bloco *pstatic-ce*(figura 3.6) ou redefinido por outros sistemas.

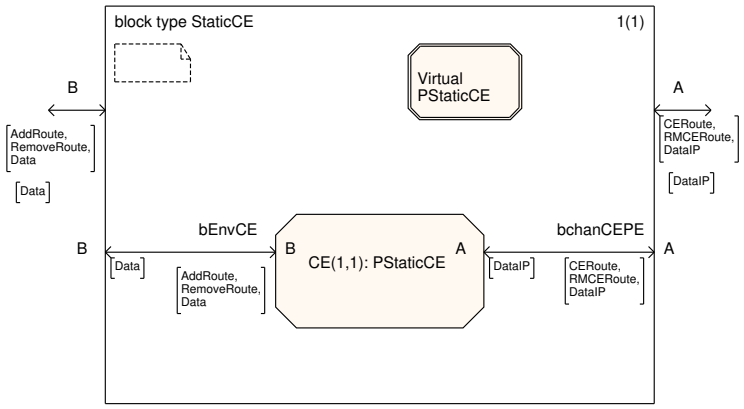


Figura 3.6: Bloco *static-ce* do sistema *basicarquitecture-scenario1* (figura 3.2)

Na figura 3.7 inicia-se a descrição comportamental do processo *pstatic-ce* com um símbolo *start* contendo a palavra *virtual*, onde a próxima transição é o estado *idle*. A declaração *virtual* usada no processo *pstatic-ce* torna a especificação possível de ser redefinida em outros sistemas, podendo assim, adicionar novas funcionalidades ao processo ou alterar as já existentes.

Através do *gateway* B o ambiente externo (*environment*) insere e remove informações de rotas no processo *pstatic-ce*(figura 3.7) com o uso dos sinais *add-route* e *remove-route* e troca

informações de trafego de dados com o uso do sinal *data*. Ao receber uma informação de rota a ser inserida, sinal *add-route* enviado pelo ambiente externo, o processo *pstatic-ce* instala esta rota na tabela de roteamento do roteador *ce*, através da *task rt-install*, e repassa essa informação para o bloco *basicrouter-pe*(figura 3.8) através do sinal *ce-route*. A remoção de rotas também procede da mesma forma, com o recebimento do sinal *remove-route* do ambiente externo, remoção da rota da tabela de roteamento através da *task rt-uninstall*, e envio de remoção de rota para o bloco *basicrouter-pe* através do sinal *rmce-route*. A troca de tráfego de dados é realizada com o ambiente externo através do sinal *data* via *gateway* B e com o bloco *basicrouter-pe* através do sinal *data-ip* via *gateway* A.

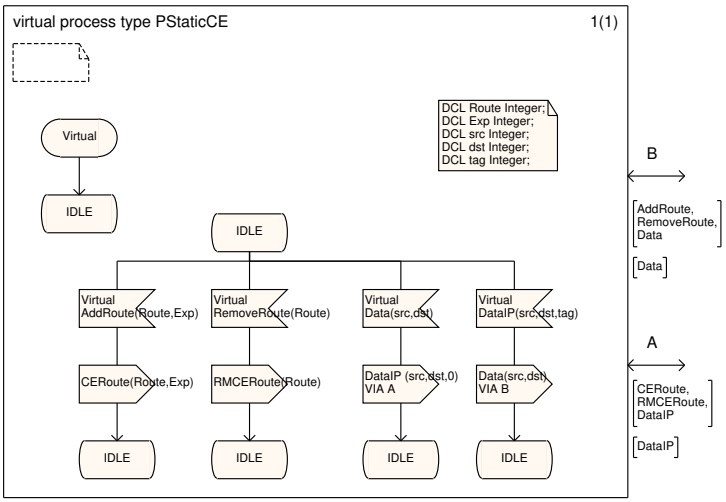


Figura 3.7: Processo *pstatic-ce* do bloco *static-ce* (figura 3.6)

Bloco *basicrouter-pe*

O bloco *basicrouter-pe* (figura 3.8) é a especificação do bloco que representa o roteador *pe* da arquitetura *vpn-mpls*. Este bloco é responsável por receber informações de rotas do bloco *static-ce*(figura 3.6), e trocar informações de roteamento através do protocolo *mp-bgp* com o ambiente externo.

A declaração do bloco *basicrouter-pe*(figura 3.8) como *block type*, permite que o bloco seja herdado e instanciado pelo sistema *basicarquitecture-scenario1*(figura 3.2), através da cláusula *USE packbasicrouter-pe* e redefinido pelo sistema *basicarquitecture-scenario2*(figura 3.13) conforme será mostrado na seção 3.3.

O bloco *basicrouter-pe* é composto pelos processos *initialization* e *prouter-pe* e pelas instâncias *init* e *pe*. O processo *initialization* tem a função de inicializar a maquina de estado do protocolo *mp-bgp* presente no processo *prouter-pe*(figura 3.9). O processo *prouter-pe*(figura 3.9) implementa as principais funcionalidades do roteador *pe*, como; a tabela *vrf* (*virtual routing and forwarding*) do cliente *vpn* e o protocolo *mp-bgp* responsável pela troca de informações de roteamento com a rede do provedor de serviço.

A declaração dos processo *initialization* e *prouter-pe*(figura 3.9) como *virtual process type*, torna os processos possíveis de serem instânciados pelo bloco *basicrouter-pe* (figura 3.8) e redefinidos pelo sistema basicarquitectur-scenario2(figura 3.13) na seção 3.3.

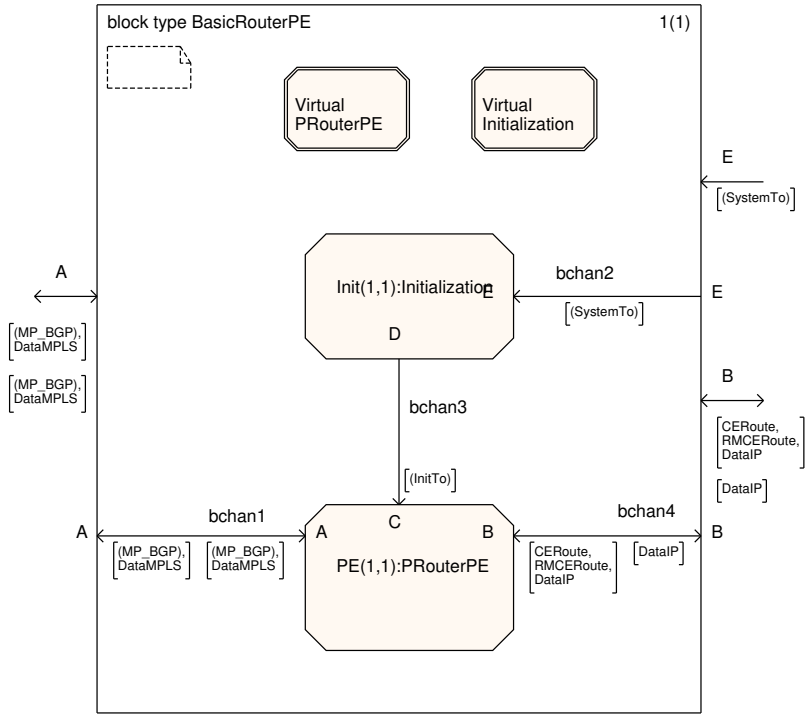


Figura 3.8: Bloco *basicrouter-pe* do sistema *basicarquitecture-scenario1* (figura 3.2)

A figura 3.9 apresenta parte da especificação do processo *prouter-pe*, responsável por implementar a tabela *vrf* do cliente *vpn* e o protocolo *mp-bgp*. A especificação deste processo foi dividida em diagramas, e a figura 3.9 representa um dos sete diagramas utilizados para especificar o processo *prouter-pe*, conforme pode ser visto no canto superior direito da figura 3.9. Neste diagrama esta especificado parte da descrição comportamental do processo *prouter-pe* e a lista de procedimentos usadas pelo processo.

Antes da descrição comportamental do processo, é importante descrever as funções básicas dos procedimentos para melhor entendimento da especificação. Abaixo é apresentada uma descrição resumida da função de cada um dos procedimentos do processo *prouter-pe*, na ordem em que aparecem na figura 3.9.

Procedimento *setup*: Este procedimento é responsável por inicializar as variáveis globais (versão do protocolo *mp-bgp*, número do sistema autônomo e endereço *IP*) do roteador *pe*. Além disso, este procedimento inicializa a tabela de pares *mp-bgp* que armazena os endereços *IP* dos roteadores *pe* que irão estabelecer sessões *mp-bgp*. A tabela de pares, *peers-table*, é inicializada configurando-se o endereço dos roteadores e o número do sistema autônomo (*asn - autonomous system number*) ao qual o roteador pertence.

Procedimento *set-rtable*: Este procedimento é responsável por inicializar a tabela de roteamento do roteador *pe*, *ip-table*. Nesta tabela estão as informações das rotas do roteador *pe* e informações dos próximos *hops* para alcançar as rotas.

Procedimento *vrf-setup*: Este procedimento tem a função de inicializar as variáveis da tabela *vrf*, como; *route distinguisher* (rd), *import* e *export target*, e o rótulo *mpls* identificador da *vrf*. Além disso, este procedimento também inicializa a tabela *vrf* que contém as informações iniciais das rotas da *VPN*.

Procedimento *verify-peers*: Este procedimento tem a função de, dado um endereço *IP*, realizar uma busca na tabela de pares *mp-bgp* (*peers-table*) para verificar se o endereço *IP* dado está presente na tabela, ou seja, se o endereço *IP* faz parte da tabela de pares *mp-bgp*. O valor retornado (*recv-ip*) igual a 0 (zero) caso não haja este valor na tabela ou, é igual a 1 (um) caso haja este valor na tabela.

Procedimento *verify-open*: Este procedimento tem a função de verificar se a mensagem *open* recebida está em conformidade com os parâmetros necessários, ou seja, o procedimento *verify-open* checa todos os campos da mensagem para verificar se há alguma inconsistência na mensagem recebida. O valor retornado pelo procedimento (*ckopen*) igual a 0 (zero) caso

ocorra alguma inconsistência nos campos da mensagem, sendo informada através da variável *erro-subcode* ou, é igual a 1 (um) caso não ocorra nenhuma inconsistência na mensagem.

Procedimento *vrf-install*: Este procedimento é responsável por inserir novas rotas na tabela *vrf* da *VPN*. Este procedimento é executado quando o roteador *pe* recebe a nova rota via mensagem *update* enviada pelo ambiente externo ou via mensagem *ce-route* enviada pelo processo *pstatic-ce* (figura 3.7).

Procedimento *vrf-uninstall*: Este procedimento tem a função de remover rotas da tabela *vrf*, e é executado sempre que uma mensagem *update* enviada pelo ambiente externo contém rotas a serem removidas no campo *withdraw* da mensagem ou caso o processo *pstatic-ce* (figura 3.7) através do sinal *rmce-route*, sinalize a intenção de remover uma rota da tabela.

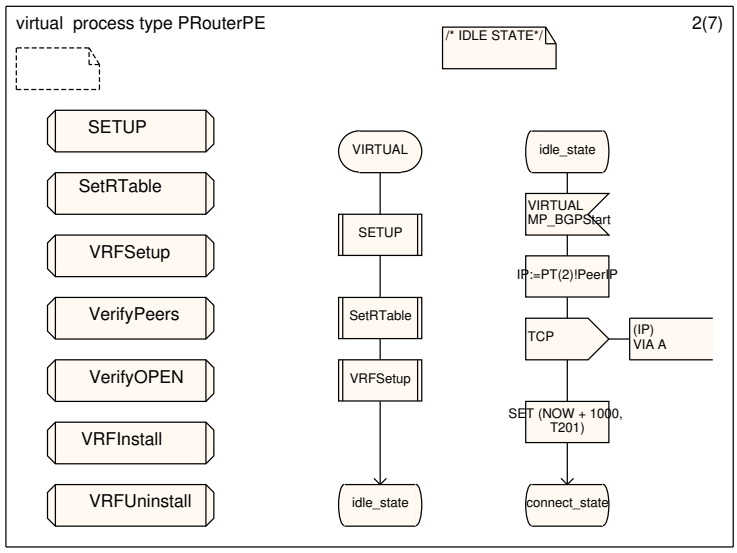


Figura 3.9: Estado *idle*, processo *prouter-pe* do bloco *basicrouter-pe*(figura 3.8)

A descrição comportamental do processo *prouter-pe* presente na figura 3.9, representa o comportamento da maquina de estado do protocolo *mp-bgp* para o estado *idle*. De acordo com a figura 3.9, durante a criação da instância *pe* (figura 3.8), o processo *prouter-pe* inicializa todos os recursos *mp-bgp* através das chamadas aos procedimentos *setup, set-rtable* e *vrf-setup* e o processo entra no estado *idle*(*idle-state*).

Seguindo a estrutura do diagrama *SDL* da figura 3.9, estando o processo *prouter-pe* no estado *idle*, ao receber um sinal *mpbgp-start*, vindo do processo *initialization*, para a inicialização da maquina de estado do protocolo *mp-bgp*, o processo *prouter-pe* verifica na tabela *peers-table* o endereço *IP* do roteador com quem se estabelecerá a sessão *mp-bgp*. Ao enviar o sinal *tcp* para o endereço escolhido, o roteador *pe* tenta iniciar o estabelecimento da sessão *mp-bgp*. Após o envio do sinal *tcp* o temporizador *t201* (*connectretry timer* descrito em [41]) é inicializado a fim de se estabelecer um tempo para o recebimento da confirmação de abertura da sessão *mp-bgp* e o processo *prouter-pe* entra no estado *connect*.

A figura 3.10 representa o estado *open sent* do protocolo *mp-bgp*. A descrição dos estados *open sent*(figura 3.10), e *established*(figura 3.11) é importante pois o comportamento de cada um destes estados será redefinido no sistema *basicarquitecture-scenario2* na seção 3.3.

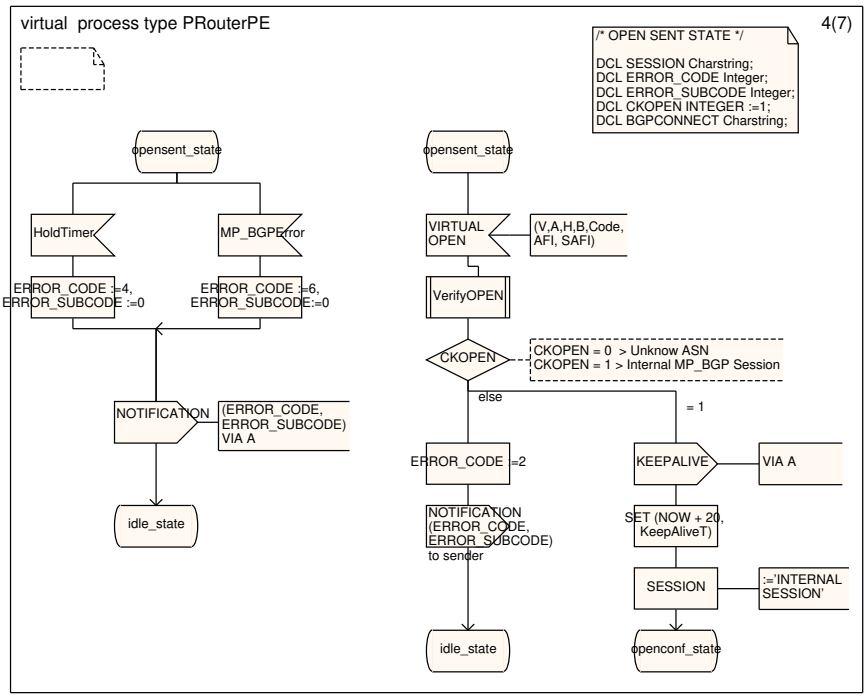


Figura 3.10: Estado *open sent*, processo *prouter-pe* do bloco basicrouter-pe(figura 3.8)

Na figura 3.10, estado *open sent*, o estabelecimento da sessão *mp-bgp* é finalizado com o recebimento do sinal *open* pelo processo *prouter-pe*. De acordo com a figura, ao receber um sinal *open*, o processo faz uma chamada ao procedimento *verify-open* a fim de verificar a versão do protocolo suportada e o número do sistema autônomo do sinal *open* recebido. Caso a versão

do protocolo ou o número de sistema autônomo não seja suportada pelo roteador *pe*, o processo *prouter-pe* envia o sinal *notification* para o emissor do sinal *open* e entra no estado *idle* . O sinal *notification* é enviado contendo o código e sub-código da notificação ocorrida (variáveis *error-code* e *error-subcode* do sinal *notification*).

Caso o processo *prouter-pe* suporte a versão do protocolo *mp-bgp* e o número do sistema autônomo presente no sinal *open* recebido, o processo envia um sinal *keepalive* para o roteador *pe* que originou o sinal *open* (representado pelo ambiente externo), inicializa o temporizador *keepalivet*, e a sessão *mp-bgp* do tipo interna (*mp-ibgp*) é estabelecida, entrando o processo no estado *open confirm*. A sessão *mp-ibgp* (*mp-ibgp - multiprotocol - internal border gateway protocol*) é uma sessão *mp-bgp* estabelecida dentro do sistema autônomo do provedor de serviço.

De acordo com a figura 3.10, caso o sinal *open* não seja recebido dentro do período de tempo estabelecido pelo temporizador *holdtimer* descrito no estado *connect*, a tentativa de estabelecimento de sessão *mp-bgp* expira através do recebimento do sinal *holdtimer* e o processo *prouter-pe* envia o sinal *notification* informando ao ambiente externo a expiração da sessão *mp-bgp*, entrando o processo no estado *idle*. Em resposta a intenção do sistema em finalizar todas as sessões *mp-bgp* via sinal *mpbgp-error*, o processo *prouter-pe* encerra as sessões através do envio do sinal *notification* para o ambiente externo, entrando o processo no estado *idle*.

A figura 3.11 representa o estado *open confirm* e parte do estado *established* do processo *prouter-pe*. No estado *open confirm* o processo aguarda o recebimento dos sinais *keepalive* e *notification* vindos do ambiente externo. Com o recebimento do sinal *keepalive*, simbolizando a manutenção da sessão *mp-bgp*, o processo *prouter-pe* reinicializa o temporizador *holdtimer* e entra no estado *established*. Caso o processo não receba o sinal *keepalive* dentro do tempo estabelecido pelo temporizador *keepalivet*, o processo *prouter-pe* recebe o sinal *keepalivet* do temporizador e envia um sinal *keepalive* para o ambiente externo tentando manter a sessão *mp-bgp* estabelecida, reinicializa o temporizador e mantém o processo no estado *open confirm*. Se durante a sessão *mp-bgp*, o processo *prouter-pe* receber o sinal *notification*, a sessão é encerrada e o processo vai para o estado *idle*.

O estado established (figura 3.11) representa o estado do protocolo *mp-bgp* em que o processo *prouter-pe* pode trocar informações de roteamento com o ambiente externo através do sinal

update, e manutenção da sessão *mp-bgp* através dos sinais *keepalive* e *notification*. De acordo com a figura 3.11, estando o processo no estado *established*, ao receber um sinal *update*, o processo verifica se existe rota a ser removida da tabela *vrf* através da variável *droute* do sinal *update*. Se existir rota a ser removida, esta é removida através da chamada ao procedimento *vrf-uninstall*. Após a remoção de rotas, o processo *prouter-pe* faz a instalação da rota contida no sinal *update*, através da chamada ao procedimento *vrf-install*, instalando assim a nova rota na tabela *vrf*, permanecendo o processo no estado *established*.

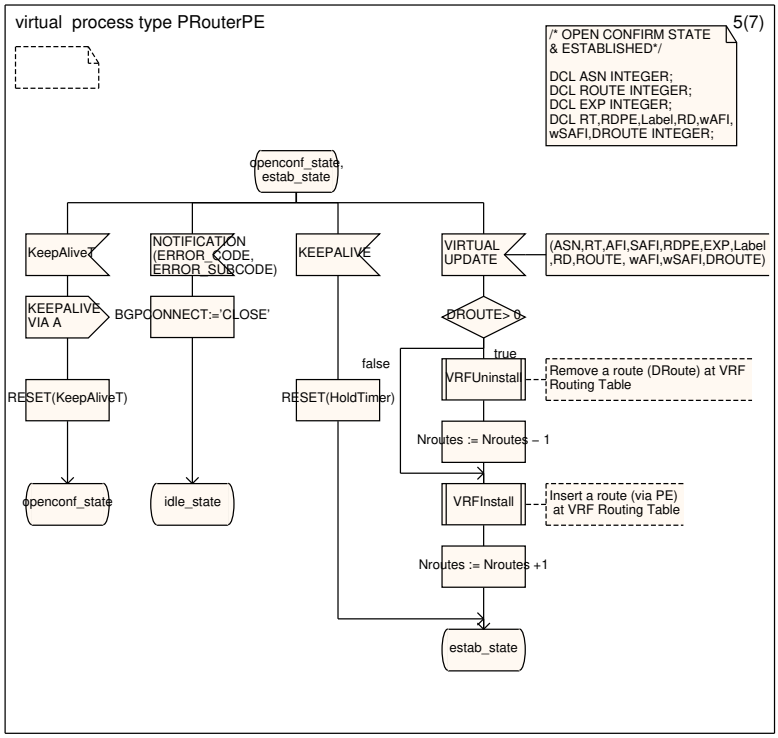


Figura 3.11: Estado *open confirm* e *established*, processo *prouter-pe* do bloco basicrouter-pe(figura 3.8)

3.3 Especificação formal da proposta de expansão da arquitetura *vpn-mpls*

A proposta de expansão da arquitetura *vpn-mpls* é baseada na proposição de implementação de qualidade de serviço, onde o cliente *vpn* (roteador *ce*) conectado ao provedor de serviço

(roteador *pe*) troca informações de sinalização a fim de prover qualidade de serviço fim-a-fim. As informações de sinalização são trocadas através das modificações feitas na estrutura do protocolo de roteamento *mp-bgp* [37](*multiprotocol - border gateway control protocol*).

Através da inserção do valor de prioridade de cada rota sugerida pelo cliente *vpn* através das mensagens *update* trocadas nas sessões *mp-bgp*, e alterações feitas no comportamento da arquitetura para que esta nova informação seja tratada pelos roteadores *ce* e *pe*, consegue-se prover qualidade de serviço na arquitetura *vpn-mpls*. O valor de prioridade de rota designado pelo cliente *vpn* através da sessão *mp-bgp* é mapeado pelo roteador *pe* dentro do campo *exp* dos rótulos *mpls* ao entrar na rede do provedor de serviço.

O mapeamento de prioridades da arquitetura *diffserv* no campo *exp* dos rótulos *mpls* já foi sugerido por alguns trabalhos, como [31] e [17]. A diferença entre estas propostas de mapeamento e o trabalho descrito nesta tese esta na possibilidade de inserção de novos mapeamentos e alteração de mapeamentos já existentes de acordo com a necessidade do cliente. Nesta proposta o cliente *vpn* pode a qualquer instante, com uso da modificação do protocolo *mp-bgp* sugerida neste trabalho, incluir ou alterar rotas da *vpn* neste mapeamento de prioridade localizado na tabela *vrf* do roteador *pe*.

As alterações no comportamento da arquitetura foram realizadas com a criação do bloco *routermpbgp-ce*(figura 3.16) e redefinição dos estados *open sent*(figura 3.10) e *established*(figura 3.11) descritos no processo *prouter-pe* pertencente ao bloco *basicrouter-pe*(figura 3.8) do sistema basicarquitecture-scenario1 (figura 3.2).

A figura 3.12 apresenta a organização hierárquica em *SDL* orientada à objetos do sistema que representa a proposta de expansão da arquitetura *vpn-mpls* (sistema *basicarquitecture-scenario2*). Na figura 3.12 a organização da especificação segue o mesmo critério descrito para o sistema *basicarquitecture-scenario1* (figura 3.1). Duas áreas de trabalho foram definidas, denominadas *basicarquitecture-scenario2* e *packages*. Na primeira área encontra-se a especificação do diagrama raiz que é o sistema *basicarquitecture-scenario2*(figura 3.13), contendo os blocos *routermpbgp-ce* (figura 3.16) e *routermpbgp-pe*(3.17). Na segunda área encontra-se a especificação dos pacotes *packmessage-ce*, *packmessage-pe* e *packbasicrouter-pe* reutilizados pelo sistema e os novos pacotes *packmpbgp-ce* e *packmpbgp-pe*.

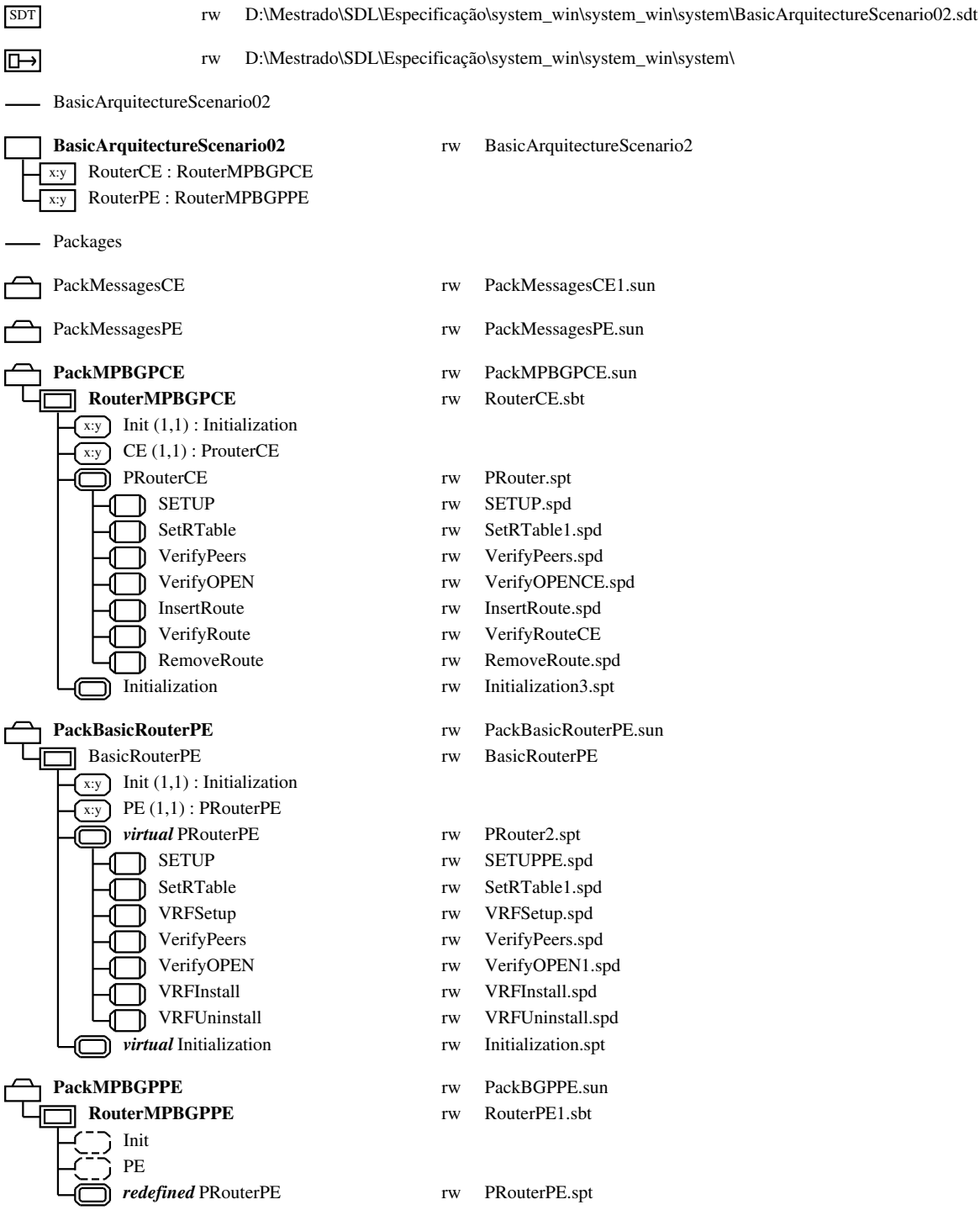


Figura 3.12: Visão do Organizer (*SDL TAU Suite*) do sistema *basicarquitecture-scenario2*

Estrutura t	Especificação
Pacote	*Package packmessage-ce*[a], *packmessage-pe*[a], *packbasicrouter-pe*, *packmpbgp-ce* e *packmpbgp-pe*
Sistema	*System basicarquitecture-scenario2*
Bloco	*Block routermpbgp-ce*, *basicrouter-pe*[a]e *routermpbgp-pe*
Processo	*Process prouter-ce*[a], *initialization*[a], *prouter-pe*[a]e *redefined prouter-pe*
Procedimentos	*Procedure setup*[a], *set-rtable*[a], *vrf-setup*[a], *verify-peers*[a], *verify-open*[a], *insert-route*[a], *verify-route*[a], *remove-route*[a], *vrf-install*[a]e *vrf-uninstall*[a]

Tabela 3.2: Estruturas *SDL* da especificação da arquitetura do sistema *basicarquitecture-scenario2*

[a]Estes diagramas estão presentes no apêndice C.

As estruturas *SDL* que foram criadas para a especificação deste sistema são apresentadas na tabela 3.2 e podem ser observadas no diagrama de hierarquia da figura 3.12. De acordo com a tabela 3.2, a estrutura do sistema *basicarquitecture-scenario2* faz uso de cinco pacotes; *packmessage-ce*, *packmessage-pe* e *packbasicrouter-pe* reutilizados do sistema *basicarquitecture-scenario1*(figura 3.1), e *packmpbgp-ce* e *packmpbgp-pe* criados para o sistema *basicarquitecture-scenario2*(figura 3.13).

O pacote *packmpbgp-ce*, apresenta a especificação do bloco *routermpbgp-ce*(figura 3.16), o qual representa o comportamento do roteador *ce* do cliente *vpn* que implementa o protocolo *mp-bgp* para troca de informações de roteamento e de qualidade de serviço com o roteador *pe*. De acordo com a figura 3.12, o pacote *packmpbgp-ce* é composto pelo bloco *routermpbgp-ce*, que contém os processos *prouter-ce* e *initialization*, e as instâncias *ce* e *init*.

O pacote *packmpbgp-pe*(figura 3.15), apresenta a especificação do bloco *routermpbgp-pe*(figura 3.17), o qual representa representa o comportamento do roteador *pe*. De acordo com a figura 3.12, o pacote *packmpbgp-pe* é composto pelo bloco *routermpbgp-pe*, que contém a redefinição do processo *prouter-pe* especificado no pacote *packbasicrouter-pe* (figura 3.14).

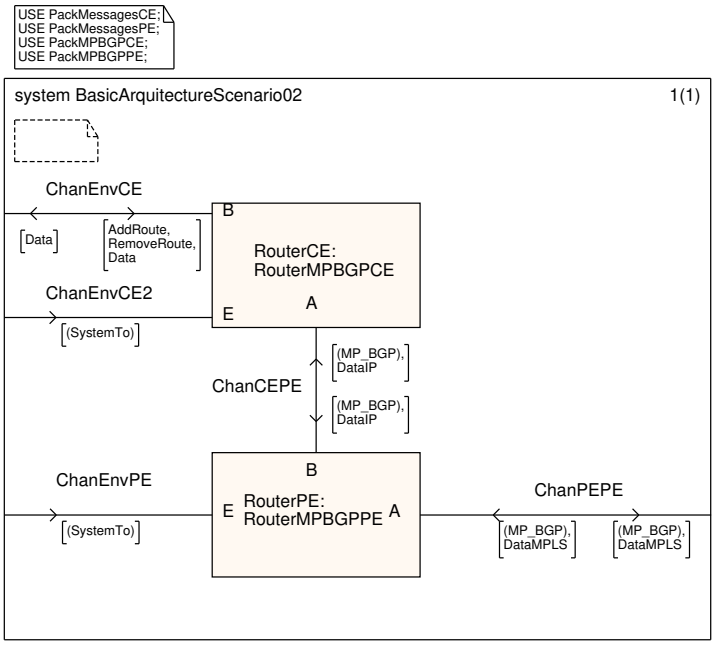


Figura 3.13: Sistema *basicarquitecture-scenario2*

A figura 3.13 apresenta o sistema *basicarquitecture-scenario2* que representa a estrutura em *SDL* da proposta de extensão da arquitetura *vpn-mpls*. Os dois componentes principais da arquitetura (roteador *ce* e roteador *pe*) são representados pelos blocos *routermpbgp-ce*(figura 3.16) e *routermpbgp-pe*(figura 3.17). Através do retângulo com uma borda dobrada localizado na parte superior esquerda da figura 3.13, o sistema *basicarquitecture-scenario2* faz uso dos pacotes(*packmessage-ce*, *packmessage-pe*, *packbasicrouter-pe*, *packmpbgp-ce* e *packmpbgp-pe*) especificados através da cláusula USE. As interações entre o bloco *routermpbgp-ce*, bloco *routermpbgp-pe* e o ambiente externo são representados pelos canais de comunicação *chan-cepe*, *chan-envce*, *chan-envce2*, *chan-envpe* e *chan-pepe*. A diferença entre os canais de comunicação deste sistema e os especificados no sistema *basicarquitecture-scenario1* (figura 3.2) descrito na seção 3.2, está na inclusão do canal *chan-envce2* para inicialização do protocolo *mp-bgp* especificado no bloco *routermpbgp-ce*(figura 3.16), e a alteração das listas de sinais transportadas pelo canal *chan-cepe*, que para este novo sistema (sistema *basicarquitecture-scenario2*) deve transportar mensagens *mp-bgp* entre os blocos *routermpbgp-ce* e *routermpbgp-pe*.

Os blocos *routermpbgp-ce*(figura 3.16) e *routermpbgp-pe*(figura 3.17) usados pelo sistema *basicarquitecture-scenario2* (figura 3.13) foram especificados dentro de uma estrutura de *pack-*

ages, que permite estes blocos serem instânciados pelo sistema através dos pacotes *packmpbgp-ce* e *packmpbgp-pe*.

A declaração do bloco *basicrouter-pe* no pacote *packbasicrouter-pe* como *type*, representado na figura 3.14 pelo retângulo com borda dupla, permite que o processo *prouter-pe* presente neste bloco seja herdado, através da cláusula *USE packbasicrouter-pe*, e redefinido pelo bloco *routermpbgp-pe* presente no pacote *packmpbgp-pe* (figura 3.15).

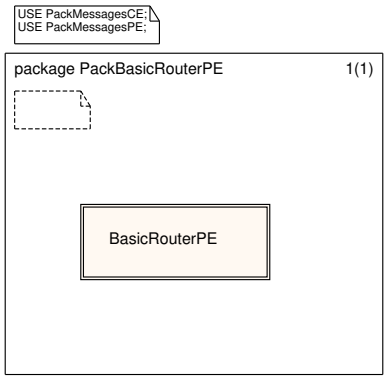


Figura 3.14: Pacote *packbasicrouter-pe*

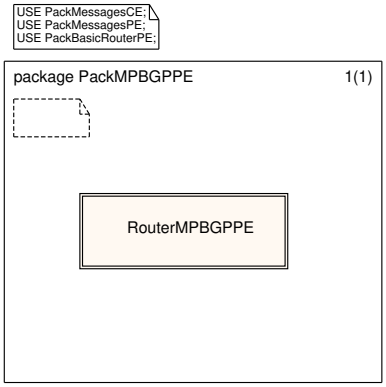


Figura 3.15: Pacote *packmpbgp-pe*

Bloco *routermpbgp-ce*

Na proposta de expansão da arquitetura *vpn-mpls*, o cliente *vpn* (roteador *ce*) deve trocar informações de roteamento através das modificações realizadas no protocolo *mp-bgp*. A declaração do bloco *routermpbgp-ce* como *block type*, permite que o bloco seja herdado e instanciado pelo sistema *basicarquitecture-scenario2*(figura 3.16), através da cláusula *USE packmpbgp-ce*. A troca de sinais do bloco *routermpbgp-ce* com o bloco *routermpbgp-pe*(figura 3.17) ocorre através do *gateway* A, e do bloco *routermpbgp-ce* com o ambiente externo através dos *gateways* B e E.

O bloco *routermpbgp-ce* (figura 3.16) é composto pelos processos *initialization* e *prouter-ce* e pelas instâncias *init* e *ce*. O processo *initialization* tem a função de inicializar a maquina de estado do protocolo *mp-bgp* presente no processo *prouter-ce*. O processo *prouter-ce* implementa o protocolo de roteamento *mp-bgp*, que faz uso da mesma estrutura do processo *prouter-pe* descrito no bloco *basicrouter-pe* (figura 3.12), com a diferença que o processo *prouter-ce* não implementa as funcionalidades de tabela *vrf* e interface *mpls* com a rede do provedor de serviços da arquitetura *vpn-mpls*.

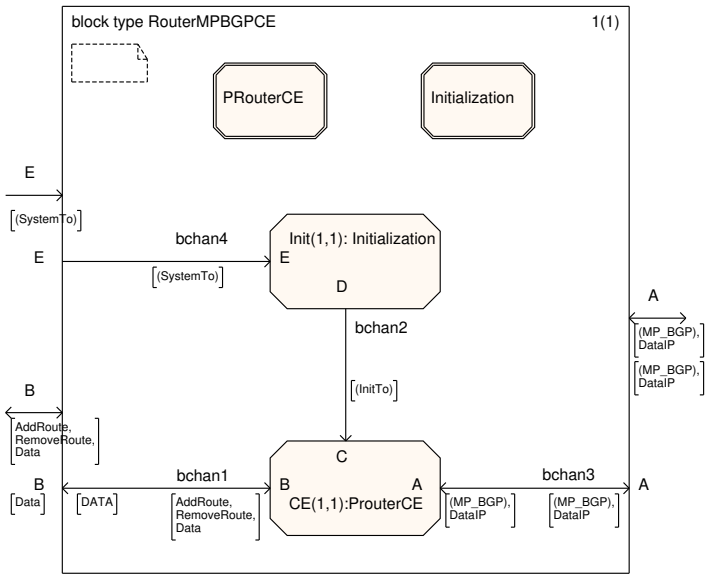


Figura 3.16: Bloco *routermpbgp-ce* do sistema *basicarquitecture-scenario2*(figura 3.13)

Bloco *routermpbgp-pe*

O bloco *routermpbgp-pe* (figura 3.17) é a especificação do roteador *pe* da proposta de expansão da arquitetura *vpn-mpls*. Este bloco é responsável por trocar informações de rotas do cliente *vpn* via protocolo *mp-bgp* com o roteador *ce*, representado pelo bloco *routermpbgp-ce* (figura 3.16), e com os roteadores *pe* pertencentes a rede do provedor de serviço, representado pelo ambiente externo.

A declaração do bloco *routermpbgp-pe*(figura 3.17) como *block type*, permite que este bloco seja herdado e instanciado pelo sistema *basicarquitecture-scenario2*(figura 3.13), através da cláusula *USE packmpbgp-pe* . O bloco *routermpbgp-pe* herda através da cláusula *inherits basicrouter-pe*, localizada dentro do retângulo pontilhado na figura 3.17, todas as características necessárias do bloco *basicrouter-pe* especificado no pacote *packbasicrouter-pe*(figura 3.12). Pode-se verificar, na figura 3.17, as instâncias *init* e *pe* herdadas do bloco *basicrouter-pe* (figura 3.8) e representadas em *SDL* pelo retângulo pontilhado com bordas achatadas.

É importante ressaltar que algumas características do bloco *basicrouter-pe*(figura 3.8) tiveram que ser redefinidas no bloco *routermpbgp-pe*(figura 3.17). As redefinições feitas no bloco *routermpbgp-pe* estão localizadas no processo *prouter-pe*. A redefinição é representada no bloco *routermpbgp-pe* da figura 3.17 pelo retângulo com bordas duplas denominado *redefined prouter-*

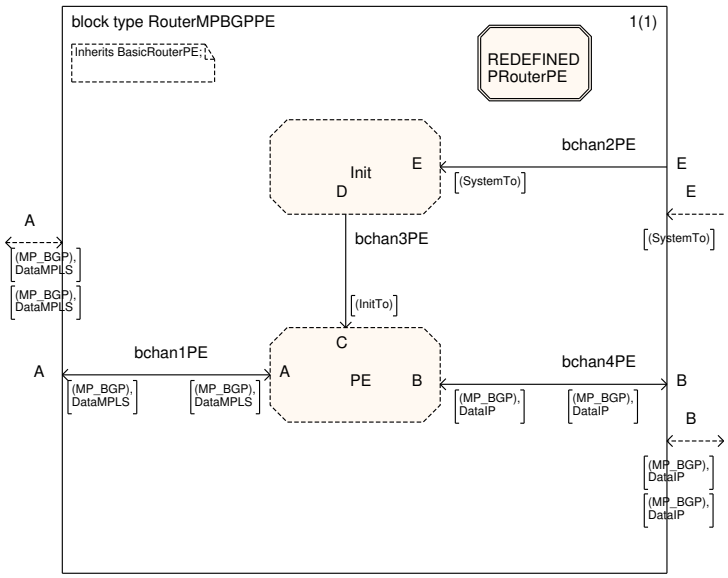


Figura 3.17: Bloco *routermpbgp-pe* do sistema *basicarquitecture-scenario2*(figura 3.13)

pe.

As redefinições feitas no processo *prouter-pe* visam permitir o roteador *pe* estabelecer sessões *mp-bgp* com o roteador *ce*, e através destas sessões trocar informações de prioridade sugeridas pelo cliente *vpn* através do roteador *ce*. A redefinição feita no estado *open sent*, mostrada na figura 3.18, permiti o processo *prouter-pe* estabelecer sessões *mp-bgp* com o processo *prouter-ce*(roteador *ce*) e com o ambiente externo (roteadores *pe* da rede do provedor) através da troca de sinais *keepalive*. A troca de informações de prioridade se torna possível através da redefinição do estado *established*, mostrada na figura 3.19, que passa a tratar o campo *route target* do sinal *update* proveniente do roteador *ce* como valor de prioridade da rota.

A redefinição do estado *open sent* é realizada a partir do recebimento do sinal *open*, descrito na figura 3.18 pelas palavras *redefined open* localizadas dentro do símbolo de recebimento de sinal da linguagem *SDL*. De acordo com a figura 3.18, ao receber um sinal *open*, o processo redefinido *prouter-pe* faz uma chamada ao procedimento *verify-open*, a fim de verificar a versão do protocolo *mp-bgp* suportada e o número do sistema autônomo do sinal *open* recebido. Caso a versão do protocolo ou o número do sistema autônomo não seja suportado pelo roteador *pe*, o processo *prouter-pe* envia o sinal *notification* para emissor do sinal *open* e entra no estado *idle*. O sinal *notification* é enviado contendo o código e sub-código da notificação ocorrida (variáveis

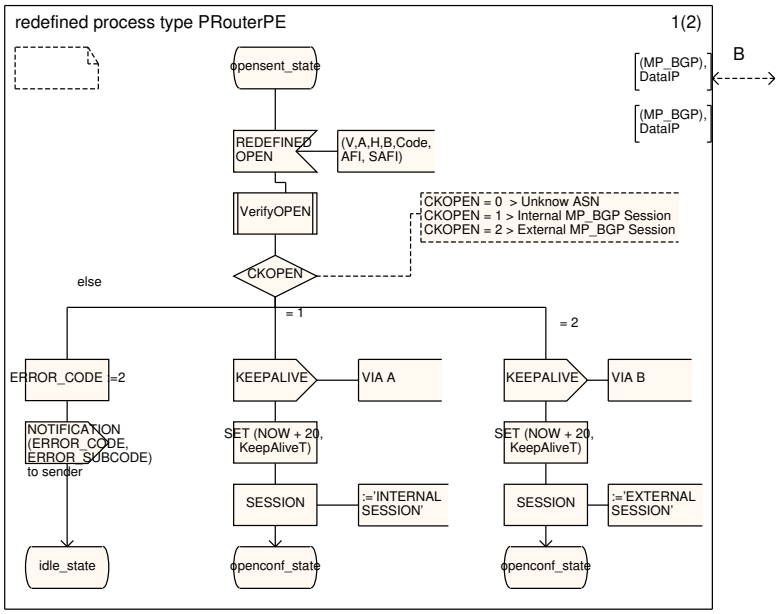


Figura 3.18: Redefinição do processo *prouter-pe* - estado *open sent* (ver figura 3.10)

error-code e *error-subcode* do sinal *notification*).

Caso o processo *prouter-pe* suporte a versão do protocolo *mp-bgp* e o número do sistema autônomo presente no sinal *open* for o mesmo do processo *prouter-pe*, indicando um sinal *open* vindo do ambiente externo, o processo *prouter-pe* envia um sinal *keepalive* para o roteador *pe* que originou o sinal *open*, inicializa o temporizador *keepalivet*, e a sessão *mp-bgp* do tipo interna (mp-ibgp) é estabelecida, entrando o processo no estado *open confirm*. A sessão *mp-ibgp*(*multiprotocol - internal border gateway protocol*) é uma sessão *mp-bgp* estabelecida dentro do sistema autônomo do provedor de serviço, ou seja, sessão *mp-bgp* estabelecida entre roteadores pe da rede do provedor de serviço.

Caso o processo *prouter-pe* suporte a versão do protocolo *mp-bgp* e o número do sistema autônomo presente no sinal *open* for do roteador *ce*(processo *prouter-ce*, figura 3.16), o processo *prouter-pe* envia um sinal *keepalive* para o processo *prouter-ce* que originou o sinal *open* (bloco *routermpbgp-ce* da figura 3.16), inicializa o temporizador *keepalivet*, e a sessão *mp-bgp* do tipo externa (*mp-ebgp - mutiprotocol - external border gateway protocol*) é estabelecida, entrando o processo no estado *open confirm*. A sessão *mp-ebgp* é uma sessão *mp-bgp* estabelecida entre sistemas autônomos diferentes, ou seja, o número do sistema autônomo do provedor é diferente

do número do sistema autônomo do cliente *vpn*.

Outra redefinição necessária realizada no processo *prouter-pe*, é apresentada na figura 3.19. Esta redefinição visa permitir a troca de informações de rotas e informações de prioridade de rotas através do sinal *update*, via *gateway* B, entre o roteador *pe* (bloco *routermpbgp-pe* da figura 3.17) e o roteador *ce* (bloco *routermpbgp-ce* da figura 3.16). Esta redefinição é realizada no estado *established* da maquina de estado do protocolo *mp-bgp*.

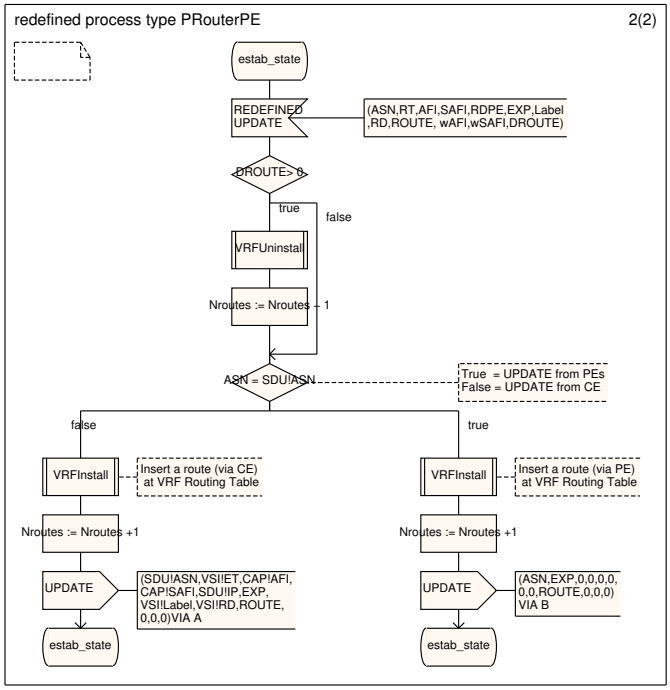


Figura 3.19: Redefinição do processo *prouter-pe* - estado *established* (ver figura 3.11)

De acordo com a figura 3.19, estando o processo no estado *established*, ao receber um sinal *update*, o processo verifica se existe rota a ser removida da tabela *vrf*, através da variável *droute* do sinal *update*. Se existir rota a ser removida, esta é removida através da chamada ao procedimento *vrf-uninstall*. Após a remoção da rota, o processo *prouter-pe* checa se o sinal *update* é proveniente do roteador *ce*, ou do ambiente externo, através da verificação do número do sistema autônomo do sinal *update*. Para ambos os casos a nova rota transportada no sinal *update* é instalada na tabela *vrf* (procedimento *vrf-install*), e informada ao roteador *ce* (bloco *routermpbgp-ce*, figura 3.16), para o sinal *update* vindo do roteador *pe* (ambiente externo), ou informada aos roteadores *pe*, para o sinal *update* vindo do roteador *ce* (bloco *routermpbgp-ce*,

figura 3.16).

4

Resultados da Validação e das Simulações

A Validação e as Simulações dos sistemas especificados neste trabalho foram realizados através do *Validator* e *Simulator*, ferramentas integrantes do pacote *SDL TAU Suite*[6].

A validação é um resultado obtido pela ferramenta *Validator* que auxilia no desenvolvimento de especificações usando a linguagem *SDL*. A validação é um resultado que possibilita o aumento da produtividade e qualidade no desenvolvimento de sistemas em *SDL*. A validação identifica possíveis erros ocorridos durante a especificação do sistema e verifica todos os estados possíveis do sistema analisando o comportamento de cada sinal. Os erros encontrados são corrigidos na especificação e o sistema se torna mais confiável para a implementação, possibilitando assim o conhecimento de inconsistências e problemas no estágio inicial de desenvolvimento.

A simulação é um resultado obtido pela ferramenta *Simulator*. A simulação é utilizada para verificar a dinâmica do sistema em casos especiais da especificação. É utilizada principalmente

para se verificar, de forma mais detalhada, o comportamento do sistema em alguns casos críticos.

Na seção 4.1 são apresentados os resultados das validações dos sistemas especificados no capítulo 3 e é analisado como o processo de validação pode ser utilizados para identificar erros da especificação. Na seção 4.2 são apresentadas algumas simulações visando exemplificar o funcionamento do sistema e a analise de casos críticos. Os resultados da simulação são mostrados através de gráficos *MSC* (*Message Sequence Chart*) [20].

4.1 Validação

A ferramenta *Validator* utilizada pelo *SDL TAU Suite* apresenta três algoritmos para eventuais validações: *Random Walk*, *Exhaustive Exploration* e *Bit State Exploration*.

O *Random Walk* é indicado para sistemas de maior porte, onde o número de estados é considerável, mas também é eficiente para sistemas menores. Este algoritmo percorre de forma aleatória os ramos da arvore com os estados possíveis do sistema, simplificando o gerenciamento de estados percorridos e não percorridos.

O *Exhaustive Exploration* é o algoritmo utilizado para percorrer todos os caminhos possíveis dos estados do sistema, sendo o mais adequado para sistemas pequenos. À medida que o número de estados cresce significativamente, a utilização do algoritmo se torna impraticável.

O *Bit State Exploration* possui características mais complexas e é utilizado para analisar sistemas razoavelmente grandes, como é o caso dos sistemas especificados neste trabalho. Os resultados deste algoritmo fornecem dados estatísticos percorridos no sistema especificado e utilizam de uma estrutura denominada *hashtable* para gerenciar os estados percorridos.

Para a execução da validação utilizando o algoritmo *Bit State Exploration*, alguns parâmetros são necessários:

- *Hash Table Size*: Indica o tamanho máximo da tabela *hash* em bytes usada para armazenar os códigos *hash* dos estados;
- *Depth*: Indica a profundidade máxima da árvore de estados atingida pelo algoritmo.

A seguir são apresentados os resultados obtidos para os sistemas durante o processo de validação e discutidos os resultados de maior relevância.

4.1.1 A Validação dos sistemas SDL especificados

Para os sistemas *SDL* desenvolvidos (sistema *basicarquitecture-scenario1* e *basicarquitecture-scenario2*), os parâmetros utilizados durante a validação foram: *depth* = 10.000 e *hash table size* = 7.000.000. Alguns parâmetros de menor valor foram testados, porém à medida que eram aumentados, o número de estados alcançados também aumentava. A partir destes valores (*depth* = 10.000 e *hash table size* = 7.000.000), mesmo que eles sejam aumentados, o mesmo número de estados continua sendo percorrido.

A figura 4.1 apresenta os resultados de validação dos sistemas *basicarquitecture-scenario1* (figura 3.2) e *basicarquitecture-scenario2* (figura 3.13). O sistema *basicarquitecture-scenario1* é a superclasse especificada para que o sistema *basicarquitecture-scenario2* herdasse suas características.

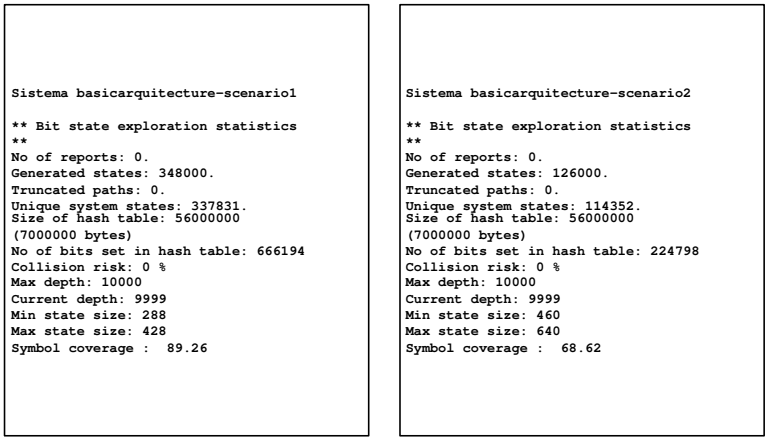


Figura 4.1: Resultados da validação dos sistema *basicarquitecture-scenario1* (figura 3.2) e *basicarquitecture-scenario2* (figura 3.13)

Interpretando os resultados da validação do sistema *basicarquitecture-scenario1* e do sistema *basicarquitecture-scenario2*, o número de mensagens (*number of reports*) igual a 0 (zero) significa que nenhum erro ou aviso foi encontrado em ambos os sistemas. De acordo com a figura 4.1, o número de estados gerados (*generated states*) representa o número total de estados do sistema gerados no processo de validação. O número de estados únicos do sistema (*unique system states*)

representa o número de estados do sistema gerados pela validação que não foram duplicados em nenhum lugar da árvore de comportamento. Árvore de comportamento é uma estrutura de árvore que representa o comportamento do sistema *SDL*. Os nós da árvore representam os estados do sistema *SDL*, que são definidos como; instâncias de processos ativos, valores de variáveis de processo, controle de fluxo do estado *SDL* da instância do processo, chamadas de procedimentos, sinais que estão presentes nas filas do sistema, e temporizadores (*timers*) ativos. Logo pode-se verificar que, o número de estados gerados pelo processo de validação (figura 4.1) não representa apenas o número de estados da especificação. Na validação dos sistemas, o número de caminhos truncados (*truncated paths*) significa a quantidade de vezes que a validação alcançou a profundidade máxima (*max depth*) da árvore de comportamento. Logo, o caminho de execução naquele estado está truncado, e a exploração do processo de validação continua em outro estado.

O resultado de risco de colisão (*collision risk*) no processo de validação fazendo uso do algoritmo *bit state* (figura 4.1), representa o risco (em porcentagem) de colisão na tabela *hash* usada para representar os estados gerados pelo sistema. Isto significa que, o tamanho da alocação de espaço (*bit array*) usado na tabela *hash* é um fator importante para definir o comportamento da exploração *bit state*. Pois a cada vez que um novo estado é checado através da comparação do valor atual com o valor anterior da tabela hash existe um risco de colisão. Quanto maior o valor da tabela hash, menor é o risco de colisão, ou seja, menor a probabilidade de dois estados diferentes possuirem o mesmo código *hash* calculado. O risco de colisão (*collision risk*) para ambos sistemas (sistema *basicarquitecture-scenario1* e *basicarquitecture-scenario2*) foi de 0 %. Na figura 4.1 os valores *min state size* e *max state size* representam o menor e o maior número de *bytes* usados pelo processo de validação para armazenar um estado do sistema durante o cálculo do código *hash*.

Conforme a figura 4.1, o sistema *basicarquitecture-scenario1* apresenta 89,26 % dos símbolos percorridos (*symbol coverage*) enquanto o sistema *basicarquitecture-scenario2* apresenta 68,62 %. A porcentagem de símbolos percorridos é menor no sistema *basicarquitecture-scenario2* devido à herança das características do sistema ancestral(sistema *basicarquitecture-scenario1*), através do uso e redefinição do pacote *packbasicrouter-pe*(figura 3.1) pelo pacote *packmpbgp-*

pe(figura 3.12). Na validação do sistema *basicarquitecture-scenario2*, a varredura dos símbolos é feita em ambos os pacotes(*packmpbgp-pe* e *packbasicrouter-pe*), as partes que não foram redefinidas são percorridas no sistema ancestral (*pacote packbasicrouter-pe*), e as partes redefinidas são percorridas no novo sistema(pacote *packmpbgp-pe*). Portanto, a porcentagem de símbolos percorridos no sistema *basicarquitecture-scenario2* diminuem. Outro motivo para a diminuição dessa porcentagem de cobertura dos símbolos é o aumento da presença de ações condicionais na especificação, por exemplo, a condição de *if*. Apenas uma das condições, verdadeiro ou falso, será percorrida, diminuindo a quantidade de símbolos percorridos.

Os símbolos não alcançados podem ser identificados através do *Coverage View*, que exibe uma árvore com a hierarquia de pacotes, blocos, processos, procedimentos e símbolos do sistema *SDL*, mostrando o que foi e o que não foi testado pela validação. A figura 4.2 mostra a parte do resultado do *Coverage View* para a validação do sistema *basicarquitecture-scenario2*(figura 3.13). Os pacotes, blocos, processos, procedimentos e símbolos preenchidos com cinza indicam que foram testados, enquanto os parcialmente preenchidos indicam que alguns de seus símbolos não foram alcançados. Estes símbolos não alcançados podem ser identificados na especificação *SDL* para que se possa verificar se há algum erro na especificação ou se eles tratam casos inatingíveis pela validação.

A figura 4.2 descreve a árvore composta pelo sistema *basicarquitecture-scenario2*(figura 3.13), que apresenta os pacotes *packmpbgp-ce*, *packmpbgp-pe* e *packbasicrouter-pe*. Na cobertura dos símbolos, os pacotes *packmpbgp-ce* e *packbasicrouter-pe* não estão totalmente preenchidos, o que significa que alguns símbolos destes pacotes não foram totalmente percorridos. Para o *packbasicrouter-pe*(figura 3.15) isto se deve ao fato do pacote *packmpbgp-pe*(figura 3.14) herda-lo, ou seja, alguns símbolos declarados como *virtual* no pacote *packbasicrouter-pe* são redefinidos no pacote *packmpbgp-pe*. Desta forma estes símbolos nunca são alcançados pelo processo de validação, uma vez que estes símbolos foram redefinidos, diminuindo a porcentagem dos símbolos percorridos neste pacote, e conseqüentemente no sistema *basicarquitecture-scenario2*(figura 3.12). Para o pacote *packmpbgp-ce* isto se deve ao aumento da presença de ações condicionais na especificação, ou seja, durante o processo de validação apenas uma das condições é percorrida, diminuindo assim a porcentagem dos símbolos percorridos.

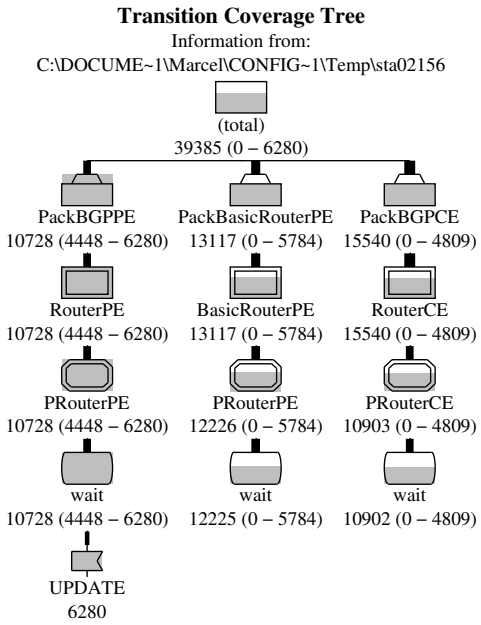


Figura 4.2: Símbolos percorridos no sistema *basicarquitecture-scenario2*(figura 3.13)

Para que esta porcentagem (89,26 %) fosse atingida durante a validação, foi utilizado um recurso de não-determinismo da linguagem *SDL* para atribuir valores de teste aos parâmetros de entrada dos sinais que o usuário envia ao sistema. Desta forma, diversas possibilidades diferentes foram testadas. A figura 4.3 apresenta parte da definição de valores atribuídos que foram usados pelo processo de validação. Nesta figura esta representada a definição dos valores do sinal *update*, para diversas possibilidades, enviados ao sistema *basicarquitecture-scenario2*(figura 3.12) durante o processo de validação.

```
clear-signal-definitions UPDATE
define-signal UPDATE 100 100 1 128 100 0 22 100 100 1 128 22
define-signal UPDATE 2 11 55 100 2 65000 22 0 65000 1 4 55
define-signal UPDATE 128 128 100 65000 0 1 22 65000 55 1 128 0
define-signal UPDATE 0 0 0 128 22 4 1 1 55 65000 1 4
define-signal UPDATE 4 128 65000 128 100 4 1 22 22 0 1 55
define-signal UPDATE 2 11 100 11 0 128 1 2 11 22 1 2
define-signal UPDATE 128 128 128 55 55 65000 4 4 4 55 55 2
define-signal UPDATE 22 100 0 4 65000 11 4 11 22 22 55 22
define-signal UPDATE 2 65000 2 55 128 4 2 4 55 0 65000 11
define-signal UPDATE 100 128 0 1 22 22 2 22 128 100 11 100
```

Figura 4.3: Definição dos valores de teste para o sinal *update*

4.1.2 Detecção de erros de especificação através da validação

Através do processo de validação, pode-se detectar erros de especificação que seriam dificilmente encontrados manualmente. Com o uso do *Coverage View*, pode-se identificar os símbolos não alcançados pela validação, e assim tem-se uma noção mais precisa dos pontos onde podem haver erros.

A figura 4.4 exemplifica a detecção de um erro em uma comparação realizada no procedimento *verify-open* do processo *prouter-pe*(figura 3.9). O erro pode ser encontrado após a execução do *Validator*, analisando-se o resultado apresentado pelo *Coverage Viewer*. Nenhum dos símbolos abaixo de um dos ramos da comparação foi alcançado, o que sugere que a comparação esteja errada. Após a correção, o sistema foi validado novamente e todos os símbolos foram alcançados.

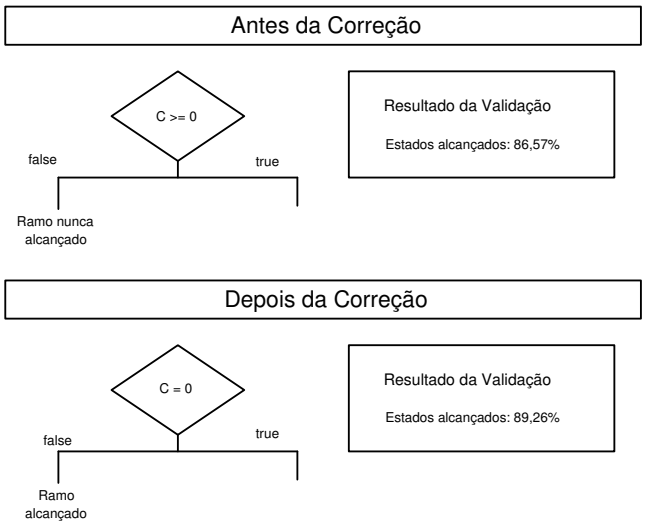


Figura 4.4: Detecção de erro de comparação através da validação

Outros tipos de erros também podem ser detectados durante a validação. Caso durante a especificação do sistema, um sinal declarado não seja tratado pelo processo no estado em que ele se encontra, a validação informa a ocorrência de um erro de "consumação implícita de sinal"(*implicit signal consumption*) indicando que a especificação do processo deve ser revista. A figura 4.5 mostra o erro de consumação implícita de sinal gerado pela ferramenta *Validator*, reportando que o sinal *rmce-route* enviado pela instância *ce* do processo *pstatic-ce* (figura

3.6) não foi consumido pela instância *pe*. A figura 4.6 ilustra o diagrama *MSC* deste erro de consumação implícita de sinal, onde o sinal *rmce-route* enviado pela instância *ce* não foi consumido pela instância *pe*(figura 3.8).

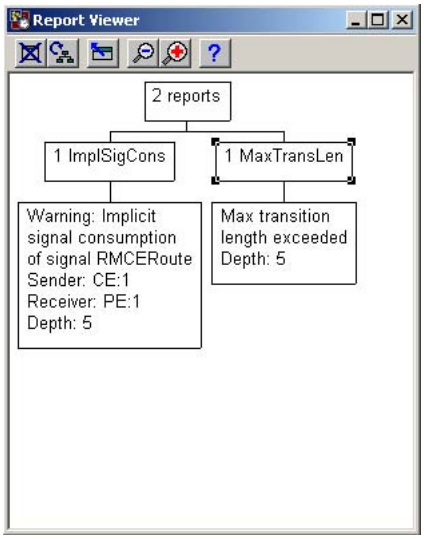


Figura 4.5: Notificação de erro de consumo de sinal implícito durante a validação

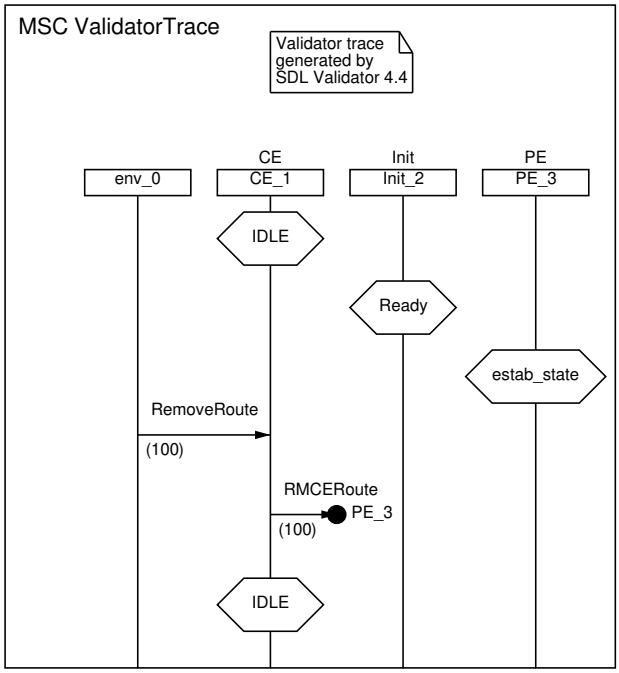


Figura 4.6: MSC gerado com erro de consumo de sinal implícito

Um outro erro bastante comum, detectado pela validação, é o envio de sinais (ou chamadas de procedimentos exportados) a processos que não existam mais. Caso uma referência a um processo que já foi destruído continue sendo armazenada em outro processo, um sinal pode ser enviado para uma referência (*Pid*) inexistente. Para evitar isso, sempre que um processo é destruído todos os processos que possam possuir referências para ele devem ser notificados.

Após a correção dos erros detectados pelo processo de validação, o sistema ainda pode ser submetido a testes mais detalhados visando a verificação da existência de erros de lógica, que acarretem no funcionamento inadequado de algum procedimento especificado.

Pode-se então utilizar do recurso de simulação para a realização de testes de eventuais casos críticos e das principais funcionalidades do sistema, analisando-se de forma mais criteriosa cada passo de sua execução.

4.2 Simulações

Através das simulações pode-se verificar as diversas funcionalidades dos sistemas especificados em *SDL*, na tentativa de encontrar algum erro de lógica que tenha passado despercebido durante o processo de especificação. Através do uso de *MSCs* (*Message Sequence Charts*)[20] é possível visualizar de forma gráfica o resultado do processo de simulação, permitindo uma visão bastante detalhada do comportamento do sistema.

No capítulo 3 foram propostos dois sistemas especificados em *SDL* que envolvem a arquitetura *vpn-mpls*. O primeiro, representado pelo sistema *basicarquitecture-scenario1* (figura 3.1) apresenta uma arquitetura padrão de *vpn-mpls* onde a troca de informações de sinalização entre o cliente e o provedor de serviço ocorre através do estabelecimento de rotas estáticas entre o roteador *ce* e o roteador *pe*, não ocorrendo troca de informações de qualidade de serviço. O segundo, representado pelo sistema *basicarquitecture-scenario2* (figura 3.12) é a proposta deste trabalho de tese, onde a troca de informações de sinalização entre o cliente *vpn* e o provedor de serviço ocorre através do estabelecimento de sessões *mp-bgp* entre o roteador *ce* e o roteador *pe*, havendo assim a modificação do protocolo *mp-bgp* para troca de informações de qualidade de serviço.

Baseado nos sistemas propostos, são apresentadas algumas simulações de casos críticos considerados de maior importância e erros detectados durante o processo de simulação e posteriormente corrigidos, a partir de gráficos *MSC*.

A figura 4.7 apresenta o *MSC* gerado durante a simulação da tentativa de estabelecimento de uma sessão *mp-bgp* para troca de informações dos clientes *vpn* entre os roteadores *pe* do provedor. Neste exemplo a sessão *mp-bgp* não é estabelecida pois o roteador *pe*(processo *pe*) ao receber o sinal *open* verifica que o roteador que originou este sinal não pertence ao seu sistema autônomo, e a tentativa de estabelecimento da sessão *mp-bgp* é encerrada através do envio do sinal *notification* pelo roteador *pe*, informando a notificação ocorrida. O *MSC* é composto pelos processos *env-0*, *ce*, *init* e *pe* que correspondem às instâncias do sistema *basicarquitecture-scenario1* (figura 3.2). Nesta simulação o roteador *pe* (processo *pe*) tenta estabelecer uma sessão *mp-bgp* com outro roteador *pe* que é representado pelo ambiente externo(*env-0*). Esta sessão é inicializada com o roteador *pe* enviando o sinal *tcp* com a identificação do seu endereço (2).

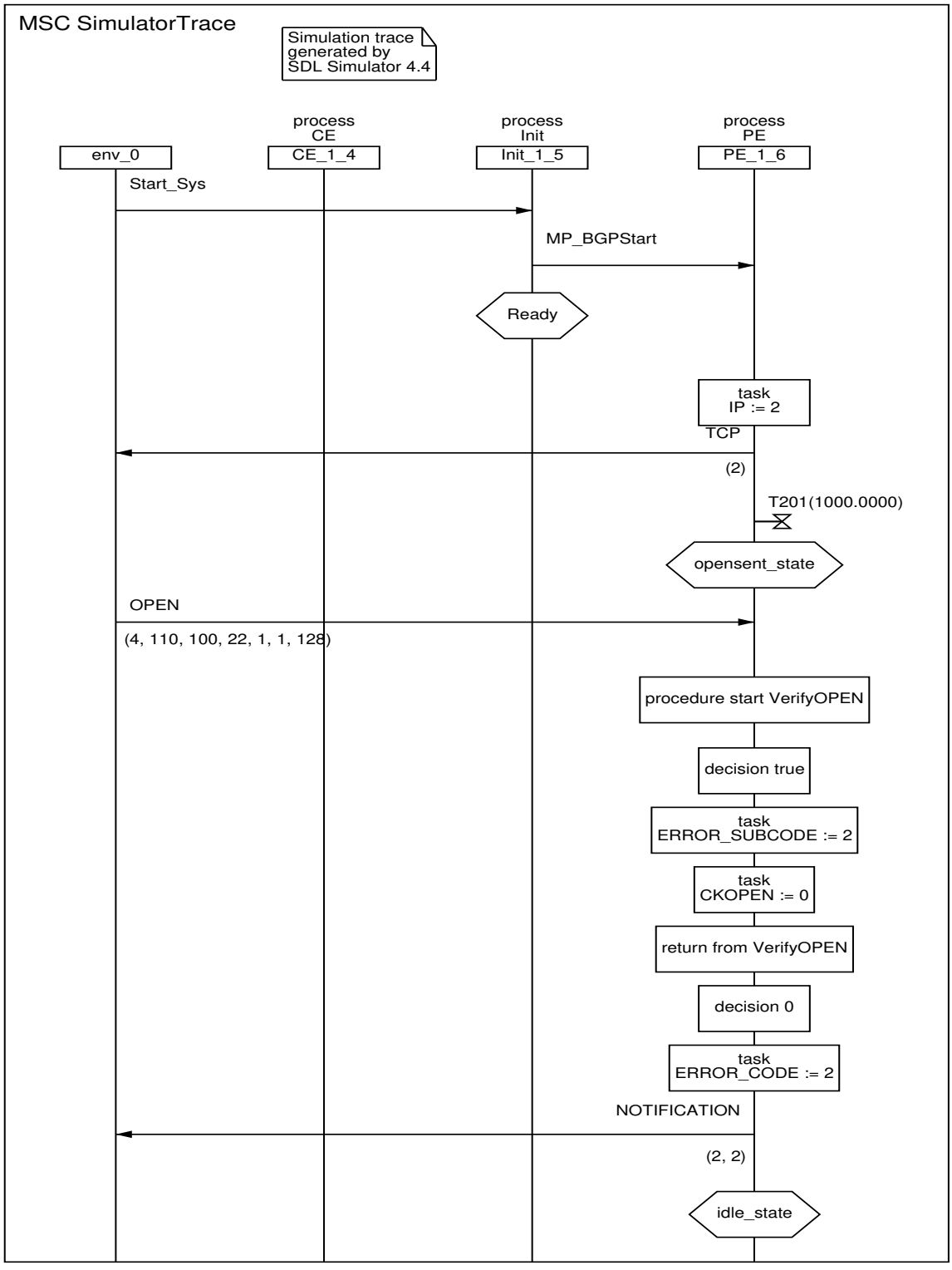


Figura 4.7: Número de sistema autônomo não suportado (ver figura 3.10)

De acordo com a maquina de estado do *mp-bgp*, descrita na seção 2.6.6, ao enviar o sinal *tcp* o roteador *pe* aguarda o recebimento do sinal *open*, onde ocorre a tentativa de abertura da sessão *mp-bgp*, caso todos os parâmetros estejam em conformidade. Neste exemplo o roteador *pe* (processo *pe*) ao receber o sinal *open* do ambiente externo (roteador *pe* da rede do provedor de serviço) não suporta o valor do número do sistema autônomo contido no sinal *open*(*asn* - *autonomous system number*), enviando assim um sinal de notificação (sinal *notification*). Os valores (2,2) enviados pelo sinal *notification* representam o código da notificação(2 - *open message error*) e o sub-código da notificação (2 - *bad peer as*) segundo [41]. De acordo com o caso crítico ilustrado na figura 4.7, roteadores *pe* só devem trocar informação de clientes *vpn* com roteadores *pe* que participem do mesmo sistema autônomo, garantindo assim a segurança dos dados do cliente *vpn* [28].

A figura 4.8 apresenta o *MSC* gerado durante a simulação de manutenção da sessão *mp-bgp* entre os roteadores *pe* através do sinal *keepalive*. O roteador *pe* representado pelo processo *pe*, inicia o estabelecimento de uma sessão *mp-bgp* através do envio do sinal *tcp*. Ao receber o sinal *tcp*, o ambiente externo (*env-0*) envia um sinal *open* para abertura da sessão *mp-bgp*. Ao receber o sinal *open* o roteador *pe* (processo *pe*) verifica se todos os campos deste sinal estão em conformidade. Como nesta simulação de caso crítico todos os campos do sinal *open* estão em conformidade, o processo *pe* envia o sinal *keepalive* para manutenção da sessão *mp-bgp*. Esta sessão *mp-bgp* estabelecida é uma sessão do tipo interna (mp-ibgp), ou seja, uma sessão estabelecida entre roteadores *pe* da rede do provedor de serviço.

A figura 4.9 apresenta o *MSC* gerado durante a simulação de inserção de rota estática pelo ambiente externo nos roteadores *ce* (processo *ce*) e *pe* (processo *pe*), para o sistema *basicarquitecture-scenario1* (figura 3.2). De acordo com a figura 4.9, o ambiente externo (*env-0*) insere através do sinal *add-route* a rota 555 no processo *ce*. Este sinal é trocado com dois parâmetros (555,0), onde o primeiro representa a identificação da rota e o segundo um valor nulo que só será utilizado pelo sistema *basicarquitecture-scenario2*(figura 3.13) como campo para troca de informação de prioridades. Ao receber o sinal *add-route*, o processo *ce* insere a rota na sua tabela de roteamento através da chamada ao procedimento *rt-install*(figura 3.7). Após a instalação da rota, o processo *ce* re-encaminha esta informação para o processo *pe* via sinal

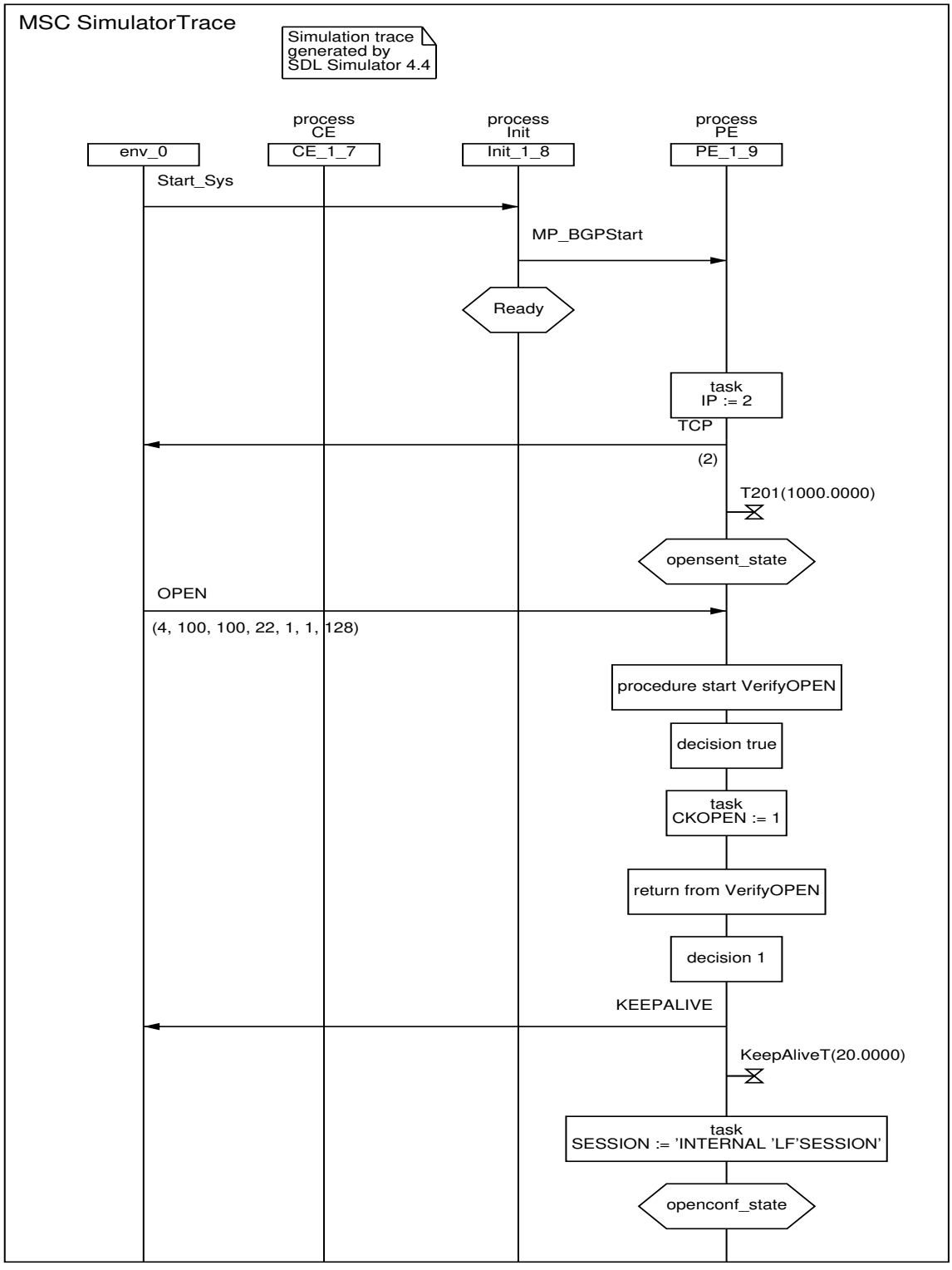


Figura 4.8: Manutenção da sessão *mp-bgp* (ver figura 3.10)

ce-route para que o processo *pe* possa instalar a rota na sua tabela de roteamento, via chamada de procedimento *vrf-install*. Ao adicionar a nova rota, o processo *pe* propaga ao ambiente externo via sessão *mp-bgp*, através do sinal *update*, a informação desta nova rota adicionada na tabela *vrf*. A fim de verificar a informação de rota inserida no roteador *pe* (processo *pe*), o ambiente externo via sinal *data-mpls*, envia um sinal de dados da rede 666 para a rede 555 ligada ao roteador *pe*(processo *pe*). O sinal de dados *data-mpls* é uma mensagem encapsulada na rede *mpls* do provedor com o rótulo 22 representando a rede 555 ligado ao roteador *pe*. Com a chegada do sinal *data-mpls* o roteador *pe* verifica o rótulo *mpls* (rótulo 22 que representa a conexão da rede 555 do cliente *vpn*) e encaminha o pacote de dados (sinal *data-ip*) para o processo *ce* do cliente, que re-encaminha para a rede 555 (sinal *data*).

A figura 4.10 descreve o *MSC* gerado pela especificação do sistema *basicarquitecture-scenario2* (figura 3.12) para troca de informações de roteamento e qualidade de serviço, através do envio do sinal *update* pelo processo *ce*. No sistema *basicarquitecture-scenario2*, além do roteador do cliente (roteador *ce*) ser capaz de trocar informações de roteamento com o roteador do provedor (roteador *pe*), o cliente pode fazer uma escolha em tempo real dos parâmetros de qualidade de serviço que certo tráfego deve possuir ao entrar na rede do provedor. Esta troca de informações de roteamento e de parâmetros de qualidade de serviço ocorre durante a troca de sinais *update* implementada pela sessão *mp-bgp* entre o roteador *ce* (processo *ce*) e o roteador *pe* (processo *pe*). De acordo com a figura 4.10, a partir do momento em que uma nova rota é adicionada a rede do cliente e o cliente informa que esta rede deve possuir prioridade 5 (sinal add-route(555,5)), o roteador *ce* (processo *ce*) insere a rota e o valor de prioridade na sua tabela de roteamento através da chamada ao procedimento *insert-route*. Após a instalação da rota no processo *ce*, a sessão *mp-bgp* encarrega-se de informar ao processo *pe*, via sinal *update*, à existência desta nova rota(rota 555). Junto com a informação da nova rota o sinal *update* também se encarrega de transportar a informação de prioridade sugerida pelo cliente (prioridade 5) para a rede adicionada e o valor do número do sistema autônomo do processo *ce*(65000). Ao incluir esta nova rota na tabela *vrf* via chamada ao procedimento *vrf-install* (figura 3.19), o processo *pe* informa a nova rota aos roteadores *pe* que participam da *VPN* via sinal *update* enviada ao ambiente externo (*env-0*). Com isso o sinal de trafego de dados (sinal *data(555,666)*) que tem

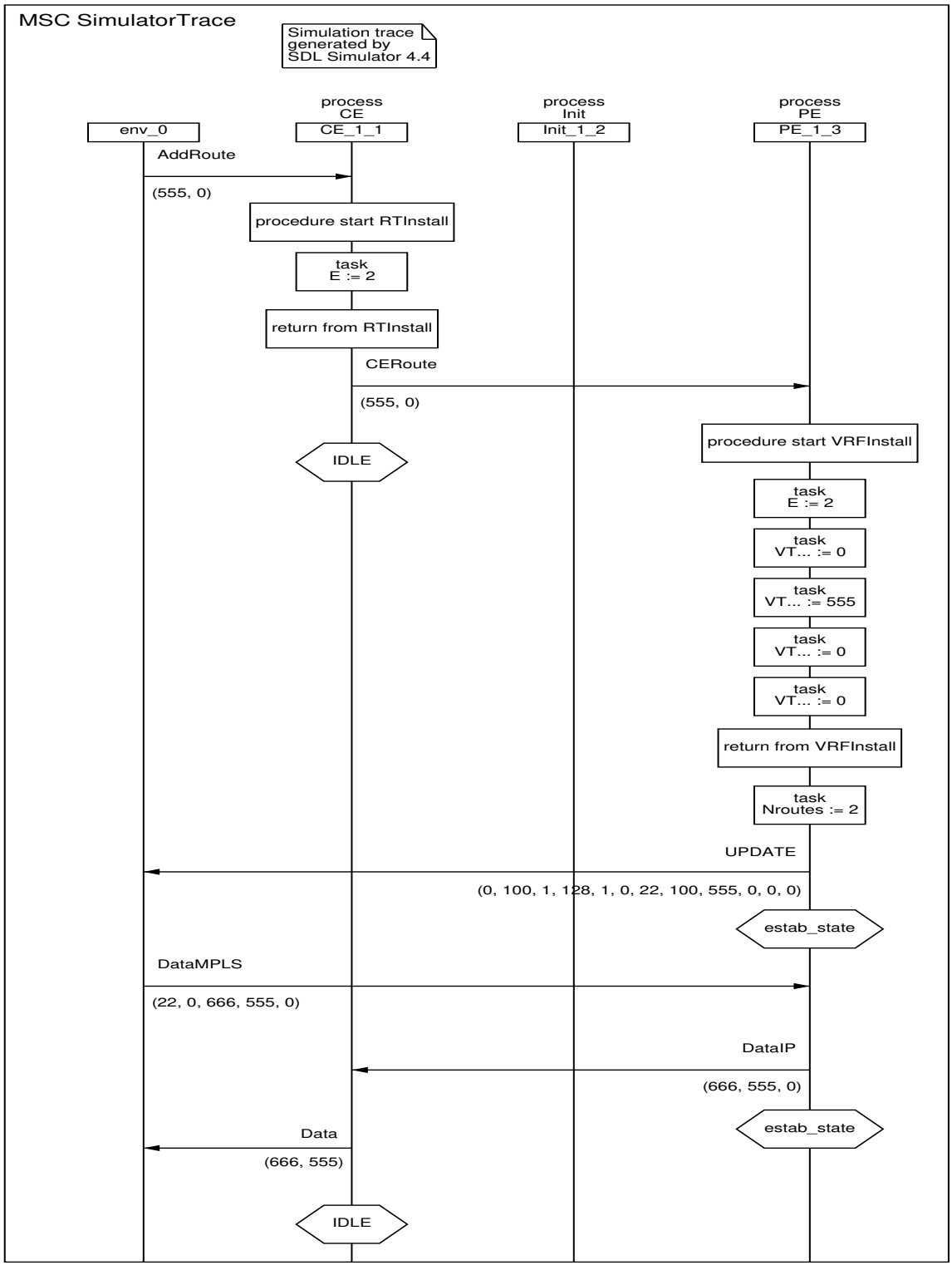


Figura 4.9: Inserção de rota estática, sistema *basicarquitecture-scenario1* (figura 3.2)

como endereço de origem a rede 555 ligado ao roteador *ce* (processo *ce*) e endereço destino a rede 666 localizada no ambiente externo (*env-0*)terá prioridade 5 no enlace entre o roteador *ce* e o roteador *pe* (sinal *data-ip(555,666,5)*) e prioridade 5 mapeada no rótulo *mpls* da rede do provedor de serviço(sinal *data-mpls*), garantindo uma qualidade de serviço fim-a-fim. Podendo o cliente variar estes parâmetros de prioridade do tráfego em tempo real, através da troca de sinais *update*.

A figura 4.11 descreve o *MSC* gerado para troca de informações de roteamento e qualidade de serviço, através do envio do sinal *update* pelo processo *pe*. De acordo com a figura 4.11, a partir do momento em que uma nova rota com seu valor de prioridade é informado pelo ambiente externo (*env-0*) ao processo *pe* através do sinal *update*, o processo *pe* insere esta nova rota através da chamada ao procedimento *vrf-install*. Após a instalação da rota no processo *pe*, a sessão *mp-bgp* encarrega-se de informar ao processo *ce*, via sinal *update*, à existência desta nova rota(rota 999) junto com seu valor de prioridade (6) e o número do sistema autônomo do processo *pe* (100), continuando o processo no estado *established*(figura 3.19). Ao receber o sinal *update*, o processo *ce* insere esta nova informação de rota e seu valor de prioridade na sua tabela de roteamento via procedimento *insert-route*. Com o recebimento do sinal de tráfego de dados (*data(555,999)*), o processo *ce* faz uma chamada ao procedimento *verify-route* a fim de verificar a rota e o valor de prioridade para aquela rota. Após a chamada ao procedimento, o processo *ce* envia o tráfego de dados(sinal *data-ip(555,999,6)*) para o processo *pe*, que mapeia o valor de prioridade sugerida pelo cliente *vpn* (6) no rótulo *mpls* da rede do provedor de serviço(sinal *data-mpls*).

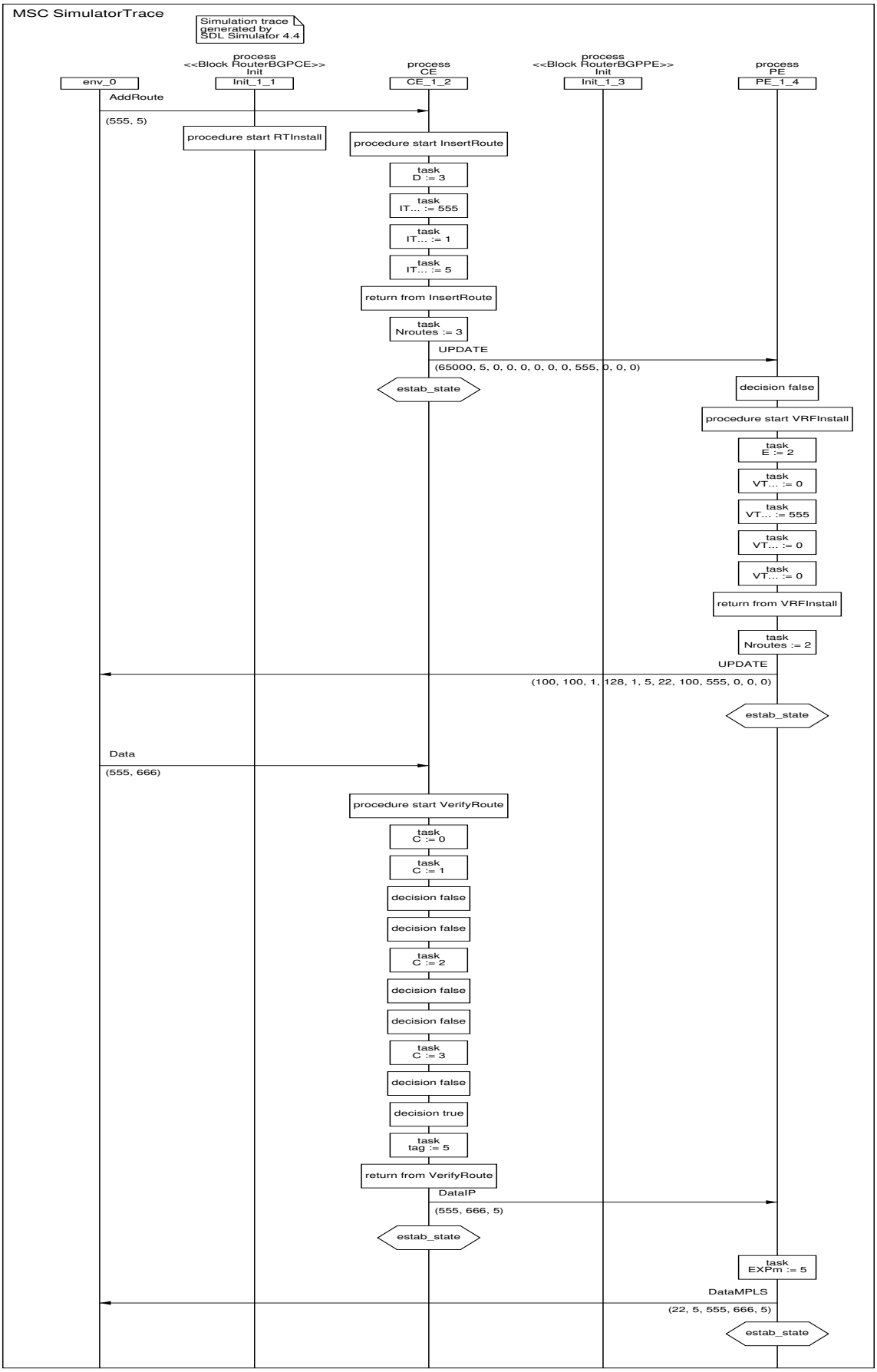


Figura 4.10: Inserção de rota via roteador *ce*, sistema *basicarquitecture-scenario2*(figura 3.13)

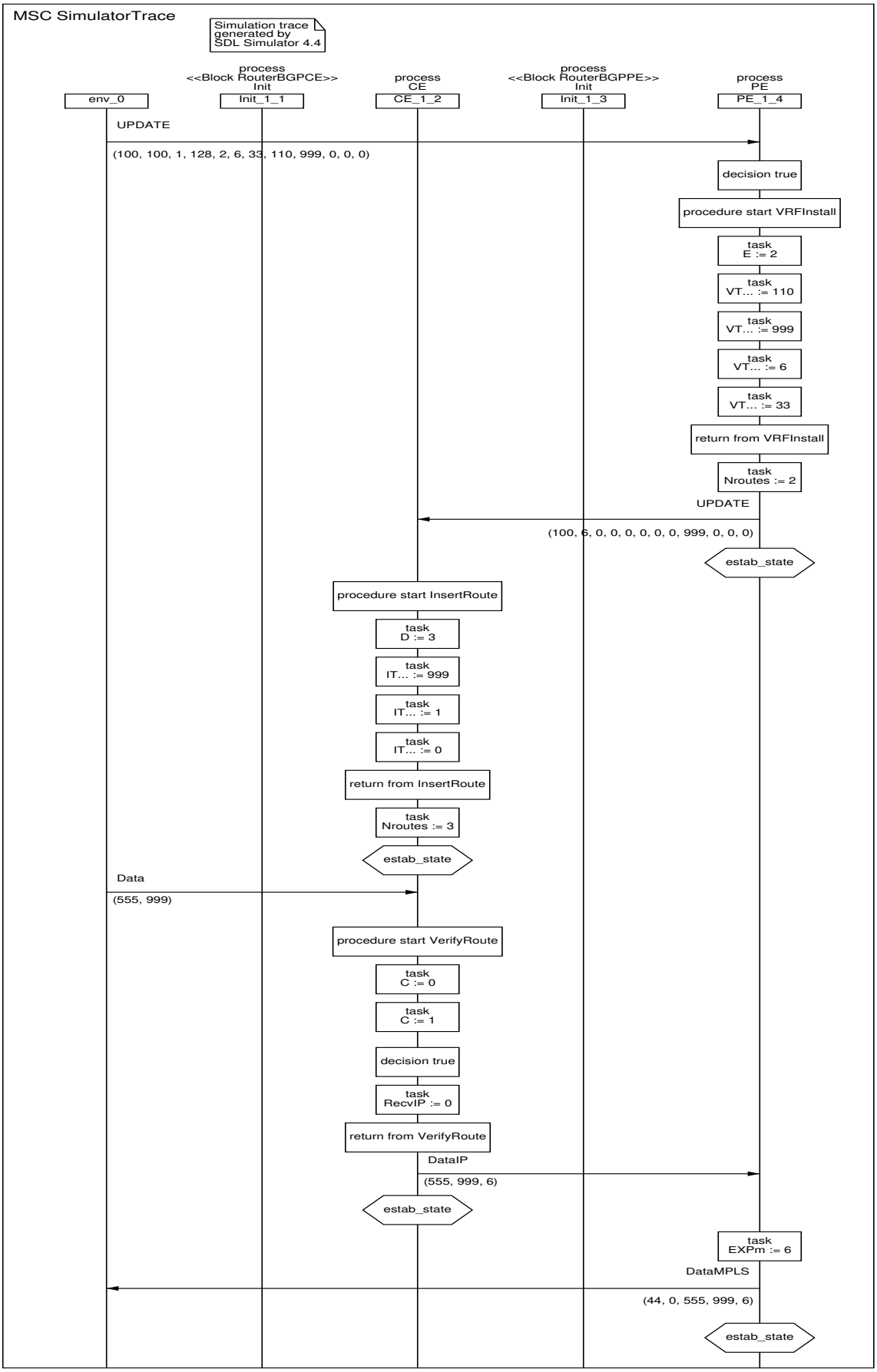


Figura 4.11: Inserção de rota via roteador *pe*, sistema *basicarquitecture-scenario2*(figura 3.13)

5

Análise de desempenho em *vpn-mpls* com qualidade de serviço

5.1 Introdução

A grande demanda por novos serviços que ofereçam informação e comunicação de uma forma integrada e com qualidade de serviço está levando os cientistas, fabricantes e operadoras a repensarem os conceitos das redes e forma de operação das tecnologias atuais. A motivação para essa mudança de paradigma é a necessidade por convergência. A visão é que diferentes tipos de redes com diferentes tecnologias de controle e de transporte deverão convergir para uma rede multi-serviços que será capaz de oferecer diferentes tipos de serviços com diferentes qualidades, usando plataformas abertas e interoperáveis. Nesse contexto, esse capítulo apresenta diversas

simulações do transporte de diferentes serviços como voz, vídeo e dados, sobre uma infra-estrutura de provedor baseado na arquitetura *vpn-mpls*.

São apresentados e comparados os resultados de simulações de cenários com implementação de qualidade de serviço na arquitetura *vpn-mpls* de provedor. A finalidade deste estudo é analisar e comparar o desempenho das propostas de qualidade de serviço da arquitetura *vpn-mpls* [12]. É importante ressaltar que os cenários simulados visam validar a proposta de expansão da arquitetura *vpn-mpls* descrita no capítulo 3, através do uso do mapeamento da arquitetura de serviços diferenciados nos rótulos *mpls* da rede do provedor de serviço.

Neste capítulo, a seção 5.2 apresenta os cenários de rede simulados através da descrição da topologia de rede, aplicações utilizadas e os detalhes de cada um dos cenários simulados. A seção 5.3 apresenta os resultados de desempenho de rede obtidos para cada cenário. A comparação efetuada entre os cenários é realizada em função da vazão, atraso fim-a-fim de pacotes e taxa de perda de pacotes. As simulações são realizadas no simulador de redes *Opnet Modeler* [3].

5.2 Cenários de simulação com o *Opnet Modeler*

Esta seção descreve os três cenários de simulação usados para a análise de desempenho das propostas de qualidade de serviço. Os cenários representam conexões de clientes *vpn* ao provedor de serviço que implementa a arquitetura *vpn-mpls*. Nestes cenários, comparações são realizadas a partir dos resultados de desempenho das aplicações multimídia usadas pelos clientes *vpn* que compartilham a rede do provedor de serviço. Os cenários fazem uso da mesma topologia de rede, mas distinguem-se basicamente pelo tipo de implementação de qualidade de serviço utilizada, e são descritos como:

- *Cenário de melhor esforço*: também conhecido como cenário *best effort*, representa o cenário da arquitetura *vpn-mpls* sem nenhuma implementação de qualidade de serviço.
- *Cenário com combinação de serviços diferenciados e mpls*: expansão do cenário de melhor esforço com a implementação do mapeamento entre as classes de serviço da arquitetura de serviços diferenciados nos rótulos *mpls* da rede do provedor.

- *Cenário com engenharia de tráfego*: expansão do cenário com combinação de serviços diferenciados e *mpls*, onde técnicas de engenharia de tráfego são aplicadas.

A análise de desempenho de cada uma das estratégias de qualidade de serviço implementadas baseou-se nos resultados de vazão (bits/segundo), atraso (segundos) e taxa de perda de pacotes (pacotes/segundo). Todos os cenários analisados correspondem a um "tempo real"de 3600 segundos (1 hora). Por tempo real entenda não o tempo gasto para simulação (ou "tempo de simulação"), mas o tempo que esta simulação representou (ou "tempo simulado").

Considerando-se que a implementação de qualidade de serviço na arquitetura *vpn-mpls* visa garantir um nível de serviço ao cliente *vpn*, a topologia escolhida para os três cenários de simulação experimentados procura explorar a rede com "gargalos"no *backbone* do provedor de serviço. Os três cenários de simulação fazem uso dos mesmos elementos de rede, ou seja, da mesma topologia de rede simulada. A diferença entre os cenários está nos parâmetros de qualidade de serviço implementados nos elementos de rede. Sendo assim, primeiramente é apresentada a topologia da rede utilizada pelos cenários de simulação e logo em seguida são apresentados os cenários de simulação propostos.

Topologia da rede simulada no *Opnet*

A figura 5.1 ilustra a topologia da rede simulada no *Opnet*. Esta topologia visa representar um provedor de serviço com dois clientes *vpn* conectados. Na topologia da rede (figura 5.1), cada cliente *vpn*, denominados *cliente-1* e *cliente-2*, possue dois sites conectados nas bordas da rede *mpls* do provedor de serviço que implementa a arquitetura *vpn-mpls*.

A construção da topologia da rede para os três cenários descritos neste capítulo foi baseada em um cenário real de laboratório da fundação *CPqD*[1]. Esta topologia visa descrever em ambiente de simulação o cenário do laboratório *pa-ngn* (*Laboratório de pesquisa aplicada em redes de próxima geração*) pertencente ao projeto de pesquisa *ngn* (*Next Generation Network*) realizado nos anos de 2001 à 2003. O principal objetivo da criação destes cenários foi a análise,

[1]Fundação *CPqD* : Centro de Pesquisa e Desenvolvimento em Telecomunicações, Rod. Campinas–Mogi-Mirim(SP-340), Km 118.5, Campinas-SP, Brasil

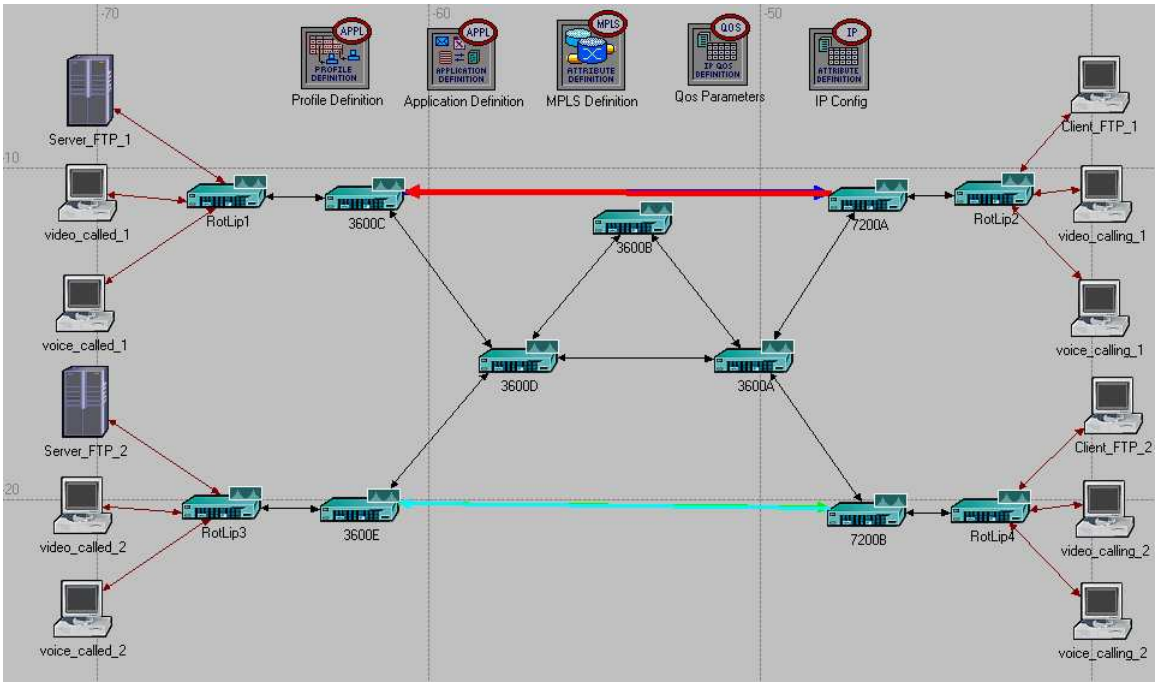


Figura 5.1: Topologia da rede simulada no *Opnet*

do ponto de vista de desempenho, dos parâmetros de qualidade de serviço possíveis de serem implementados na arquitetura *vpn-mpls*. Este trabalho resultou no artigo [31].

Na figura 5.1, os elementos nomeados *profile definition* e *application definition* são objetos especiais do simulador *Opnet* utilizados respectivamente para definição das características de cada elemento utilizado na topologia da rede e definição das aplicações a serem utilizadas nas simulações. O elemento *mpls definition* é utilizado na definição dos parâmetros *mpls* configuráveis pelo simulador nos roteadores que pertencem ao provedor de serviço. Já os elementos *qos parameters* e *ip config* são utilizados para definição dos parâmetros de qualidade de serviço configuráveis nos elementos da rede.

Na topologia da rede simulada, figura 5.1, os roteadores *3600a*, *3600b* e *3600d* representam os roteadores de núcleo do provedor de serviço (roteadores *p* da arquitetura *vpn-mpls*). Os roteadores *3600c*, *3600e*, *7200a* e *7200b* representam os roteadores de borda do provedor de serviço (roteadores *pe*). Os roteadores *rotlip1*, *rotlip2*, *rotlip3* e *rotlip4* são roteadores dos clientes *vpn* (roteadores *ce*). Os roteadores são nomeados com as iniciais *3600*, *7200* e *rotlip* pois pertencem a família de roteadores *cisco* [1] do modelo de fabricação *3600*, *7200* e *2600*

respectivamente, sendo esta a nomenclatura usada para distinguir os roteadores no laboratório *pa-ngn* do *CPqD*.

Os servidores e estações utilizados na construção da topologia(figura 5.1) são elementos de rede dos clientes *vpn*(*cliente-1* e *cliente-2*). Os servidores *server-ftp1* e *server-ftp2* conectados aos roteadores *rotlip1* e *rotlip3* são servidores *ftp* responsáveis por gerar tráfego *ftp* para os clientes, *client-ftp1* e *client-ftp2*, que estão conectados aos roteadores *rotlip2* e *rotlip4* localizados na outra extremidade da rede do provedor de serviço. As estações denominadas *video-called1* e *video-called2* são estações de video que estabelecem comunicações com as estações de video, *video-calling1* e *video-calling2*, localizadas na outra extremidade da rede. As estações *voice-called1* e *voice-called2* são estações que estabelecem trafegos de voz com as estações *voice-calling1* e *voice-calling2*, também localizadas na outra extremidade da rede.

Todos os enlaces da topologia da rede simulada (figura 5.1) são *full duplex* e configurados da seguinte forma:

- Os enlaces entre os roteadores *3600a*, *3600b* e *3600d*, que representam o núcleo da rede do provedor de serviços (roteadores *p*), são de 6.5 Mbps.
- Os demais enlaces entre os roteadores, que representam a borda da rede e a rede dos clientes *vpn* (roteadores *pe* e roteadores *ce*), são de 10 Mbps.
- Os enlaces entre os servidores de *ftp*, clientes *ftp* e estações de *vídeo* e de voz, que estão ligados aos roteadores do cliente (*rotlip1*, *rotlip2*, *rotlip3* e *rotlip4*) são configurados como *10base-t*, ou seja, 10 Mbps.

Configuração das aplicações de *ftp*, *vídeo* e *voip*

Com o intuito de simular cenários reais, onde aplicações não prioritárias compartilham a rede com aplicações prioritárias, foram definidas três aplicações; aplicação de transferência de arquivo ou *ftp* (*file transfer protocol*), aplicação de *vídeo* (*vídeo conferência*) e aplicação de voz sobre *IP* (*voip*) para cada cliente *vpn*. Estas três aplicações de tráfego utilizadas visam modelar o tráfego multimídia usado pelos clientes *vpn*. As características de cada aplicação são

modeladas através dos parâmetros configuráveis pelo simulador *Opnet*, e são apresentadas em forma de tabelas.

$$E(x) = \frac{c.a}{(c-1)} \ \{Location \rightarrow a > 0, Shape \rightarrow c > 1 \tag{5.1}$$

A aplicação *ftp* está descrita na tabela 5.1. O valor *command mix(get/total)* em 100% representa um tráfego *ftp* do tipo 100% *get*, ou seja, um tráfego *ftp* gerado no sentido dos servidores *ftp* (*server-ftp1* e *server-ftp2*) para os clientes *ftp* (*client-ftp1* e *client-ftp2*). A aplicação *ftp* foi definida como uma distribuição do tipo *Pareto* [11], e o cálculo dos valores da distribuição de *Pareto* utilizada são obtidos através da equação 5.1, mostrada pela tabela 5.1 na forma *pareto(a,c)*. As variáveis de tempo de requisição (*inter-request time*) e tamanho de pacote (*file size*) são configuradas para gerar um tráfego *ftp* a uma taxa de 2 Mbps para cada cliente. O parâmetro *type of service* é usado para marcação de prioridade da aplicação, que neste caso está configurado como melhor esforço (*best effort*) representado pelo valor 0(zero).

parâmetros da aplicação *ftp*	
Parâmetros de configuração	Valores
Command Mix (Get/Total)	*100%*
Inter-Request Time (sec)	*exponential(1)*
File Size (bytes)	*pareto(83333.33, 1.5)*
Type of Service	*best effort* (0)

Tabela 5.1: Configuração dos parâmetros da aplicação ftp

A aplicação de *vídeo* definida na tabela 5.2 estabelece um tráfego entre as estações *video-called* e *video-calling* da figura 5.1. Os parâmetros configuráveis para o tráfego de *vídeo* são os valores de tempo de chegada de pacotes (*frame interarrival time*), expressado por um valor constante, e tamanho do pacote (*frame size*), que é uma variável aleatória com distribuição exponencial. Os valores atribuídos à estas duas variáveis da tabela 5.2 geram uma aplicação de *vídeo* com taxa de 1,5 Mbps para cada cliente *vpn*(*cliente-1* e *cliente-2*). É importante ressaltar que de acordo com os parâmetros configurados, a aplicação de *vídeo* é uma aplicação bidirecional, ou seja, um tráfego de *vídeo* com taxa de 1,5 Mbps das estações *video-called* para

video-calling e vice-versa. O parâmetro *type of service* também é configurado e representa a prioridade que será dada a aplicação na rede, que de acordo com a tabela 5.2 é do tipo melhor esforço para o cenário de melhor esforço e será configurado para os demais cenários conforme a tabela de mapeamento de prioridades das aplicações.

parâmetros da aplicação de *vídeo*	
Parâmetros de configuração	Valores
Frame Interarrival Time (sec)	*constant(0.1)*
Frame Size (bytes)	*exponential(15625)*
Type of Service	*best effort* (0)

Tabela 5.2: Configuração dos parâmetros da aplicação de vídeo

A configuração da aplicação de *voip* é apresentada na tabela 5.3 e através dos parâmetros de configuração da tabela, o *Opnet* modela a aplicação de *voip* a ser utilizada na simulação. Os tempos de fala e de silêncio usados para modelar a aplicação de voz são representados pelos parâmetros *talk spurt length* e *silence length* da tabela 5.3. Os parâmetros *encoder scheme* e *voice frame per packet* caracterizam respectivamente o tipo de codificador usado na geração do tráfego de voz e a quantidade de quadros de voz por pacote durante a simulação. De acordo com a tabela 5.3, a aplicação *voip* utiliza a codificação *gsm*, e gera uma taxa de aproximadamente 20 Kbps no sentido da estação *voice-called* para estação *voice-calling* e vice-versa (tráfego bidirecional). O parâmetro *type of service* também é configurado e representa a prioridade que será dada a aplicação na rede, que de acordo com a tabela 5.3 é do tipo melhor esforço.

Cenário de melhor esforço

O cenário de melhor esforço, também conhecido como cenário *best effort*, representa o cenário da arquitetura *vpn-mpls* sem nenhuma implementação de qualidade de serviço. O cenário de melhor esforço funciona como cenário de referência para os outros cenários de simulação, construídos para avaliar o desempenho dos mecanismos de qualidade de serviço na arquitetura *vpn-mpls*. Este cenário, assim como os outros cenários simulados apresentam todos os componentes da arquitetura *vpn-mpls*, como: roteadores *ce* (*rotlip1*, *rotlip2*, *rotlip3* e

parâmetros da aplicação *voip*	
Parâmetros de configuração	Valores
Silence Length (sec)	*exponential(0.65)*
Talk Spurt Length (sec)	*exponential(0.352)*
Encoder Scheme	*GSM (silence)*
Voice Frames per Packet	1
Type of Service	*best effort* (0)

Tabela 5.3: Configuração dos parâmetros da aplicação *voip*

rotlip4), roteadores *pe* (*3600c*, *3600e*, *7200a* e *7200b*) e roteadores *p* (*3600a*, *3600b* e *3600d*), que fazem uso da topologia de rede da figura 5.1.

No cenário de melhor esforço as aplicações multimídias definidas em cada cliente *vpn*; aplicação *ftp*, de *vídeo* e *voip*, compartilham a rede do provedor de serviço. Neste cenário o provedor de serviço não possui nenhum método de garantia de qualidade de serviço, ou seja, as aplicações não recebem nenhum tipo de prioridade ao passarem pela rede do provedor de serviço . As três aplicações são encaminhadas pelo *IP* através do serviço de melhor esforço como pode ser observado no campo tipo de serviço (*type of service*) apresentado nas tabelas 5.1, 5.2 e 5.3.

De acordo com a topologia da rede apresentada na figura 5.1 e as configurações das aplicações *ftp*(tabela 5.1), *vídeo* (tabela 5.2) e *voip* (tabela 5.3), o *cliente-1* gera um tráfego *ftp* de 2.0 Mbps no sentido do servidor *server-ftp1* à estação *client-ftp1*, tráfego de *vídeo* de 1.5 Mbps entre as estações *video-called1* e *video-calling-1*, e tráfego de voz de 20 Kbps entre as estações *voice-called1* e *voice-calling1*, totalizando assim um tráfego de 3.5 Mbps gerado pelo *cliente-1*. Este tráfego total do *cliente-1* entra na rede do provedor de serviço pelo roteador *3600c* com destino ao outro site do *cliente-1*, conectado ao roteador *7200a*.

As aplicações *ftp*, *vídeo* e *voip* do *cliente-2* possuem as mesmas configurações feitas no *cliente-1*. Logo, o *cliente-2* também gera um tráfego total de 3.5 Mbps que entra na rede do provedor de serviço pelo roteador *3600e* com destino ao outro site do *cliente-2*, conectado ao roteador *7200b*. Os tráfegos dos clientes *vpn* totalizam uma carga de tráfego de 7.0 Mbps

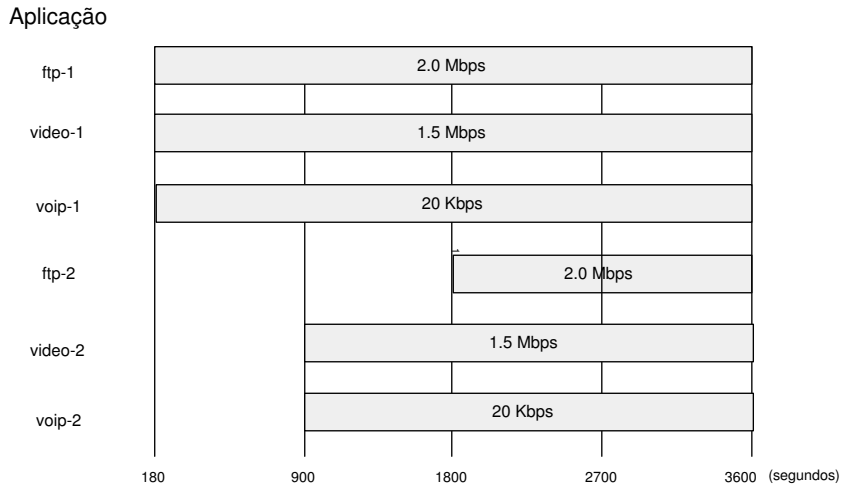


Figura 5.2: Carga média imposta pelas aplicações durante a simulação

imposta ao núcleo do provedor de serviço, representado pelos roteadores *3600a*, *3600b* e *3600d*, que apresentam enlaces de 6.5 Mbps. Como o tráfego gerado é maior que os enlaces entre os roteadores de núcleo, a diferença entre a taxa do enlace entre os roteadores e o tráfego total gerado pelos clientes é descartado nas filas dos roteadores, conforme será visto nos resultados de simulação.

A fim de analisar o impacto de cada aplicação multimidia na rede e o impacto entre os clientes, com uso de priorização de trafego, todas as aplicações são iniciadas durante a simulação em tempos diferentes. A figura 5.2 apresenta em forma de barras, o tempo de início e fim de cada aplicação dos clientes *vpn*. De acordo com a figura 5.2, todas as aplicações do *cliente-1* são iniciadas no tempo 180 segundos, enquanto as aplicações de *vídeo* e *voip* do *cliente-2* são iniciadas no tempo 900 segundos e a aplicação *ftp* do *cliente-2* no tempo 1800 segundos. Através da figura 5.2, pode-se calcular a carga de tráfego imposta à rede em cada intervalo de tempo da simulação. Assim, somando-se as taxas das aplicações presentes em cada intervalo de tempo apresentado na figura 5.2, pode-se observar que os intervalos de tempo - denotados na forma (tempo de início, tempo de fim) - de maior sobrecarga no núcleo da rede do provedor é o seguinte: (1800,3600) com carga igual a 7.0 Mbps. Todos os intervalos da simulação serão observados na análise dos resultados de desempenho de rede para os cenários de simulação.

Cenário com combinação de serviços diferenciados e *mpls*

O objetivo deste cenário é analisar os resultados de desempenho de cada aplicação com a combinação de serviços diferenciados aplicados na rede do cliente, mapeados nos rótulos *mpls* da rede do provedor de serviço. Este cenário usa o cenário de melhor esforço como cenário de referência e faz uso da mesma topologia de rede mostrado na figura 5.1. Neste cenário, as redes dos clientes *vpn* implementam a arquitetura de serviços diferenciados através do uso do campo *type of service* do *IP* para priorização das aplicações.

De acordo com a arquitetura de serviços diferenciados [36], as classes de serviço são classificadas como AF (*Assurance Forwarding*), EF (*Expedited Forwarding*) e BE (*Best Effort*). Dentro da classe AF, quatro sub-classes são padronizadas, indo da mais prioritária AF4x até a menos prioritária AF1x. O valor x nestas sub-classes podem variar de 1 a 3, representando a prioridade dentro de cada sub-classe, sendo a classe AF43 de maior prioridade dentro das sub-classes AF. A classe EF é a classe com o maior índice de prioridade, superando todas as sub-classes AF e a BE, e dentro da arquitetura de serviços diferenciados pode ser utilizada para priorizar o tráfego de sinalização da rede e tráfego de voz. A classe BE, também denominada classe de melhor esforço, é utilizada para tráfegos não prioritários. Neste cenário, a priorização das aplicações no campo *type of service* são configuradas de acordo com a tabela 5.4 para o *cliente-1* e tabela 5.5 para o *cliente-2*.

Aplicação	**Prioridade**
ftp-1	AF21
video-1	AF41
voice-1	EF

Tabela 5.4: Mapeamento de prioridades das aplicações do *cliente-1*

Aplicação	**Prioridade**
ftp-2	AF11
video-2	AF11
voice-2	AF11

Tabela 5.5: Mapeamento de prioridades das aplicações do *cliente-2*

De acordo com a tabela 5.4, a aplicação menos prioritária do *cliente-1* é a aplicação *ftp*, que faz parte da classe AF21. A aplicação de *vídeo* é uma aplicação mais prioritária que a de transferência de arquivos (*ftp*) participando da classe AF41, mas perde em prioridade para a aplicação *voip*, que faz parte da classe EF. Para o *cliente-2*, tabela 5.5, todas as aplicações foram mapeadas na classe AF11 e são menos prioritárias que as aplicações geradas pelo *cliente-*

1(tabela 5.4). O mapeamento das aplicações em classes de serviços da arquitetura de serviços diferenciados não seguem uma regra definida. Este mapeamento pode levar em consideração diversos aspectos, mas é comumente implementado de acordo com a contratação de serviços estabelecida entre o cliente *vpn* e o provedor de serviço. Este tipo de contratação feita pelo cliente ao provedor de serviço é conhecida como acordo de nível de serviço, ou *SLA* (*Service Level Agreement*).

Classes de Serviço	**Peso *wfq***
AF11	5
AF21	10
AF22	10
AF31	15
AF32	15
AF41	25
AF42	25
EF	55

Tabela 5.6: Pesos da fila *wfq* utilizados pelas classes de serviço

A atribuição de classes de serviço às aplicações, propicia uma diferenciação dos tipos de tráfego, permitindo que elementos de rede (roteadores) ofereçam prioridade no tratamento de pacotes de determinadas classes. Além da configuração dos campos tipos de serviço em classes de serviço para as aplicações multimídia dos clientes , descritas nas tabelas 5.4 e 5.5, foi implementado neste cenário esquemas de filas nas interfaces dos roteadores da topologia da rede (figura 5.1) para tratar as classes de serviço, e designado os mapeamentos das classes de serviço no campo *exp* dos rótulos *mpls*.

A implementação do esquema de fila visa tratar as diferentes classes de serviço utilizadas pelas aplicações *ftp*, *vídeo* e *voip* nas interfaces dos roteadores. Em todas as filas dos roteadores foi implementado o mecanismo de fila *wfq* (*weight fair queue*) com os valores de peso para cada aplicação exposto na tabela 5.6. De modo geral, *wfq* é uma disciplina que designa uma fila a cada fluxo [4]. Um peso pode ser dado a cada fila para a diferenciação na prioridade dos fluxos.

Como resultado, a fila *wfq* oferece proteção entre fluxos diferentes, além da garantia de um valor de atraso máximo fim-a-fim. De acordo com a tabela 5.6, os pesos utilizados pelo algoritmo *wfq* na simulação variam de 5 à 55, sendo a classe EF mapeada no peso 55 representando a classe de maior prioridade a ser tratada pelos roteadores. Na tabela 5.6 foram expostos somente os mapeamentos utilizados pela simulação.

Além do mapeamento das aplicações em classes de serviço e da implementação do esquema de fila *wfq* nas interfaces dos roteadores, este cenário implementa o mapeamento das classes de serviço no campo *exp* dos rótulo *mpls*(figura 5.3). Neste cenário de simulação, as classes de serviço da arquitetura de serviços diferenciados são mapeadas no campo *exp* dos rótulos *mpls*, a fim de que as aplicações *ftp*, *vídeo* e *voip* sejam tratadas na rede do provedor de serviço de acordo com o mapeamento das classes realizado na tabela 5.4 e 5.5. Este mapeamento, mostrado na figura 5.3, é implementado em todos os roteadores de borda da rede do provedor de serviço (roteadores *pe*), que de acordo com a topologia de rede da figura 5.1 são os roteadores *3600c*, *3600e*, *7200a* e *7200b*. A figura 5.3 ilustra este mapeamento realizado no simulador *opnet*, através do elemento *mpls definition*, dentro do parâmetro *exp-phb*.

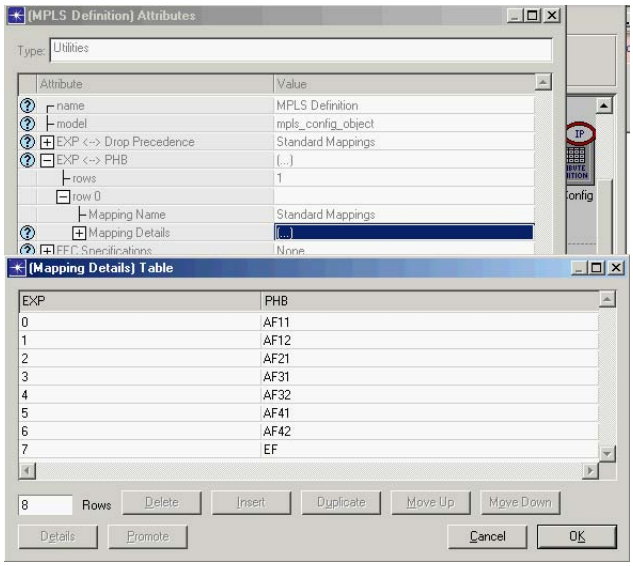


Figura 5.3: Mapeamento das classes de serviço no campo *exp* do rótulo *mpls*

Cenário com engenharia de tráfego

Engenharia de tráfego (TE - *traffic engineering*) é o aspecto das redes *IP* que está ligado à otimização do desempenho da rede [39]. *TE* visa facilitar a operacionalidade da rede, ao mesmo tempo em que é otimizada a utilização dos recursos disponíveis e que são minimizados congestionamentos. O cenário com engenharia de tráfego usa o cenário com combinação de serviços diferenciados e *mpls* como cenário de referência, e tem como objetivo analisar os resultados de desempenho de cada aplicação fazendo uso da engenharia de tráfego na combinação de serviços diferenciados e *mpls*. A topologia de rede usada neste cenário é a mesma dos cenários anteriores (figura 5.1).

A aplicação de engenharia de tráfego neste cenário visa a construção de caminhos comutados por rótulos (*lsp - label switching path*) com restrições mínimas de banda na rede *mpls* do provedor de serviço. Os *lsps* estáticos criados na simulação, que interligam os sites dos clientes *vpn*, passam a possuir restrição mínima de banda com valor de 4 Mbps para estabelecimento do *lsps*. Logo, se a rede do provedor de serviço não disponibilizar este valor mínimo de banda, o *lsp* não é estabelecido entre os sites do cliente *vpn*. Os *lsps* estáticos utilizados pela simulação são representados na figura 5.1 pelas setas horizontais que interligam os roteadores *3600c* e *7200a* para o *cliente-1*, e os roteadores *3600e* e *7200b* para o *cliente-2*.

5.3 Resultados de simulação

Os resultados de simulação apresentados nesta sessão foram obtidos com o uso do simulador *opnet*. Estes resultados apresentam do ponto de vista de desempenho; a vazão da rede, tempo de resposta de *download ftp*, atraso fim-a-fim dos pacotes de *vídeo* e voz, e perda de pacotes, para cada um dos cenários apresentados.

A análise da vazão da rede é realizada no núcleo do provedor de serviço através da monitoração dos enlaces entre os roteadores *3600a*, *3600b* e *3600d*. A figura 5.4 apresenta os resultados de vazão do enlace entre os roteadores *3600d* e *3600a* (parte superior da figura 5.4) e do enlace entre os roteadores *3600d* e *3600b* (parte inferior da figura 5.4) para o cenário de melhor esforço e cenário com combinação de serviços diferenciados e *mpls*. Para estes dois cenários, toda a carga média imposta à rede durante a simulação é transmitida no núcleo da

rede pelo menor caminho (enlace *3600d-3600a*), devido ao uso do algoritmo de caminho mais curto (*spf - shortest path first*) utilizado pelo protocolo *ospf* (*open shortest path*) implementado nos roteadores da rede do provedor de serviço.

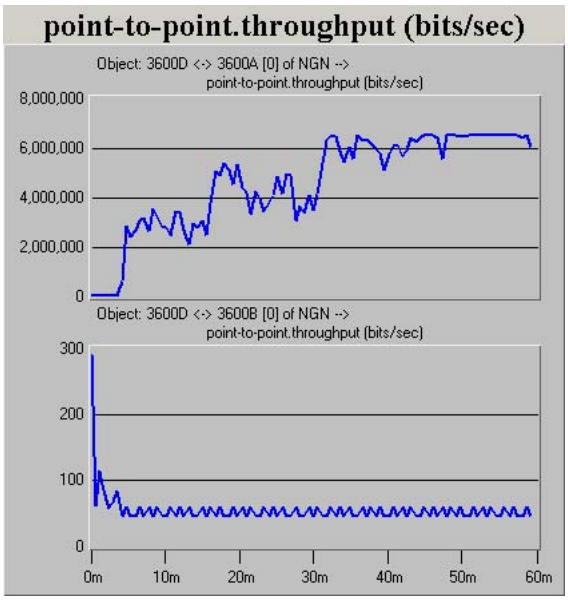


Figura 5.4: Vazão dos enlaces *3600d-3600a* e *3600d-3600b* para o cenário de melhor esforço

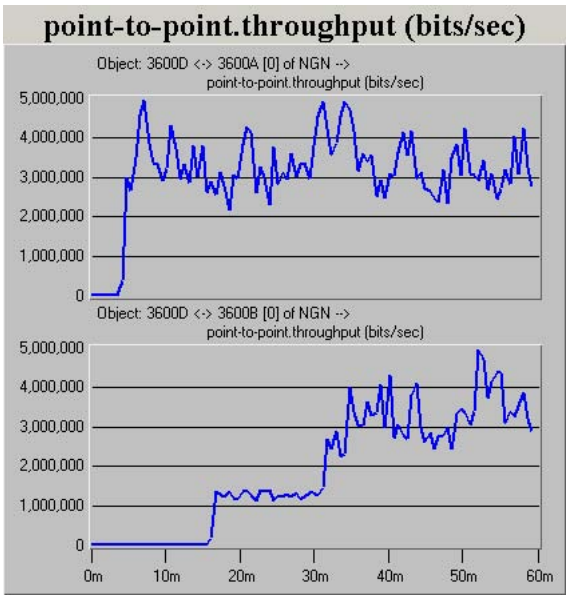


Figura 5.5: Vazão dos enlaces *3600d-3600a* e *3600d-3600b* para o cenário com engenharia de tráfego

Analisando-se o enlace *3600d-3600a* (figura 5.4), verifica-se que o tráfego da rede começa próximo ao início da simulação, dado pelo início das aplicações de *ftp*, *vídeo* e *voip* do *cliente-1* no tempo de 180 segundos, gerando uma carga média neste enlace de 3.5 Mbps. Segundo a figura 5.2, no tempo 15 minutos (900 segundos) as aplicações de *vídeo* e *voip* do *cliente-2* são iniciadas, aumentando a carga média do enlace *3600d-3600a* de 3.5 Mbps para aproximadamente 5 Mbps. No tempo 30 minutos (1800 segundos) a aplicação *ftp* do *cliente-2* é iniciada, o que representa um aumento do uso do enlace *3600d-3600a* de 5 Mbps para 6.5 Mbps, que é a capacidade máxima deste enlace. Analisando-se a parte inferior da figura 5.4, verifica-se que o enlace entre os roteadores *3600d* e *3600b* não é utilizado, apresentando apenas um tráfego pequeno e constante que representa as informações trocadas pelo protocolo *ospf* entre estes roteadores, ou seja, este enlace é sub-utilizado para estes dois cenários de simulação.

Para o cenário com engenharia de tráfego, apresentado pela figura 5.5, o uso da rede do

provedor de serviço é otimizado e garantido pelo parâmetro de banda mínima configurado no estabelecimento dos *lsps*. Neste cenário os *lsps* do *cliente-1* são estabelecidos entre os roteadores *3600c* e *7200a* (figura 5.1) passando pelo enlace *3600d-3600a*, e os *lsps* do *cliente-2* são estabelecidos entre os roteadores *3600e* e *7200b* (figura 5.1) passando pelo enlace *3600d-3600b*, pois o enlace *3600d-3600a* não suporta o estabelecimento dos *lsps* dos dois clientes devido a limitação de banda. Com isso pode-se analisar na parte superior da figura 5.5, que todo o tráfego das aplicações do *cliente-1* gerados desde o início da simulação com carga média de 3.5 Mbps são transmitidos através do enlace *3600d-3600a*. A parte inferior da figura 5.5 apresenta a vazão do enlace *3600d-3600b*, que é de 1.5 Mbps a partir do tempo de 15 minutos (900 segundos) representando as aplicações de *vídeo* e *voip* do *cliente-2*, e aumenta para aproximadamente 4.5 Mbps a partir do tempo de 30 minutos (1800 segundos) com o início da aplicação *ftp* do *cliente-2*.

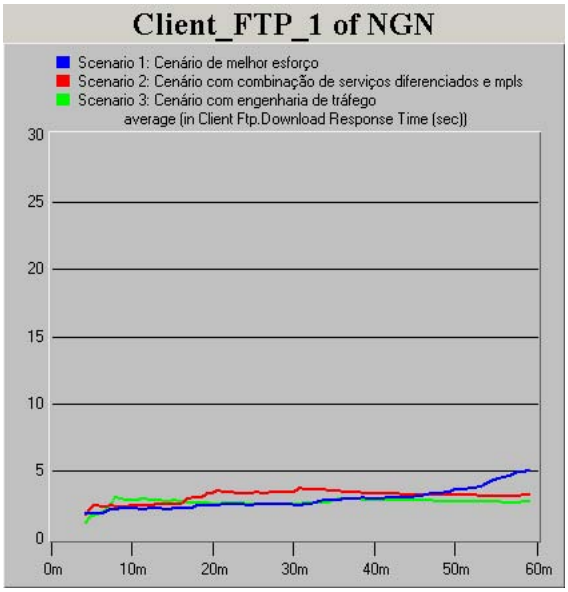


Figura 5.6: Tempo de resposta de *download* do *cliente-1*

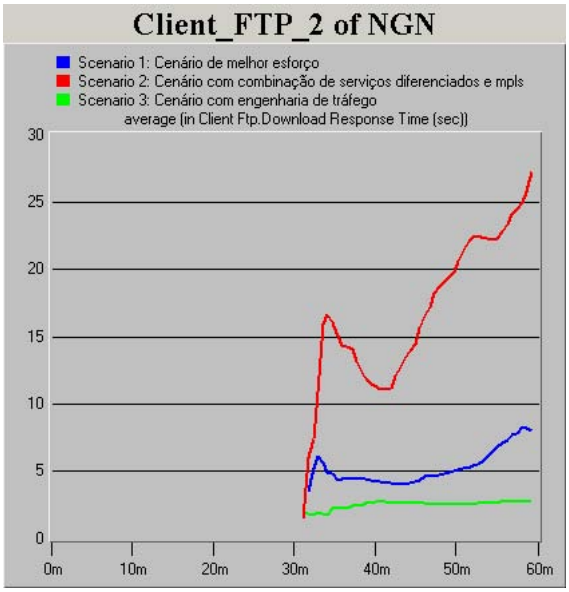


Figura 5.7: Tempo de resposta de *download* do *cliente-2*

A figura 5.6 apresenta os resultados de tempo de resposta de download do *cliente-1* para os três cenários simulados. Para o cenário de melhor esforço todas as aplicações são tratadas pela rede da mesma forma (sem priorização) o valor de tempo de resposta de *download* do *cliente-1* começa a piorar a partir do tempo de 30 minutos devido ao aumento da carga imposta a rede

do provedor pelo início das aplicações do *cliente-2*. Para o cenário com combinação de serviços diferenciados e *mpls*, onde a aplicação *ftp* é mapeada na classe AF21 (tabela 5.4), o valor de tempo de resposta de *download ftp* aumenta um pouco no instante de tempo 15 minutos devido ao inicio das aplicações de *vídeo* e *voip* do *cliente-2*, e a partir deste instante de tempo o valor permanece estável durante o restante da simulação. Já para o cenário com engenharia de tráfego, o tempo de resposta de download do *cliente-1* permanece estável durante toda a simulação, visto que as aplicações do *cliente-2* não disputam o mesmo enlace com as aplicações do *cliente-1*, conforme pode ser visto nos gráficos de vazão da figura 5.5.

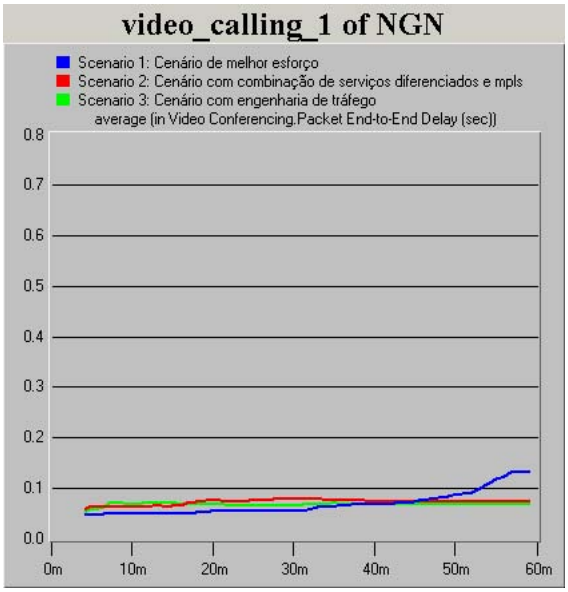


Figura 5.8: Atraso fim-a-fim de pacote de *vídeo* do *cliente-1*

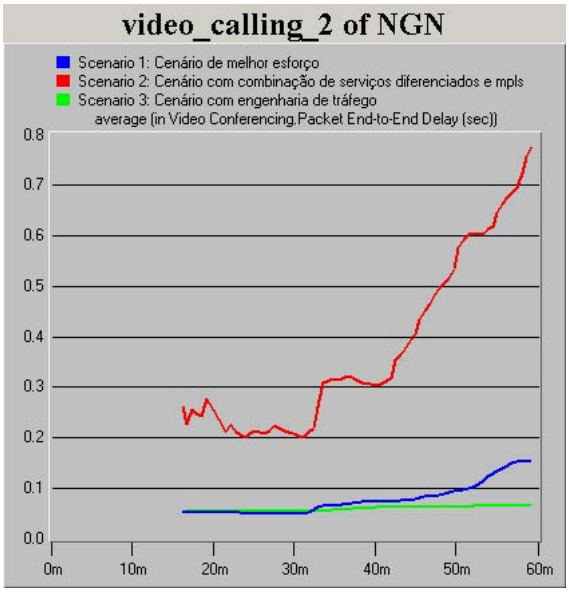


Figura 5.9: Atraso fim-a-fim de pacote de *vídeo* do *cliente-1*

O tempo de resposta de download do *cliente-2* (figura 5.7), apresenta pior resultado para o cenário com combinação de serviços diferenciados e *mpls*, pois a aplicação *ftp* assim como as aplicações de *vídeo* e *voip* do *cliente-2*, pertencem a classe de menor prioridade, classe AF11 (figura 5.5). Para os cenários de melhor esforço e com engenharia de tráfego, os resultados não apresentam grandes variações, visto que para o primeiro cenário a aplicação *ftp* apresenta a mesma prioridade das outras aplicações (melhor esforço) e para o cenário com *TE* o enlace utilizado é outro (enlace *3600d-3600b*, figura 5.5).

A figura 5.8 apresenta os resultados de atraso fim-a-fim da aplicação de *vídeo* do *cliente-*

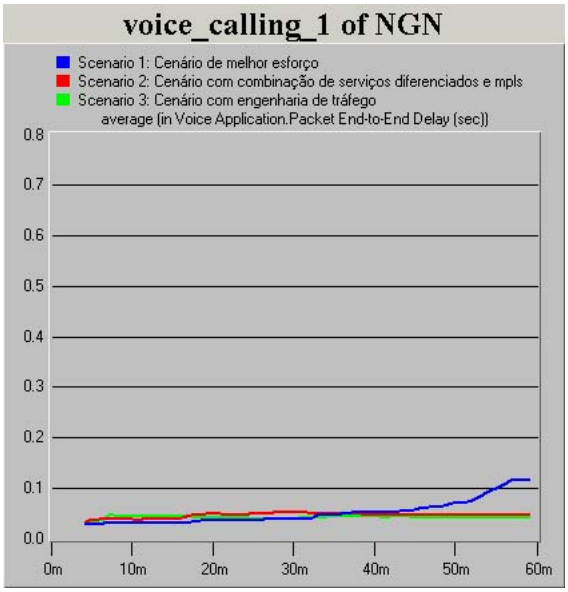


Figura 5.10: Atraso fim-a-fim de pacote de voz do *cliente-1*

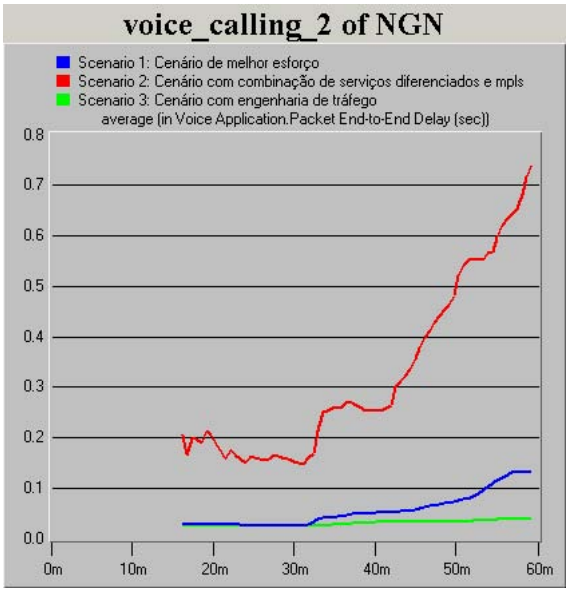


Figura 5.11: Atraso fim-a-fim de pacote de voz do *cliente-2*

1 para os três cenários simulados. Para o cenário de melhor esforço todas as aplicações são tratadas pela rede da mesma forma, o valor de atraso fim-a-fim de *vídeo*(figura 5.8), assim como o valor de atraso fim-a-fim da aplicação de *voip* (figura 5.10) do *cliente-1*, começam a piorar a partir do tempo de 30 minutos devido ao aumento da carga imposta a rede do provedor pelo início das aplicações do *cliente-2*. Para o cenário com combinação de serviços diferenciados e *mpls*, a aplicação de *vídeo* do *cliente-1* é mapeada na classe AF41 (tabela 5.4) que apresenta uma prioridade alta na rede em comparação com as outras aplicações, perdendo apenas para classe EF de *voip*. Com isso, o valor de atraso fim-a-fim da aplicação de *vídeo* do *cliente-1* para este cenário (figura 5.8) apresenta valores baixo em comparação com os valores de atraso fim-a-fim da aplicação de *vídeo* do *cliente-2* (figura 5.9). O mesmo fato ocorre para o atraso fim-a-fim da aplicação de *voip* do *cliente-1*(figura 5.10), mapeada na classe EF (tabela 5.4), que apresenta resultados melhores em comparação com aplicação de *voip* do *cliente-2*(figura 5.11) mapeada na classe AF11 (tabela 5.5).

No cenário com engenharia de tráfego, os valores de atraso fim-a-fim da aplicação de *vídeo* do *cliente-1* (figura 5.8) e do *cliente-2* (figura 5.9) permancem estáveis durante toda a simulação, visto que as aplicações do *cliente-2* não disputam o mesmo enlace com o *cliente-1*. O mesmo

ocorre para a aplicação *voip* do *cliente-1* (figura 5.10) e do *cliente-2* (figura 5.11).

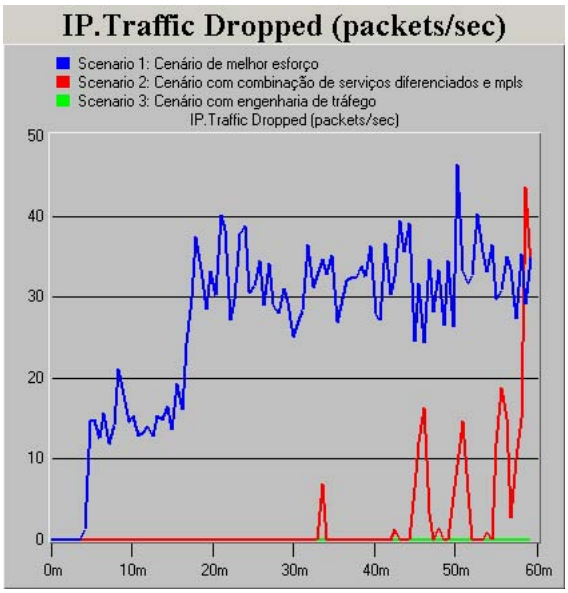


Figura 5.12: Perda de pacotes durante a simulação

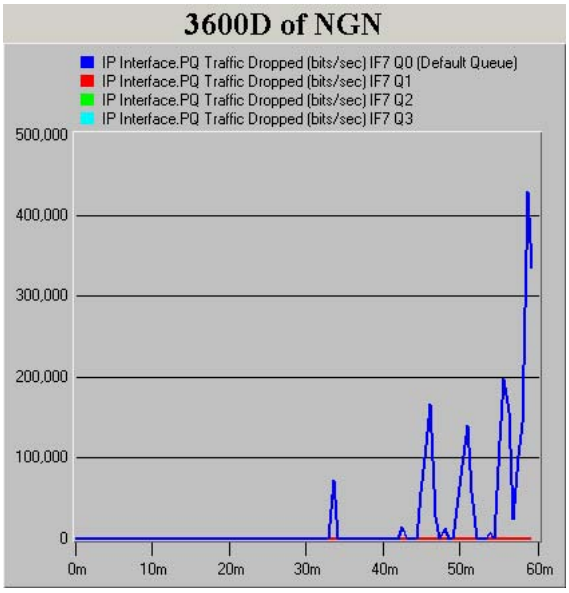


Figura 5.13: Perdas de pacote na interface que liga os roteadores *3600d-3600a*

Outro resultado importante para análise de desempenho destes cenários, é o valor de perda de pacotes na rede do provedor devido ao congestionamento dos enlaces. De acordo com a figura 5.12, uma considerável perda de pacotes ocorre no cenário de melhor esforço e no cenário com combinação de serviços diferenciados e *mpls*. No cenário de melhor esforço, que faz uso do esquema de fila *FIFO* (*first-in-first-out*), o excesso de tráfego gerado pelas aplicações dos clientes *vpn* (*cliente-1* e *cliente-2*) no enlace *3600d-3600a* é descartado pelo roteador *3600d* sem a aplicação de nenhum critério, conforme pode ser visto na figura 5.12.

Para o cenário com combinação de serviços diferenciados e *mpls* a perda de pacotes também ocorre, mas em proporções menores, pois com o uso do esquema de fila *wfq*, cada fila é servida com proteção, ou seja, um tempo de serviço é dado a cada fila proporcionalmente ao peso da fila, exposto na tabela 5.6. A figura 5.13 apresenta a perda de pacotes do cenário com combinação de serviços diferenciados e *mpls* para as quatro filas *wfq* implementadas pelo roteador *3600d*. As filas *q0, q1, q2* e *q3* mapeiam as classes de tráfego AF11, AF21, AF41 e EF respectivamente. De acordo com a figura 5.13, apenas a fila *q0*, que mapeia a classe de menor prioridade (classe AF11), apresenta perda de pacotes. Logo, a perda de pacotes no cenário com combinação de

serviços diferenciados e *mpls* mostrado na figura 5.12, é proveniente do descarte de pacotes da fila *q0*, mostrado na figura 5.13, que faz o mapeamento da classe AF11 pertencente as aplicações *ftp*, *vídeo* e *voip* do *cliente-2* (tabela 5.5).

6

Conclusões

A provisão de qualidade de serviço é parte intrínseca dos serviços emergentes, como as redes privadas virtuais (*VPNs*). De acordo com a padronização do IETF para arquitetura *vpn-mpls* [12], uma solução de provimento de qualidade de serviço para esta arquitetura é a combinação da arquitetura de serviços diferenciados no cliente *vpn* com a rede *mpls* do provedor de serviço, estabelecendo-se um mapeamento de prioridade de tráfego na borda da rede do provedor de serviço.

Visando o estabelecimento de um mapeamento dinâmico de prioridade na borda da rede do provedor de serviços, onde o cliente *vpn* pode a qualquer instante criar e/ou modificar mapeamentos de prioridade, desenvolveu-se neste trabalho uma proposta de expansão da arquitetura *vpn-mpls* utilizando-se a linguagem de especificação formal *SDL*. Com a proposta de expansão da arquitetura criou-se um mecanismo de mapeamento dinâmico entre as prioridades

dos clientes *vpn* e os acordos de nível de serviço contratados do provedor, através da inserção do parâmetro de prioridade de rota do cliente *vpn* na tabela *vrf*, e a modificação do protocolo *mp-bgp* para o transporte dos valores de prioridade de rota.

As especificações de todos os sistemas propostos utilizaram a linguagem de especificação formal orientada à objetos *SDL*, padronizada pelo ITU-T. O uso da ferramenta *case* para o desenvolvimento profissional em *SDL*, como o *SDL TAU Suite* [6], traz grandes vantagens para realização das especificações. A utilização de uma representação gráfica da linguagem *SDL* (*SDL-GR*) facilita bastante o processo de especificação. No capítulo 4, a realização do processo de validação das especificações desenvolvidas permite a detecção e correção de erros na especificação. Além disso, a análise dos resultados de simulações através de diagramas *MSC* gerados pelo simulador é de compreensão bastante simples e permitiu simular diferentes ambientes e casos críticos para verificar o comportamento funcional dos sistemas, além de proporcionar a detecção de erros de lógica nos sistemas.

Este trabalho também desenvolveu uma análise de desempenho da proposta de expansão da arquitetura *vpn-mpls* através de simulações de transporte de serviços multimídia (voz, vídeo e dados) sobre uma infra-estrutura de provedor de serviço baseado na arquitetura *vpn-mpls*. A análise de desempenho em ambiente de simulação, através do uso do simulador *Opnet Modeler*, tornou-se um resultado importante, pois permitiu validar o mapeamento da arquitetura de serviços diferenciados na rede *mpls* do provedor de serviço através de comparações realizada em função da vazão, atraso fim-a-fim e taxa de perda de pacotes, apresentadas no capítulo 5.

Baseado nas propostas de implementação de qualidade de serviço para arquitetura *vpn-mpls* sugeridas neste trabalho, e na análise de desempenho destas propostas, pode-se propor a realização de alguns trabalhos futuros:

- Utilização do algoritmo *randow walk* na validação dos sistemas especificados no capítulo 3, a fim de comparar os resultados de cobertura da árvore (*coverage view*) obtidos com o uso do algoritmo *bit-state* utilizado no capítulo 4.

- Reutilizar o código de especificação da proposta de expansão da arquitetura *vpn-mpls*, fazendo uso de várias instâncias dos blocos que representam os roteadores *ce* e *pe*, a fim

de criar um cenário de rede maior para análise das trocas de informações de roteamento entre todos os nós da rede.

- Implementar a modificação do protocolo *mp-bgp* sugerida, através do uso do *software GNU Zebra* [2], a fim de validar a proposta em um ambiente real de rede.

Referências Bibliográficas

[1] Cisco Networks. http://www.cisco.com.

[2] GNU Zebra. http://www.zebra.org.

[3] OPNET Modeler. http://www.opnet.com.

[4] R. G. Gallager A. K. Parekh. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case. In *IEEE/ACM Transactions on Networking, Vol. 1, No. 3*, 1993.

[5] M. Suzuki P. Knight B. Schliesser A. Nagarajan, J. Sumimoto. Applicability Statement for Virtual Router-based Layer 3 PPVPN approaches, February 2004. draft-ietf-l3vpn-as-vr-01.txt.

[6] Telelogic AB. Telelogic TAU 4.2 SDL Suite Getting Started. Technical report, Telelogic AB Sweden, September 2001.

[7] Telelogic AB. Telelogic TAU 4.2 SDL Suite Metodology Guidelines. Technical report, Telelogic AB Sweden, September 2001.

[8] W.C. Borelli C.G. Macário, M. Pedroso. Designing a multi-user software environment for development and analysis using a combination of OMT and SDL92. In *SDL´97 Time for Testing SDL, MSC and trends, Eigth SDL Forum*, pages 351–365, 1997.

[9] D. Gan V. Srinivasan G. Swallow D. Awduche, L. Berger. RFC 3209 - RSVP-TE: Extensions to RSVP for LSP Tunnels, December 2001.

[10] L. Doldi. *Validation of Communications Systems with SDL.* ISBN: 0-470-85286-0, 2003.

[11] A. Drake. *Fundamentals of Applied Probability Theory.* McGraw-Hill, 1967.

[12] S. John Brannon C. J. Chase J. Clercq P. Hitchen D. Marshall M. J. Morrow A. Vedrenne E. C. Rosen, Y.Rekhter. BGP/MPLS IP VPNs, October 2004. draft-ietf-l3vpn-rfc2547bis-03.txt.

[13] R. Callon E. Rosen, A. Viswanathan. RFC 3031 - Multiprotocol Label Switching Architecture, January 2001.

[14] Y. Rekhter D. Farinacci T. Li A. Conta E. Rosen, G. Fedorkow. RFC 3032 - MPLS Label Stack Encoding, January 2001.

[15] S. Ganti W. C. Lau N. S. S. Van den Bosch F. Chiussi, J. Clercq. Framework for QoS in Provider-Provisioned VPNs, March 2003. draft-chiussi-ppvpn-qos-framework-01.txt.

[16] B. Davie S. Davari P. Vaananen R. Krishnan F. L. Faucheur, L. Wu. RFC 3270 - Multi-Protocol Label Switching (MPLS) Support of Differentiated Services, May 2002.

[17] B. Kang K. Jun H. Lee, J. Hwang. End-to-End QoS Architecture for VPNs: MPLS VPN Deployment in a Backbone Network. In *IEEE MILCOM 2000*, pages 479–483, 2000.

[18] G.J. Holzmann. *Design and Validation of Computers Protocols.* Prentice Hall Software Series, ISBN: 0-13-539834-7, 1991.

[19] J. Guichard I. Pepelnjak. *MPLS and VPN Architectures.* CISCO PRESS, ISBN: 1587050021, 2000.

[20] ITU. Message Sequence Chart (MSC), 1996. Z.120.

[21] ITU-T. Rec. Y.1311.1 - Network-based IP VPNs over MPLS Architecture, July 2001. Rec. Y.1311.1.

[22] ITU-T. Rec. Y.1311 - Network-based VPNs - Generic Architecture and Service Requirements, March 2002. Rec. Y.1311.

[23] ITU-T. Specification and description language (SDL)- Z.100, August 2002. Rec. Z.100.

[24] A. Krywaniuk J. Clercq, O. Paridaens. An Architecture for Provider Provisioned CE-Based Virtual Private Networks Using IPSec, February 2004. draft-ietf-l3vpn-ce-based-02.txt.

[25] O. Paridaens J. Clrcq. Scalability Implications of Virtual Private Networks. *IEEE Communications Magazine*, pages 151–157, May 2002.

[26] N. Ansari J. Zeng. Toward IP Virtual Private Network Quality of Service: A Service Provider Perspective. *IEEE Communications Magazine*, pages 113–119, April 2003.

[27] J.Clercq. QoS considerations for L3 PPVPNs, February 2003. draft-declercq-ppvpn-l3vpn-qos-00.txt.

[28] M. A. Siqueira M. C. Castro, E. T. Nakamura. Análise dos Aspectos de Segurança das VPNs MPLS. *Simpósio Brasileiro de Redes de Computadores - SBRC 2003, Natal-RN*, Maio 2003.

[29] M. F. Magalhaes L. Farias M. C. Castro, M. A. Siqueira. A BGP/MPLS PPVPN Management Information Model and a J2EE-based Implementation Architecture for Policy and Web-based Configuration Management Systems. *IEEE 3rd International Conference on Networking ICN'04, Guadeloupe, French Caribbean*, Fevereiro 2004.

[30] T. B. Pereira M. C. Castro, T. L. Resende. Improving NGN with QoS Strategies. *Opnetwork 2003, Washington-USA*, Agosto 2003.

[31] W. C. Borelli M. C. Castro, N. A. Nassif. QoS Performance Evaluation in BGP/MPLS VPN. *International Information and Telecommunication Technologies Symposium - I2TS'2003, Florianópolis-SC*, Novembro 2003.

[32] G. Wright B. Gleeson T. Sloane R. Bubenik C. Sargor I. Negusse J. Yu P. Knight, H. Ould-Brahim. Network based IP VPN Architecture using Virtual Routers, April 2004. draft-ietf-l3vpn-vpn-vr-02.txt.

[33] T. B. Pereira. Uma Estratégia de Roteamento OSPF Adaptativo Baseado em Estimação de Banda. Msc, UNICAMP, Agosto 2004. Tese de Mestrado em Engenharia Elétrica.

[34] E. C. Rosen. Applicability Statement for BGP/MPLS IP VPNs, October 2004. draft-ietf-ppvpn-as2547-07.txt.

[35] W.C. Borelli R.P. Guimarães. Object-Oriented Development of TINA systems using SDL and Java. In *X Congresso Internacional de Computación, Ciudad de México*, 2001.

[36] M. Carlson E. Davies Z. Wang W. Weiss S. Blake, D. Black. RFC 2475 - An Architecture for Differentiated Services, December 1998.

[37] R. Chandra D. Katz T. Bates, Y. Rekhter. RFC 2858 - Multiprotocol Extensions for BGP-4, June 2000.

[38] Opnet Technologies. Opnet Modeler - Modeling Concepts Manual Vol. 1. Technical report, Opnet Technologies, 2000.

[39] B. Bailey X. Xiao, A. Hannan. Traffic engineering with MPLS in the internet. In *IEEE Network Magazine*, pages 28–33, 2000.

[40] D. Karrenberg . J. de Groot E. Lear Y. Rekhter, B. Moskowitz. RFC 1918 - Address Allocation for Private Internets, February 1996.

[41] T. Li Y. Rekhter. RFC 1771 - A Border Gateway Protocol 4 (BGP-4), March 1995.

A

A linguagem *SDL* e a ferramenta *SDL TAU Suite*

A linguagem *SDL* (*Specification Description Language*)[23] foi publicada primeiramente como uma recomendação do CCITT (*Comité Consultatif International Téléphonique*), no começo dos anos 70. Atualmente o órgão CCITT é representado pelo ITU-T (*International Telecommunications Union - Telecommunication*).

O *SDL* é uma linguagem para a especificação e descrição de protocolos, sistemas de telecomunicações e sistemas de tempo real. O *SDL* se concentra basicamente na especificação dos aspectos comportamentais do sistema, envolvendo dados quando necessário. A linguagem é ao mesmo tempo textual e gráfica, ou seja, os sistemas especificados em *SDL* podem fazer uso de componentes gráficos da linguagem, os quais possuem equivalentes textuais.

O *SDL* possui algumas características que a diferencia de outras linguagens: é uma linguagem padrão, formal, orientada à objetos, portável, escalável, um padrão aberto e permite reuso.

- **Estrutura do sistema**

Primeiramente, deve-se diferenciar os termos especificação, tipo e instância. Uma especificação define um sistema que é composto por diversas entidades que se comunicam entre si. Estas entidades possuem um comportamento definido que determina qual é o seu tipo. Cada um destes tipos pode assumir inúmeras instâncias, todas possuem o mesmo comportamento e as mesmas características, mas existem de forma independente umas das outras, ou seja, o tipo pode ser visto como uma descrição das instâncias que serão criadas para que, ao se relacionar, executem tarefas do sistema.

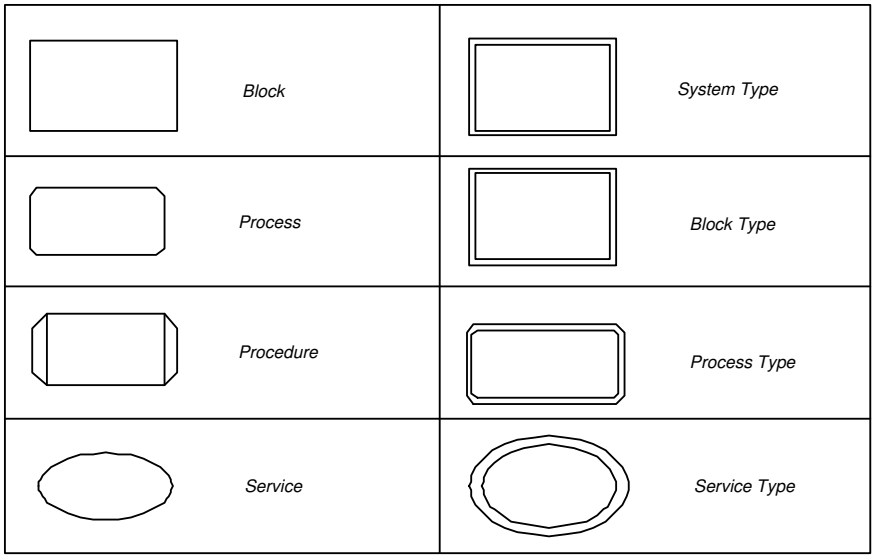


Figura A.1: Representação das principais estruturas *SDL*

Para a especificação dos tipos, ilustrado na figura A.1, o *SDL* oferece algumas estruturas básicas. Dentre elas pode-se destacar:

***System*:** É a entidade mais externa, que representa o sistema como um todo. É composto por diversos blocos que se comunicam entre si e com o ambiente através de canais de sinais. Os sinais e tipos de dados utilizados pelo sistema e que devem ser conhecidos por diversos blocos devem ser declarados neste nível.

***Block*:** É a entidade que agrupa um ou mais processos que se comunicam entre si e com os canais de comunicação do sistema (podendo se comunicar com outros blocos ou com o ambiente). Os sinais e tipos de dados utilizados pelo bloco e que ainda não foram declarados no nível do sistemas devem ser declarados neste nível.

***Process*:** É a estrutura responsável pela especificação comportamental do sistema, através da descrição de uma máquina de estados finita com dados, que é independente das demais. O conjunto de processos que são executados em paralelo é que definem o comportamento do sistema.

***Service*:** Usado para definir parte do comportamento de um processo. Em algumas situações o *service* pode reduzir a complexidade e aumentar o entendimento da especificação de um processo.

As especificações realizadas dentro dos processo, por exemplo na descrição de uma maquina de estado, faz uso dos elementos apresentados nas figuras A.2 e A.3.

- **Troca de sinais**

Os sinais são transportados de um bloco para outro ou para o ambiente externo (*enviroment*) através de canais de comunicação. Cada um destes canais pode ser uni e bidirecional e possui listas entre processos, faz-se uso de rotas de sinais, os quais possuem uma estrutura similar aos canais de comunicação. Os processos também fazem uso de rotas de sinais para se comunicar com o nível mais externo, conectando uma das extremidades da rota de sinais à um canal de comunicação do nível de sistema. Neste caso, os sinais transportados pelas rotas de sinais devem ser os mesmos transportados pelo canal de comunicação.

- **Herança e Especialização de tipos**

No processo de desenvolvimento de um sistema é muito comum a especialização de um *tipo ou type* pré-existente, incluindo novas funcionalidades ou redefinindo antigas. Para isso, o *SDL* fornece a facilidade do reuso de especificações através do conceito de herança de *tipos*, onde todas as propriedades do *supertipo* são herdadas pelo *subtipo*, devendo ser especificadas apenas as diferenças existentes entre os mesmos.

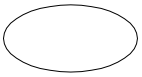
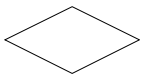
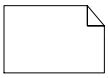
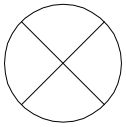
Representação	Descrição
	*Start:* utilizado apenas uma vez no processo indicando o início de sua execução
	*Stop*: utilizado para indicar o término da sua execução de um processo
	*State*: utilizado para indicar o estado em que se encontra a máquina finita que representa o processo/procedimento. Uma mudança de estado só ocorre quando se recebe um sinal. O estado pode ter um nome, ou pode ser declarado o estado ´*´ que se refere a todos os estados existentes, indicando que as ações a serem tomadas são comuns a todos eles. Além disso, após a recepção de um sinal (e execução das ações relacionadas), a execução pode encontrar o estado ´-´que indica que o estado não se alterou devido às ações.
	*Decision*: utilizado para uma tomada de decisão. Consiste de uma questão e diversas possíveis respostas e, dependendo da resposta, a execução toma um caminho distinto.
	*Text*: utilizado para a declaração de variáveis, sinal e tipos de dados.
	*Input*: utilizado para indicar um sinal de entrada. Deve vir sempre após um estado indicando uma transição devido ao recebimento deste sinal.
	*Output*: utilizado para indicar um sinal de saída. Através desta estrutura envia-se sinais e dados para outros processos. Para cada output deve haver um input associado no processo de destino. Para indicar o processo destino pode-se utilizar o endereçamento implícito - omitindo-se o endereço do destino - ou explícito - indicando-se o endereço destino através da opção TO <dest> ou indicando a rota do sinal a seguir através da opção VIA <rota>.
	*Task*: utilizado para definir uma tarefa ou um conjunto de tarefas (separados por ´;´). As tarefas podem ser substituídas por um string com descrição textual das mesmas.
	*Return*: utilizado para indicar o término da execução de um procedimento. Indica um valor de retorno do procedimento, quando for o caso.

Figura A.2: Descrição e representação de algumas estruturas *SDL* utilizada pelo *process*

A herança de *tipos* é feita através da construção *INHERITS*. Caso a herança seja apenas para adicionar novas funcionalidades (sem a redefinição de antigas), deve-se utilizar a

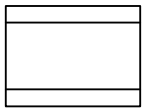
Representação	Descrição
	*Procedure Call*: utilizado para indicar uma chamada a um procedimento.
	*Create*: utilizado para criar uma instância de um processo, desde que esteja no mesmo bloco do processo criador.
	*Comment*: utilizado para a inserção de comentários na especificação.

Figura A.3: Descrição e representação de algumas estruturas *SDL* utilizada pelo *process*

construção *INHERITS ¡supertipo¿ ADDING*. Entretanto, se além da inclusão de novas propriedades, outras existentes forem modificadas, utiliza-se a construção *INHERITS ¡supertipo¿* e define-se a propriedade a ser modificada como *VIRTUAL* no *supertipo*, redefinindo-a com a construção *REDEFINED* no *subtipo*.

Para que se possa utilizar este conceito de herança, alguns *tipos*, presentes na figura A.1, devem ser utilizados:

***System Type*:** Define um sistema a ser reusado por outro através do mecanismo de *Package*.

***Block Type*:** Define um conjunto de blocos com as mesmas características (ou seja, um tipo de bloco). Cada *Block Type* pode ser instanciado inúmeras vezes.

***Process Type*:** Define um conjunto de processos com as mesmas características (ou seja, um tipo de processo). Cada *Process Type* pode ser instanciado inúmeras vezes e cada instância se comporta como uma máquina finita de estados que trabalha de forma independente e concorrente com as demais.

***Service Type*:** conceito de estrutura que pode ser definido como tipo. Usado para diminuir a complexidade de especificações de um processo.

Sendo assim, podem ser especializados apenas os *tiposSystem Type, Block Type* e *Process Type*, sendo que o *tipo System Type* apenas pode ser reutilizado por outro sistema, não podendo

ser redefinido. Para a especialização de processos, novas funcionalidades podem ser adicionadas através da inclusão de novos estados e novas transições e estados pré-existentes. Além disso, pode-se redefinir transições de *Input* e *Start*, declarando-as como *VIRTUAL* no *supertipo* e *REDEFINED* no *subtipo*. Este *subtipo* pode ainda dar origem a um outro *tipo* que herde todas as suas características, e por conseqüência todas as características de seu ancestral. Neste caso, para este novo *tipo*, todas as propriedades *REDEFINED* podem ser novamente redefinidas. Para que isso seja evitado, ao invés de definir as propriedades como *REDEFINED* pode-se utilizar *FINALIZED*, proibindo os *tipos* herdeiros deste *subtipo* de redefinir estas propriedades.

- ***Package***

Um dos conceitos mais importantes da linguagem *SDL* é o conceito de *package*. O *package* permite que uma definição de tipo seja usada por vários sistemas diferentes, similar ao objetivo da biblioteca de classes em C++.

O *Package* permite a criação de uma biblioteca que pode ser utilizada em um sistema. No *package* podem ser definidos tipos de dados, listas de sinais, *System Types, Block Types, Process Types*, dentre outros, podendo ser utilizado integralmente ou parcialmente pelo sistema que o inclua através da cláusula *use*.

Além de poder ser incluído em um sistema, um *package* pode ser incluído em outro *package*, criando assim uma hierarquia de *packages*.

- ***SDL TAU Suite***

O *SDL TAU Suite* [6][7] é uma ferramenta *CASE* (*Computer Aided Design Software Engineering*), distribuída pela Telelogic, que utiliza a linguagem *SDL* para a especificação. validação e simulação dos sistemas. Dentre os componentes da versão utilizada, o *SDL TAU Suite 4.2*, pode-se destacar os seguintes:

Organizador: utilizado para a organização geral das especificações. Fornece uma visão de todos os diagramas e documentos que compõem o sistema, os quais podem ser livremente agrupados em capítulos e módulos de forma a facilitar a organização.

Editor SDL: possibilita a edição comportamental do sistema através do uso da representação gráfica da linguagem *SDL*.

Visualizador de Tipos: permite a visualização do impacto dos mecanismos de herança e especialização no sistema.

Analisador: realiza a análise sintática e semântica da especificação desenvolvida verificando a existência de erros e gerando código para os processos de simulação e validação.

Simulador: permite a simulação das características do sistema, tendo como saída um diagrama de troca de mensagens (*MSC - Message Sequence Chart*) que mostra o comportamento do sistema ao longo do tempo.

Validador: permite a validação do sistema, ou seja, percorre todos os possíveis estados em que o sistema pode se encontrar auxiliando assim na detecção de erros, como por exemplo o não tratamento de algum sinal em determinados estados.

Editor MSC: utilizado pelo simulador para a geração dos diagramas de troca de mensagens.

Visualizador de área coberta: utilizado pelo simulador e pelo validador para exibir uma árvore com todos os estados do sistema destacando aqueles alcançados pelo processo em questão (simulação ou validação).

Geradores de código: possibilitam a geração de código C, Cmicro e CHILL a partir das especificações em *SDL*.

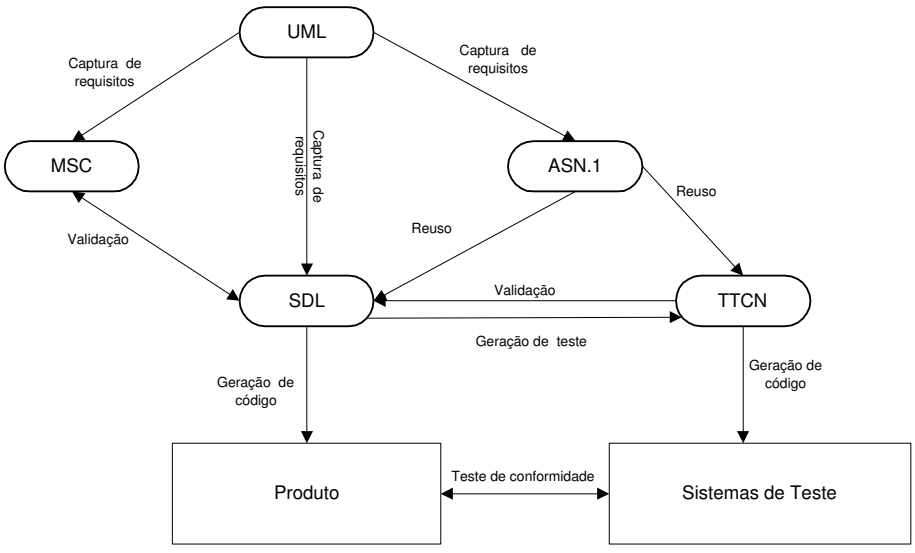


Figura A.4: Relação entre linguagens que fazem interface com o *SDL TAU Suite*

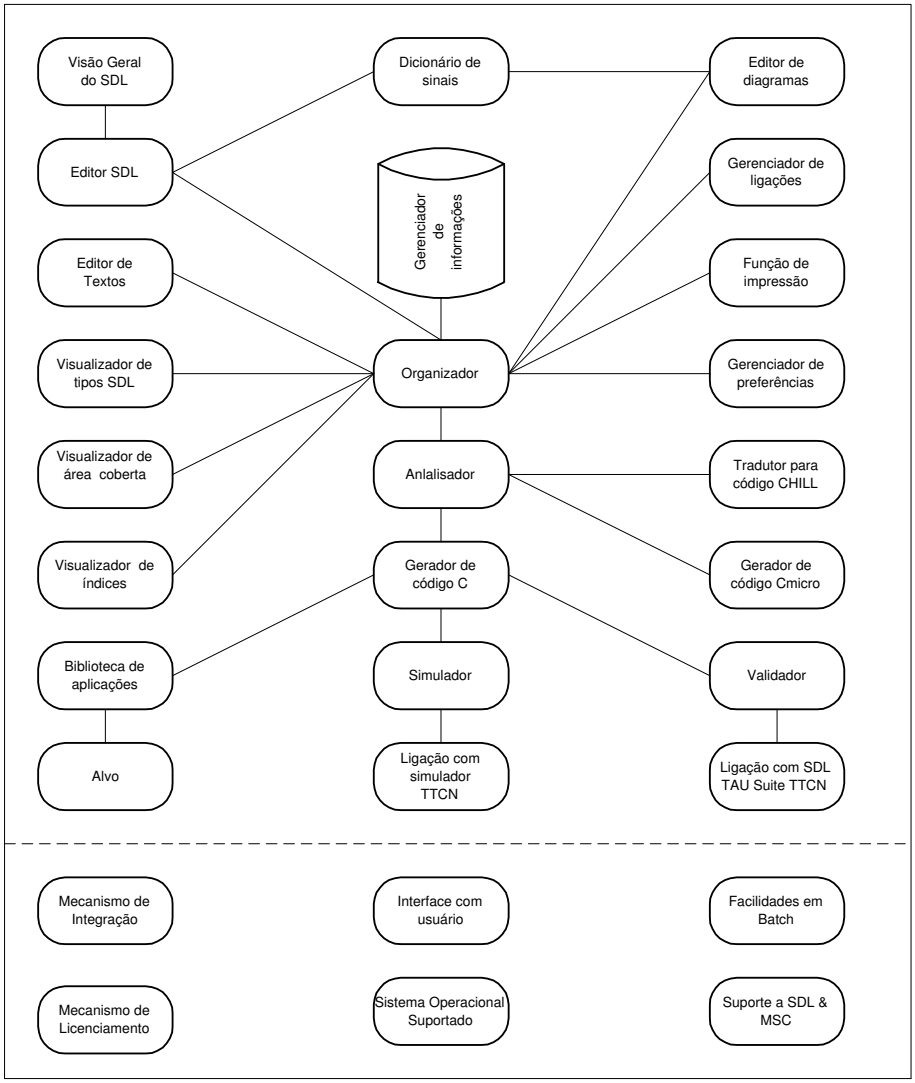


Figura A.5: Visão geral dos módulos que compõem o *SDL TAU Suite*

A presença dos componentes Simulador e Validador são importantes para a realização de testes da especificação, garantindo assim a corretude do sistema. Através do uso destes componentes pode-se encontrar erros na especificação através da simulação de casos críticos, bem como através da análise dos estados alcançados pelo Validador com o uso do visualizador de área coberta (*Coverage Viewer*).

Além disso, o *SDL TAU Suite* permite a integração do desenvolvimento em *SDL* com outras linguagens como o ASN.1 (*Abstract Syntax Notation One*) e UML (*Unifing Modelling Language*), garantindo assim uma maior flexibilidade no processo de desenvolvimento. Existe

também a possibilidade da realização de testes de casos de uso através do uso da linguagem TTCN (*Tree and Tabular Combined Notation*). A relação entre estas linguagens é apresentada na figura A.3.

Uma visão geral dos módulos componentes da ferramenta e suas relações pode ser visto na figura A.4.

O Organizador é responsável por gerenciar quase todos os componentes do pacote *SDL TAU Suite*. Através dele tem-se o acesso à todas as funcionalidades disponíveis. Tanto para a realização da simulação quanto da validação, as especificações passam primeiramente por uma análise (através do Analisador) que garante a corretude sintática das especificações. Após esta análise, é gerado o código C, o qual é utilizado pelos módulos responsáveis pela realização das simulações e da validação. Ainda é disponibilizada uma biblioteca de aplicações que, quando integrada ao gerador de código C, permite a geração de uma implementação em C do sistema especificado e a execução do mesmo.

- ***Algoritmo de Validação Bit-State Exploration***

Entre os algoritmos de validação usado pelo *SDL TAU Suite*, para se construir uma análise mais detalhada do processo de validação, o *Bit-State Exploration* é o que fornece maiores detalhes, enquanto que o *Randow-Walk* e o *Exhaustive Exploration*, apresentam resultados superficiais, melhores apenas, por serem mais simples e mais rápidos, e mais utilizados nos primeiros passos das validações no encontro de erros de especificação.

A fim de reduzir o uso de memória durante o processo de validação (exploração da arvore), o algoritmo *bit-state exploration* [18] armazena o código *hash* de cada modelo de estado, ao invés de guardar todos os estados. A figura A.6 ilustra parcialmente a idéia do algoritmo. De acordo com a figura A.6, depois de executar uma transição *SDL*, o algoritmo calcula o valor de *h*, que representa o código *hash* do estado. O código *hash* é um tipo de *checksum*. De acordo com a figura A.6, por exemplo se *h* é igual a 4 (quatro), o quarto elemento na matriz de *bits* é examinado; se o valor deste quarto elemento for 0 (zero), significa que o estado nunca foi explorado. O quarto elemento da matriz de *bits* é setado para 1. Depois da execução de outra transição, o código *hash* é computado novamente; se o *bit* correspondente da matriz ainda conter o valor 1 (um), significa que o estado já foi explorado, ou dois estados diferentes

possuem o mesmo código *hash* representando uma colisão. Segundo [10], o uso do algoritmo *bit-state exploration* é muito eficiente, mas deve-se ressaltar que este algoritmo não executa uma validação exaustiva, porque é impossível de se garantir que a função do código *hash* não irá algumas vezes dar o mesmo resultado para dois sistemas diferentes.

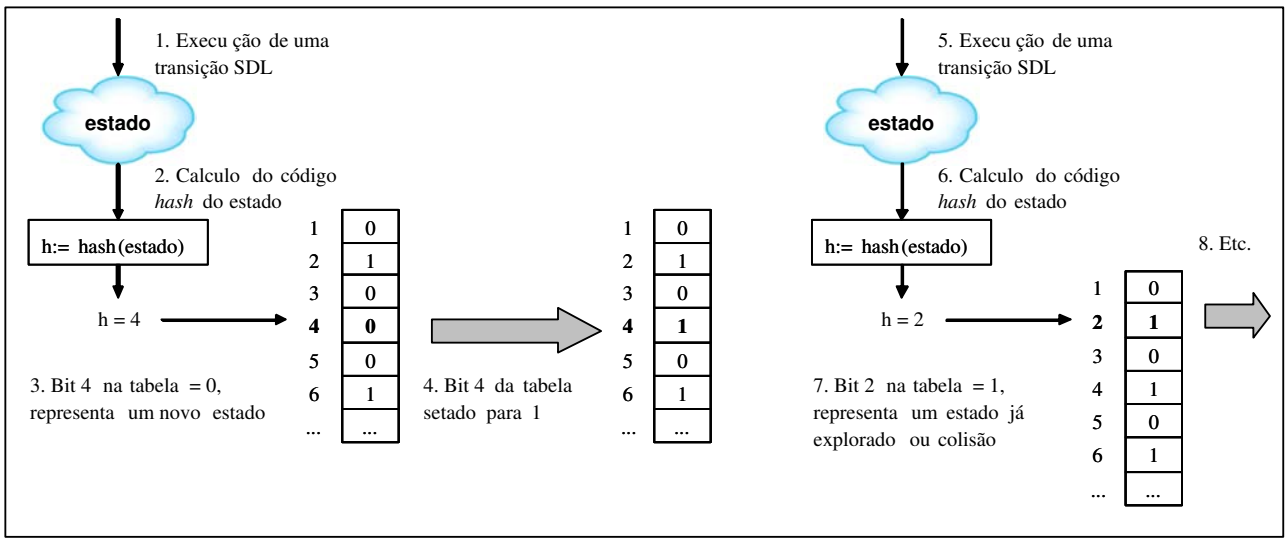


Figura A.6: Algoritmo de Validação *Bit-State Exploration*

B

Plataforma de Simulação *Opnet Modeler*

O uso de plataformas de simulação desenvolvidas comercialmente ou gratuitas ajuda no estudo de problemas que envolvem análise de desempenho de rede, estratégias de roteamento, interoperabilidade de soluções, etc., a medida que estas ferramentas já possuem implementadas várias facilidades importantes para o desenvolvimento do trabalho.

Uma das vantagens de se usar simuladores disponíveis esta na redução do tempo que levaria para escrever seu proprio programa de simulação, além da interface gráfica amigável e complexidade algorítmica que estes simuladores de rede apresentam.

A escolha de uma linguagem ou ferramenta para simulações afeta vários aspectos da simulação e seus resultados. O *Opnet Modeler* foi escolhido por ser uma ferramenta disponível, flexível, extensível, e pelos vários modelos de bibliotecas disponibilizados. *Opnet Modeler* é uma simulador de rede comercial que possui vários modelos e protocolos com implementação

disponíveis. Ele usa a linguagem de programação C para definir o comportamento dos modelos possuindo uma interface gráfica para o usuário (*gui - Graphical User Interface*) disponível para tarefas de modelamento em alto nível.

B.1 Sobre o simulador *Opnet Modeler*

O *Opnet Modeler* é um simulador orientado a eventos que trabalha principalmente com três níveis hierárquicos de implementação: rede, nó e processo. Por "orientado a eventos"entenda que o simulador *Opnet* segue as regras da programação orientada a eventos, onde métodos ou blocos de código são executados à medida que determinados eventos acontecem. Eventos em um simulador de redes podem ser, por exemplo, a chegada de um pacote na interface de um roteador, a satisfação de alguma condição envolvendo variáveis de um processo, etc.

O *Opnet* permite, por meio de um editor gráfico, a construção do modelo de rede e a criação de características de cada um dos elementos do modelo. Possui bibliotecas com informações de equipamentos de diversos fabricantes, sendo possível alterar estes modelos de acordo com necessidade do projeto.

Para cada um dos três níveis principais de implementação do *Opnet* (rede, nó e processo) temos um editor associado. Assim, os três principais editores do *Opnet* são: o Editor de Projeto, o Editor de Nós e o Editor de Processo. Além disso, existem alguns outros editores para especificação de: modelos de link, formatos de pacote, funções densidades de probabilidade, seqüências de cenários de simulação, filtros para estatísticas coletadas, etc.

A figura B.1 mostra a estrutura hierárquica dos três principais editores do Opnet e como estes editores se inter-relacionam. O editor de projeto fornece um espaço para a disposição, interconexão e configuração dos elementos de rede (como por exemplo, roteadores, *workstations*, servidores, enlaces de rede, etc) que caracterizam o cenário de redes a ser simulado [38]. Cada elemento do editor de projeto é denominado genericamente como um nó. O editor de nó fornece um espaço para o modelamento do funcionamento interno de um nó [38]. Tal modelamento é normalmente realizado pela disposição, interconexão e configuração de elementos denominados "módulos"que representam processadores e/ou buffers de propriedade do nó. O editor de

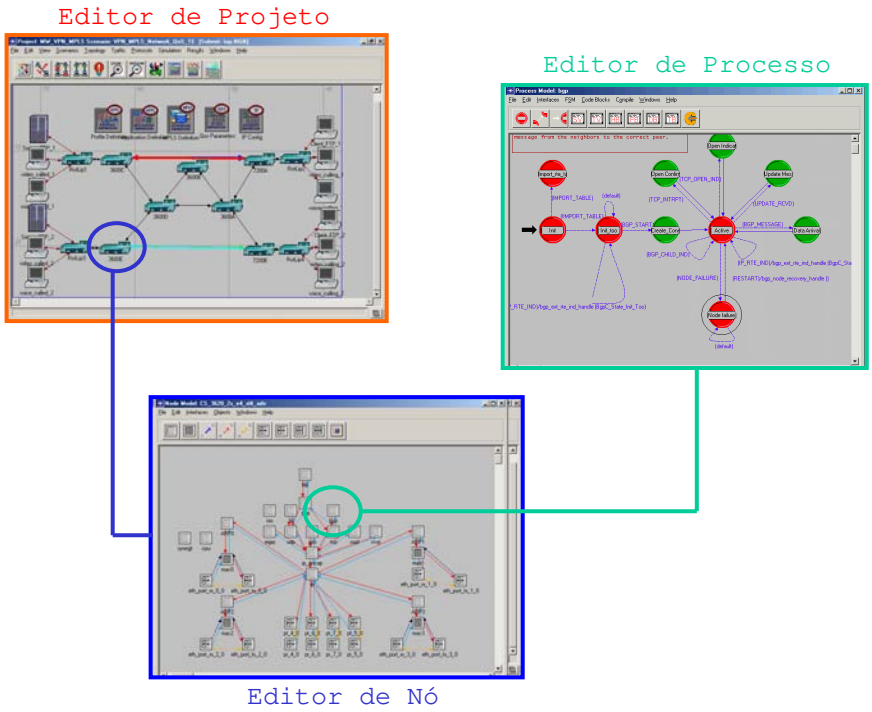


Figura B.1: Estrutura hierárquica dos três editores do *Opnet Modeler*[33]

processo, por sua vez, representa na forma de variáveis e de um diagrama de transição de estados o comportamento seguido por um buffer ou processador possível de ser utilizado em um modelo de nó [38]. Assim, a inter-relação entre os editores pode ser resumida no seguinte:

Cada nó no editor de projeto corresponde a uma instância de um modelo de nó. Um modelo de nó, por sua vez, é especificado no editor de nó por elementos chamados "módulos"que correspondem a instâncias de modelos de processos. Finalmente, são os modelos de processos que especificam através de um diagrama de transição de estado e programação orientada a evento todo o comportamento de um módulo. Modelos de processo são especificados no Editor de Processo.

A existência desta estrutura hierárquica de editores simplifica a especificação de um novo projeto e/ou alteração/criação de novos módulos de simulação, como um novo elemento de rede, ou um novo protocolo de roteamento, esquema de fila, etc. No nível de rede representa-se graficamente a topologia da rede, instanciando modelos de nós e *links*, configurando-os e interconectando-os. Também neste nível é especificado o tráfego à que a rede estará submetida,

são selecionadas as estatísticas a serem coletadas e é executada a simulação. O nível de rede especifica o que denominamos um "cenário de simulação". Através do editor de projeto pode-se especificar vários cenários de simulação para um mesmo projeto, cada cenário com seus nós, *links*, topologia, tráfego e estatísticas a serem coletadas.

A figura B.2 apresenta o editor de projeto do *Opnet* para o cenários utilizados nas simulações realizadas no capítulo 5. Além de roteadores, *servers*, *workstations* e *links*, os cenários criados para as simulações realizadas possuem também outros elementos, nomeados *profile definition*, *application definition*, *mpls definition*, *qos parameters* e *ip config*, objetos especiais do nível de rede utilizados respectivamente para, definição das características de cada elemento utilizado na topologia da rede, definição das aplicações a serem utilizadas nas simulações, definição dos parâmetros *mpls* configuráveis pelo simulador nos roteadores que pertencem ao provedor de serviço, e os dois últimos usados para definição dos parâmetros de qualidade de serviço configuráveis nos elementos da rede.

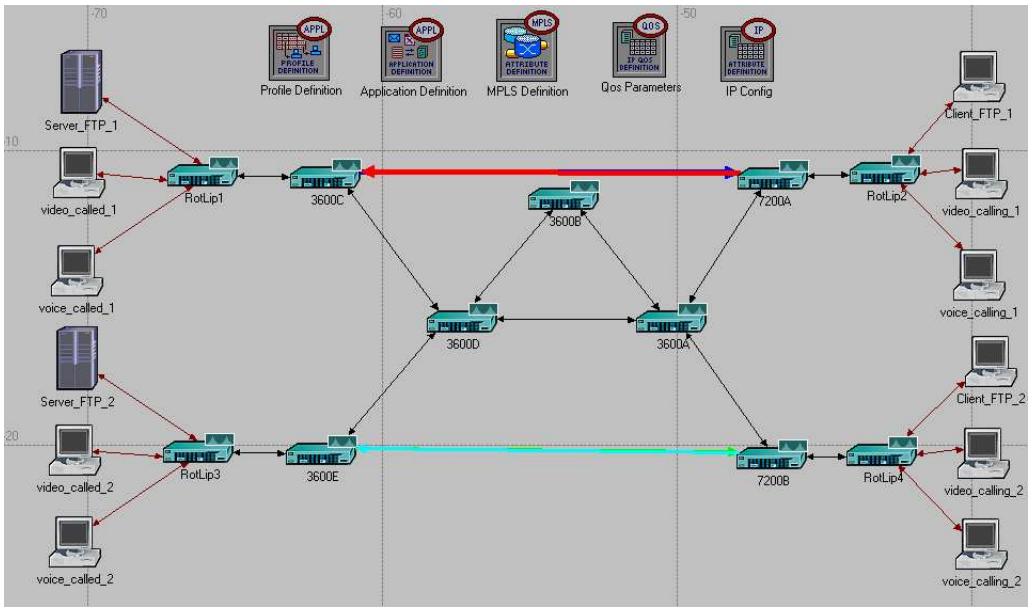


Figura B.2: Exemplo do cenário do projeto desenvolvido no *Opnet*

Cada nó adicionado a um cenário de simulação corresponde a uma instância de um "modelo de nó". Nós são utilizados para representar dispositivos que podem ser conectados formando uma rede. Um modelo de nó tem sua estrutura interna especificada no nível de nó (editor de nó).

Cada elemento da estrutura interna de um modelo de nó é denominado "módulo", podendo gerar, enviar e receber pacotes dos outros módulos a fim de realizar sua função dentro do módulo. Módulos normalmente representam protocolos de comunicação, algoritmos, disciplinas de filas, geradores de tráfego, portas de comunicação, etc. É no editor de nós, portanto, que vemos a pilha de protocolos do roteador. A figura B.3 apresenta o editor de nó do *Opnet* e o modelo de nó que define a estrutura interna dos roteadores utilizados nas simulações. Os módulos nomeados *pr-X-0* e *pt-X-0* correspondem respectivamente à interface de entrada *X* e à interface de saída *X* do roteador. *Eth-port-tx-* e *Eth-port-rx-* representam a camada *Ethernet* da interface *X* do roteador. Além destes, temos os módulos correspondentes a camada *IP* (*ip* e *ip-encap*), e outros módulos para o estudo em questão (*ospf, udp, rip, igrp, eigrp, tcp, bgp, tpal, ldp, arp, mac*).

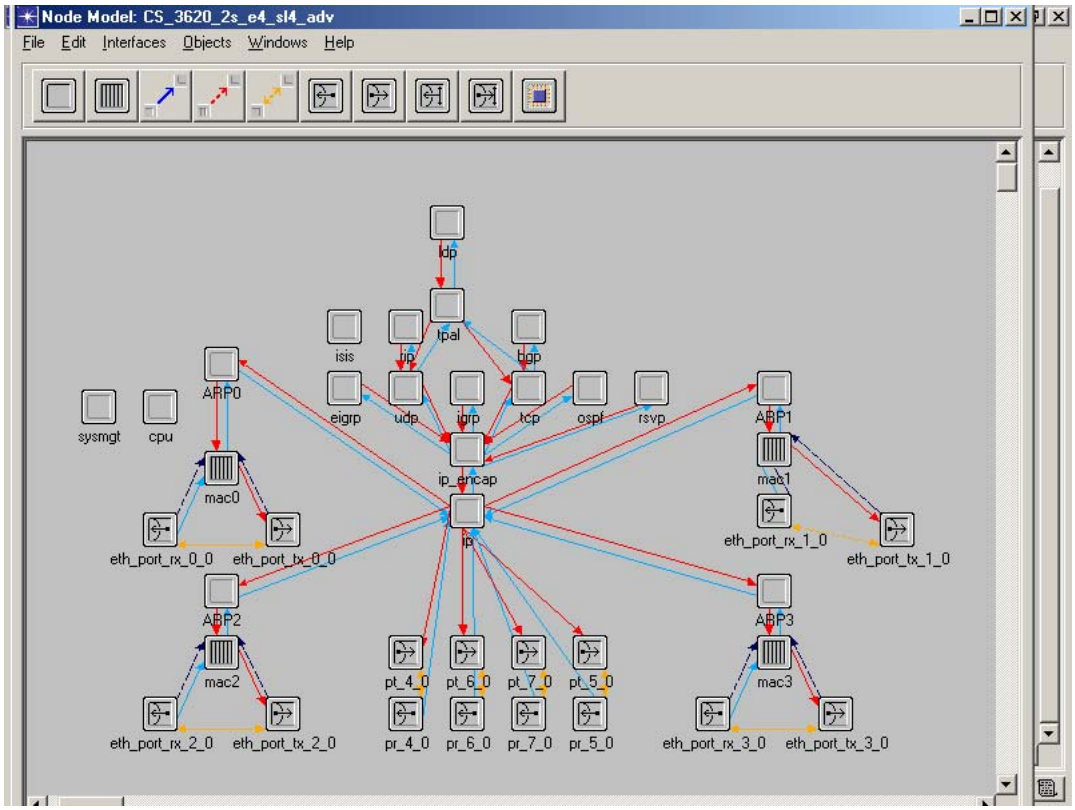


Figura B.3: Modelo de nó utilizado no *Opnet*

Com exceção dos módulos correspondentes às interfaces de entrada e saída do roteador (denominados no *Opnet* transmissores e receptores, respectivamente), todos os demais módulos

de um modelo de nó (*buffers* e processadores) correspondem à uma instância de um modelo de processo. Um modelo de processo (especificado no editor de processo) define o comportamento de um módulo através de um diagrama de transição de estado. De certa forma pode-se dizer que o núcleo de todo o comportamento final de um nó localiza-se, enfim, neste último nível da hierarquia. São os modelos de processo que implementam o comportamento dos protocolos de comunicação, algoritmos, disciplinas de filas, geradores de tráfego, etc. Além de definir também seus atributos e estatísticas. A figura B.4 apresenta o editor de nó do *Opnet* e os modelos de processo para a especificação do comportamento do módulo de roteamento *bgp*.

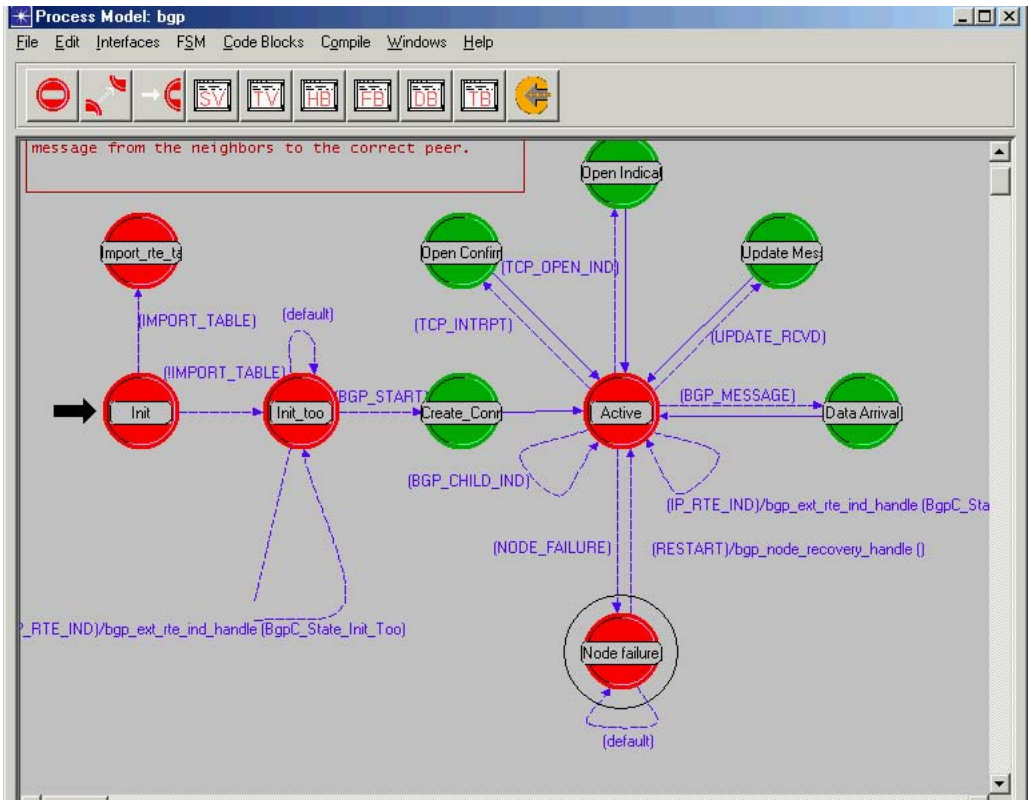


Figura B.4: Modelo de processo para o roteamento *bgp* usado no *Opnet*

C

Especificações em SDL

Este apêndice apresenta os diagramas das especificações gráficas em *SDL*, referentes as propostas dos sistemas especificados da arquitetura *vpn-mpls*. Na seção C.1 é apresentado os diagramas usados na especificação da arquitetura *vpn-mpls*, representado pelo sistema *basicarquitecture-scenario1*. Na seção C.2 é apresentado os diagramas usados na especificação da proposta de expansão da arquitetura *vpn-mpls*, representado pelo sistema *basicarquitecture-scenario2*. Por motivo de organização, e para facilitar o entendimento dos sistemas especificados, este apêndice apresenta todos os diagramas usados nas especificações, inclusive os já ilustrados no capítulo 3.

C.1 Sistema *basicarquitecture-scenario1*

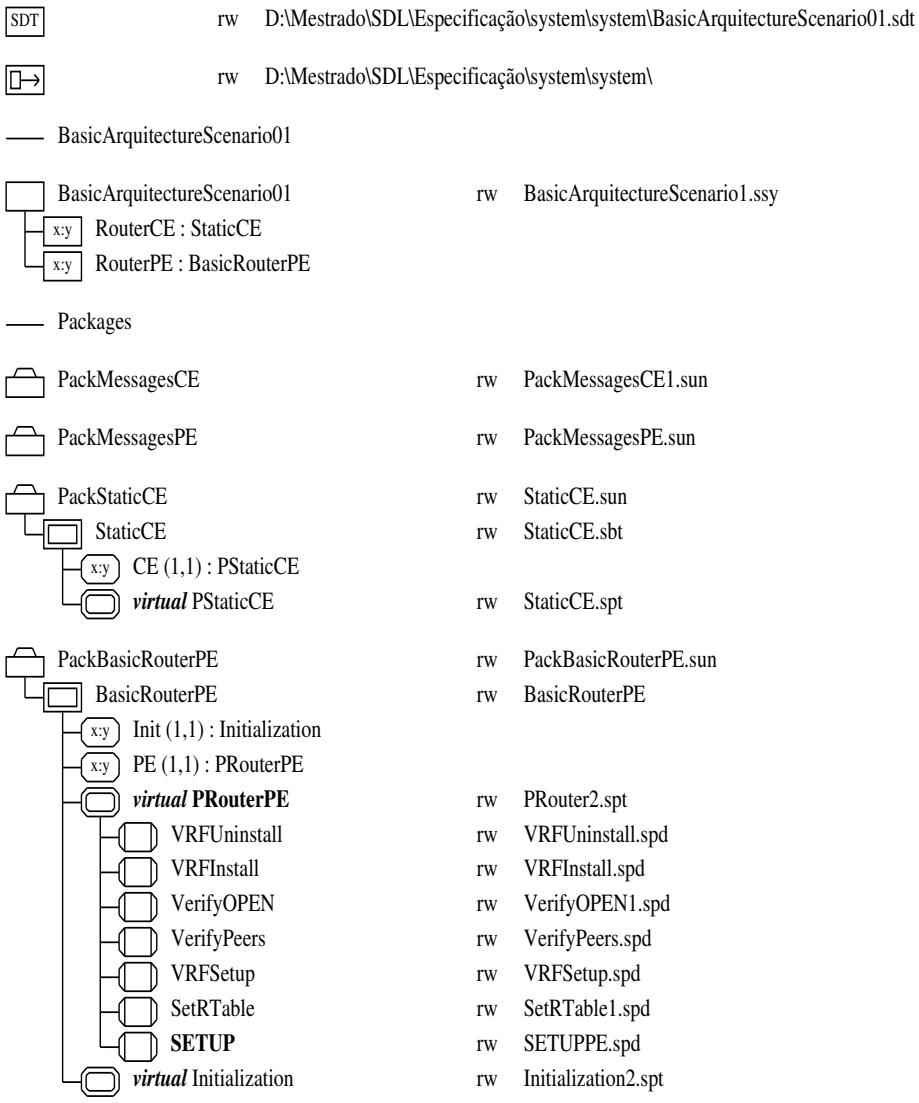


Figura C.1: Visão do Organizer(*SDL TAU Suite*) do sistema *basicarquitecture-scenario1*

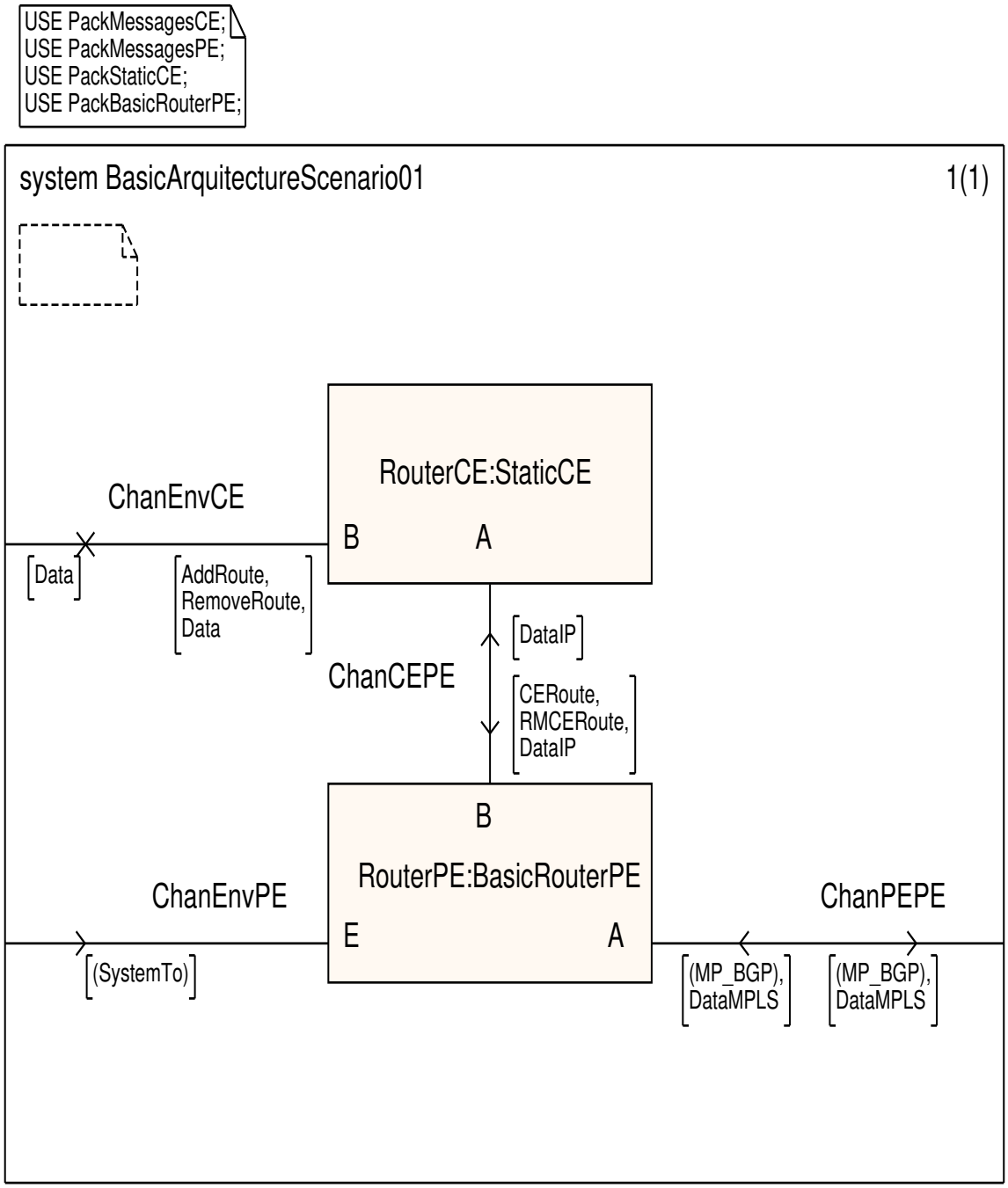


Figura C.2: Sistema *basicrquitecture-scenario1*

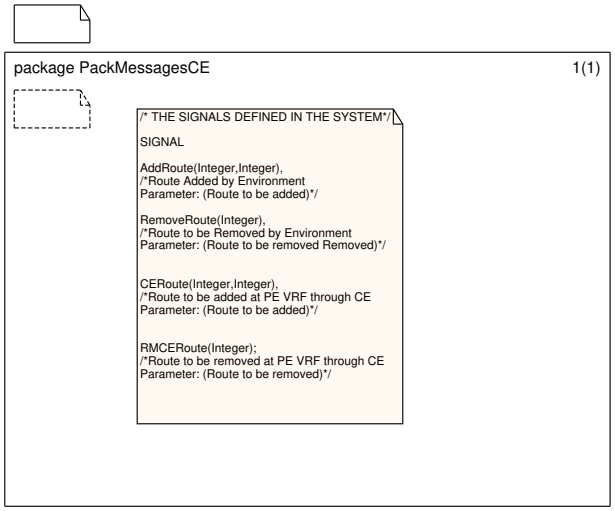


Figura C.3: Sinais do pacote *packmessage-ce*

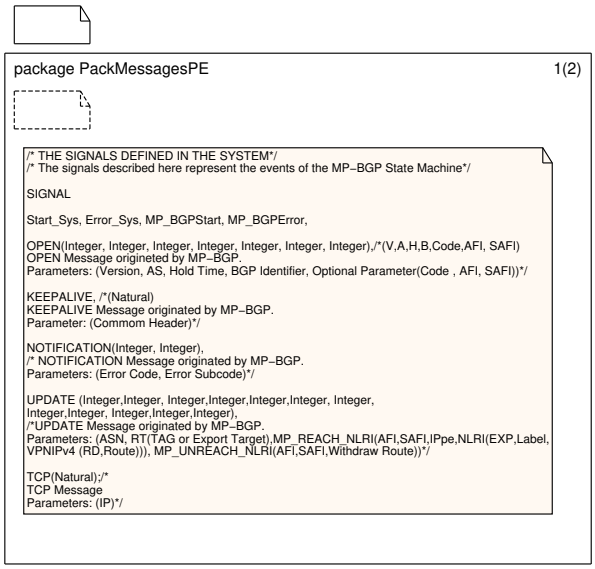


Figura C.4: Sinais do pacote *packmessage-pe*

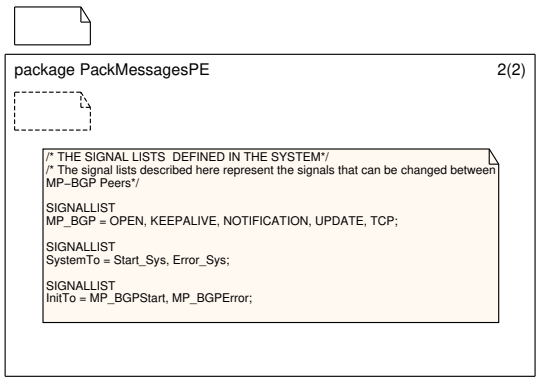


Figura C.5: Definição das listas de sinais do sistema *basicarquitecture-scenario1*

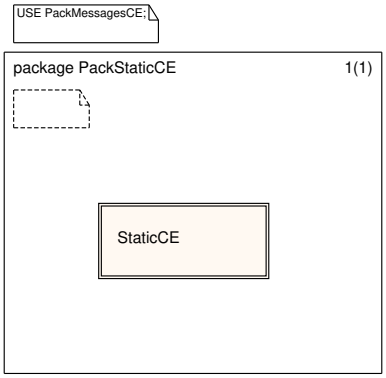


Figura C.6: Pacote *packstatic-ce*

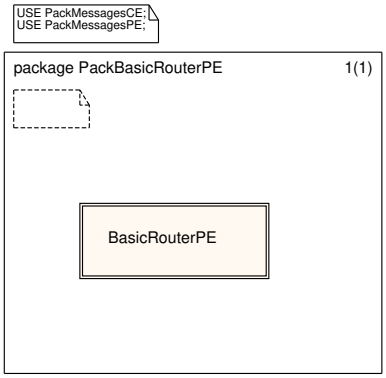


Figura C.7: Pacote *packbasicrouter-pe*

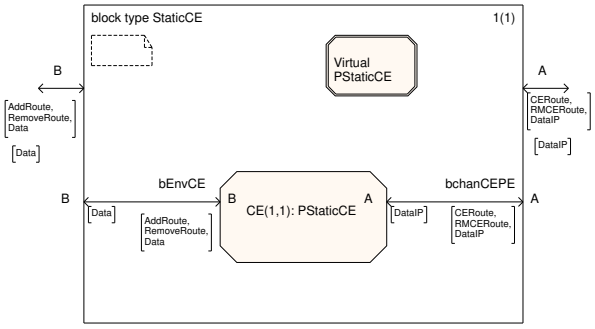


Figura C.8: Bloco *static-ce* do sistema *basicarquitecture-scenario1*

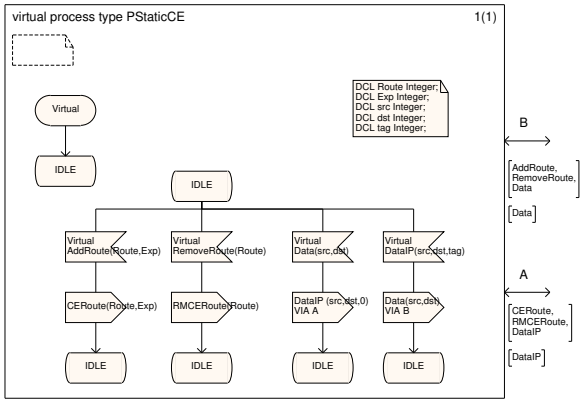


Figura C.9: Processo *pstatic-ce* do bloco *static-ce*

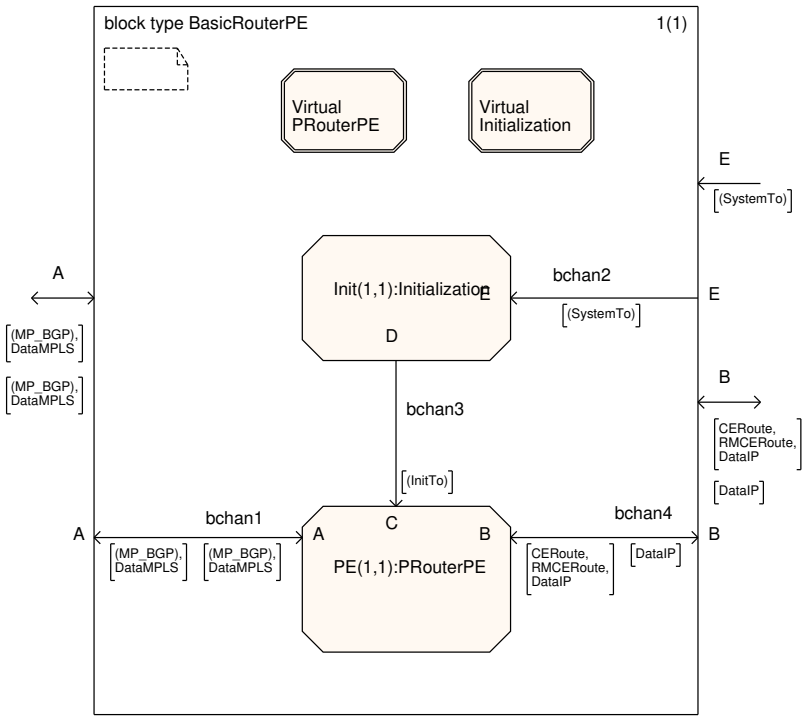


Figura C.10: Bloco *basicrouter-pe* do sistema *basicarquitecture-scenario1*

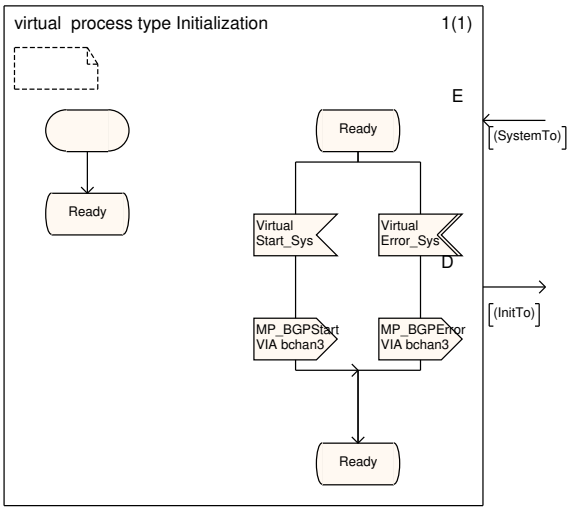


Figura C.11: Processo *Initialization* do bloco *basicrouter-pe*

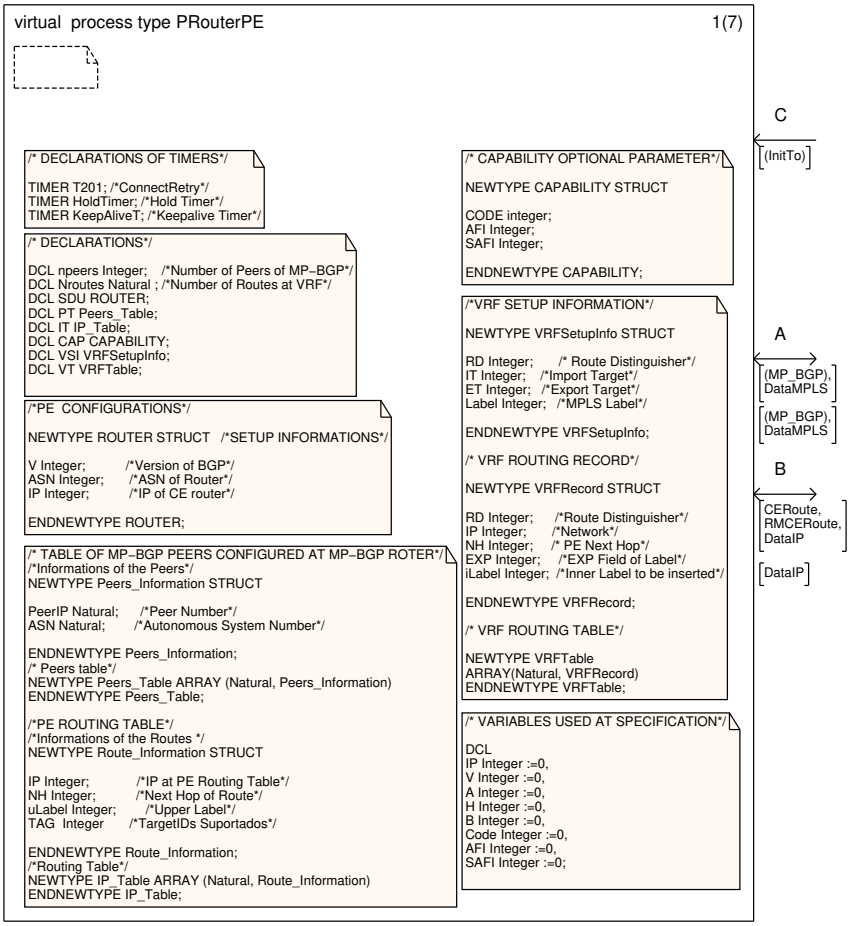


Figura C.12: Declaração das variáveis do processo *prouter-pe* pertencentes ao bloco *basicrouter-pe*

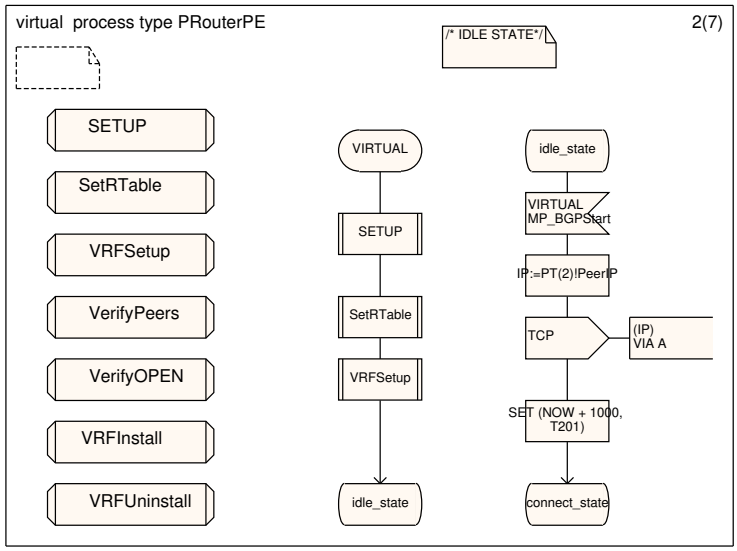


Figura C.13: Estado *idle*, processo *prouter-pe* do bloco *basicrouter-pe*

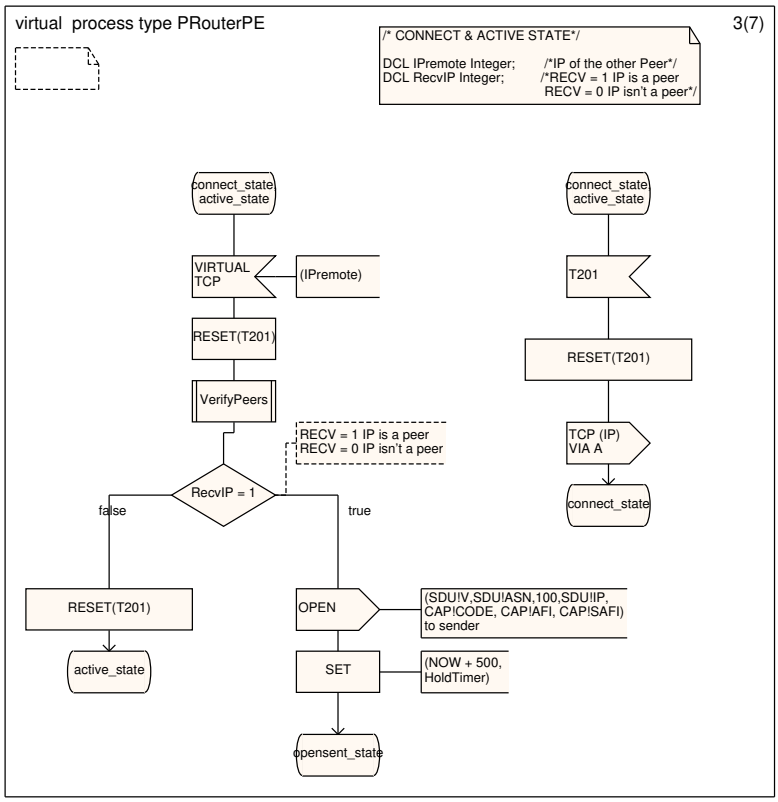


Figura C.14: Estado *connect e active*, processo *prouter-pe* do bloco *basicrouter-pe*

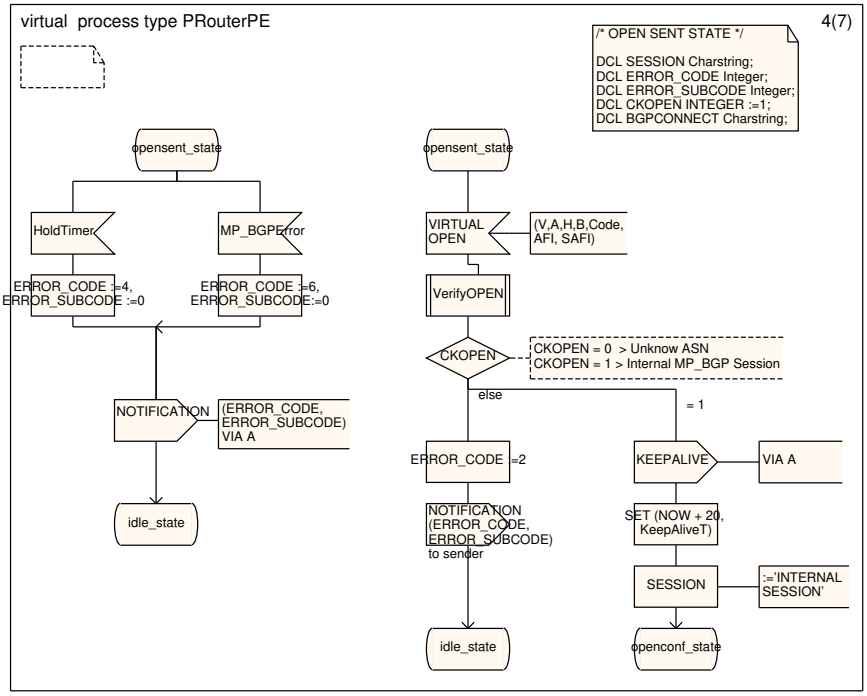


Figura C.15: Estado *open sent*, processo *prouter-pe* do bloco *basicrouter-pe*

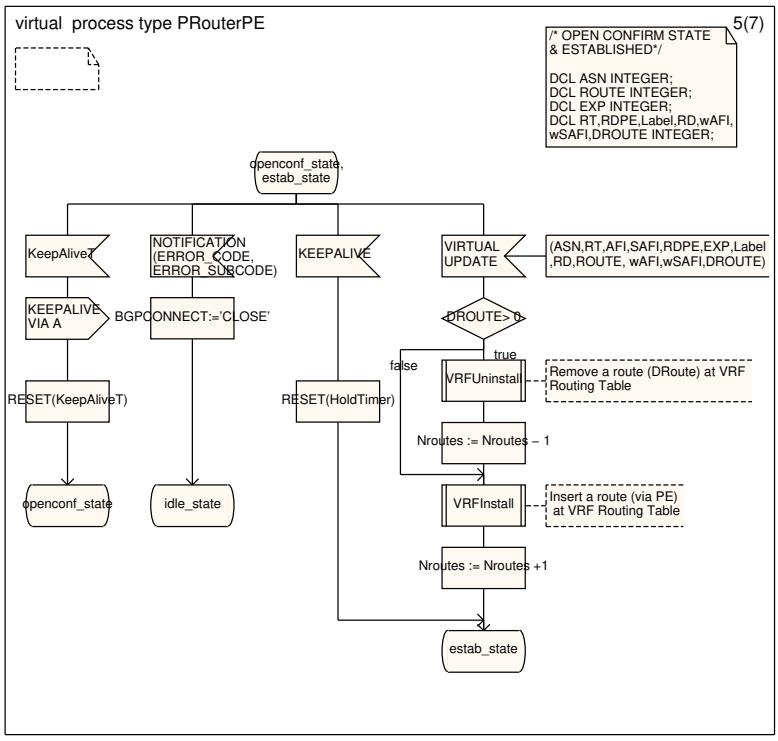


Figura C.16: Estado *open confirm* e *established*, processo *prouter-pe* do bloco *basicrouter-pe*

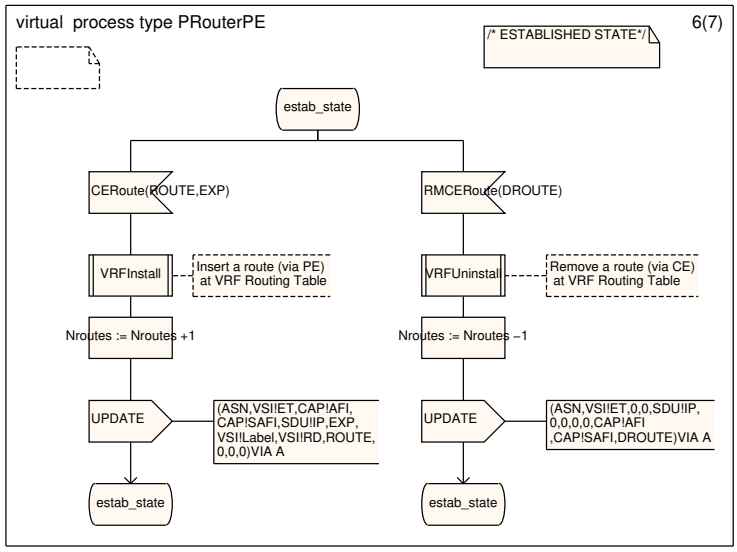


Figura C.17: Estado *established*, processo *prouter-pe* do bloco *basicrouter-pe*

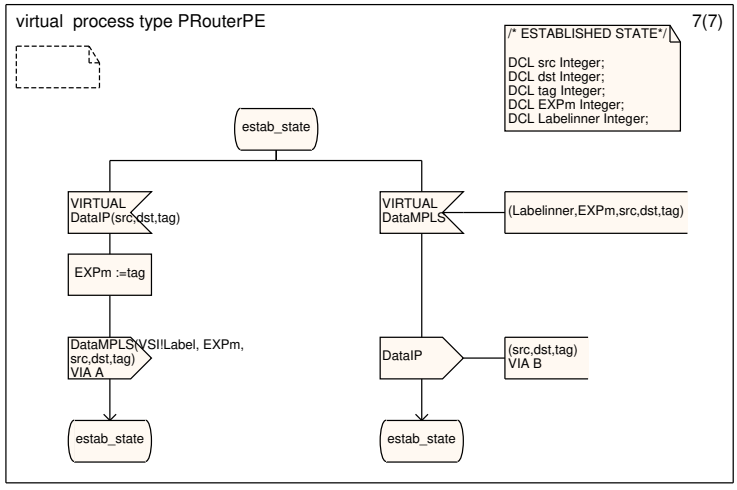


Figura C.18: Estado *established*, processo *prouter-pe* do bloco *basicrouter-pe*

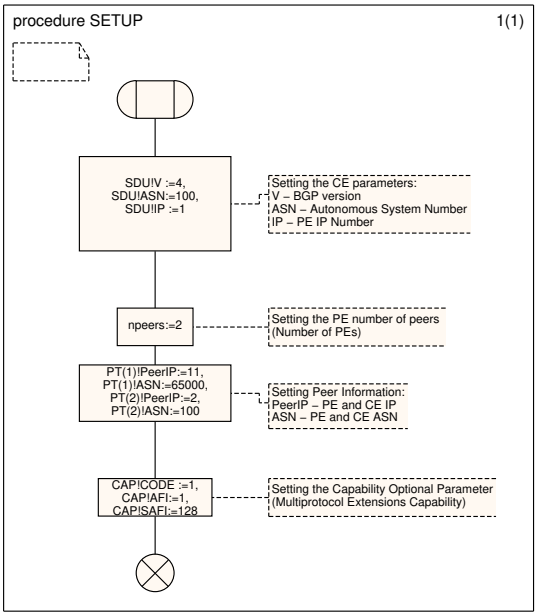


Figura C.19: Procedimento *setup* do processo *prouter-pe*

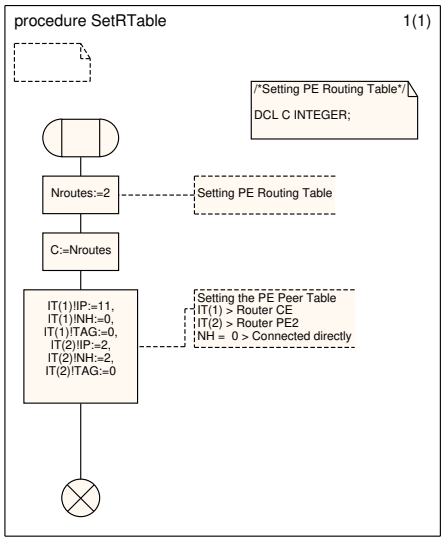


Figura C.20: Procedimento *setrtable* do processo *prouter-pe*

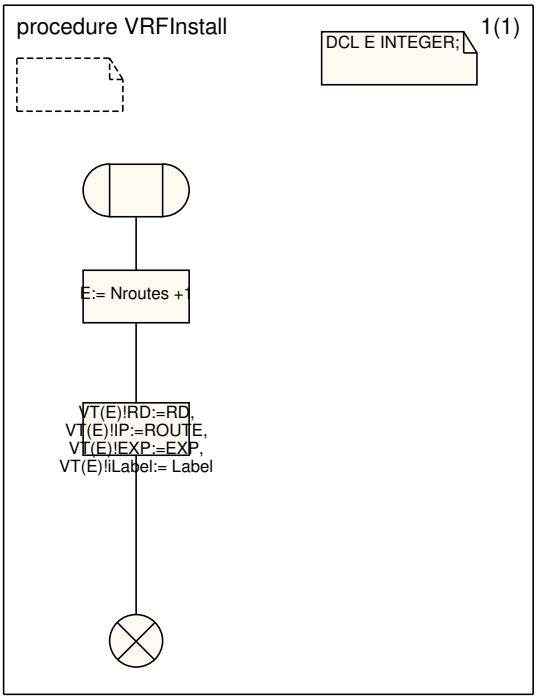


Figura C.21: Procedimento *vrf-install* do processo *prouter-pe*

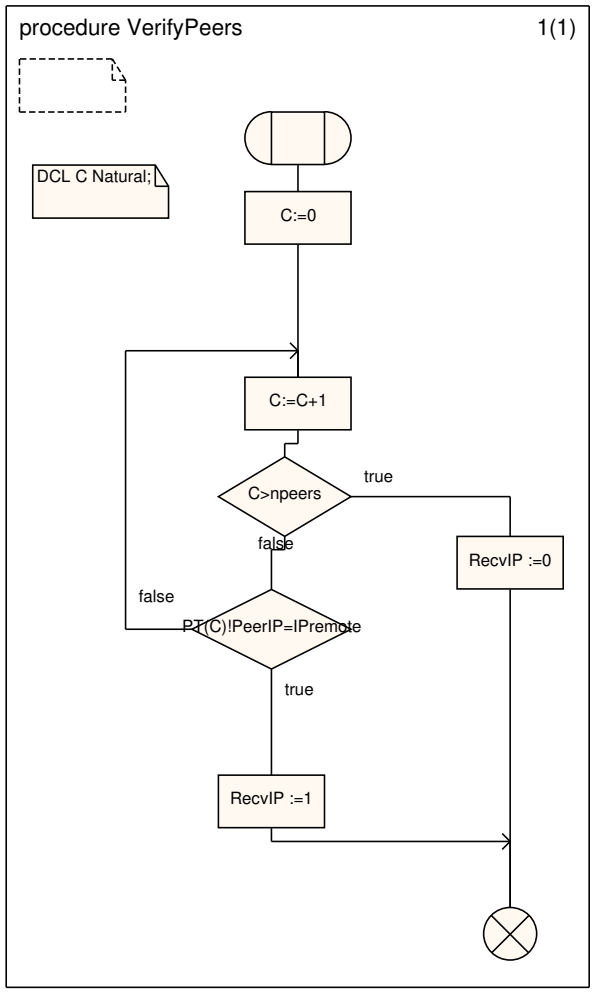


Figura C.22: Procedimento *verify-peers* do processo *prouter-pe*

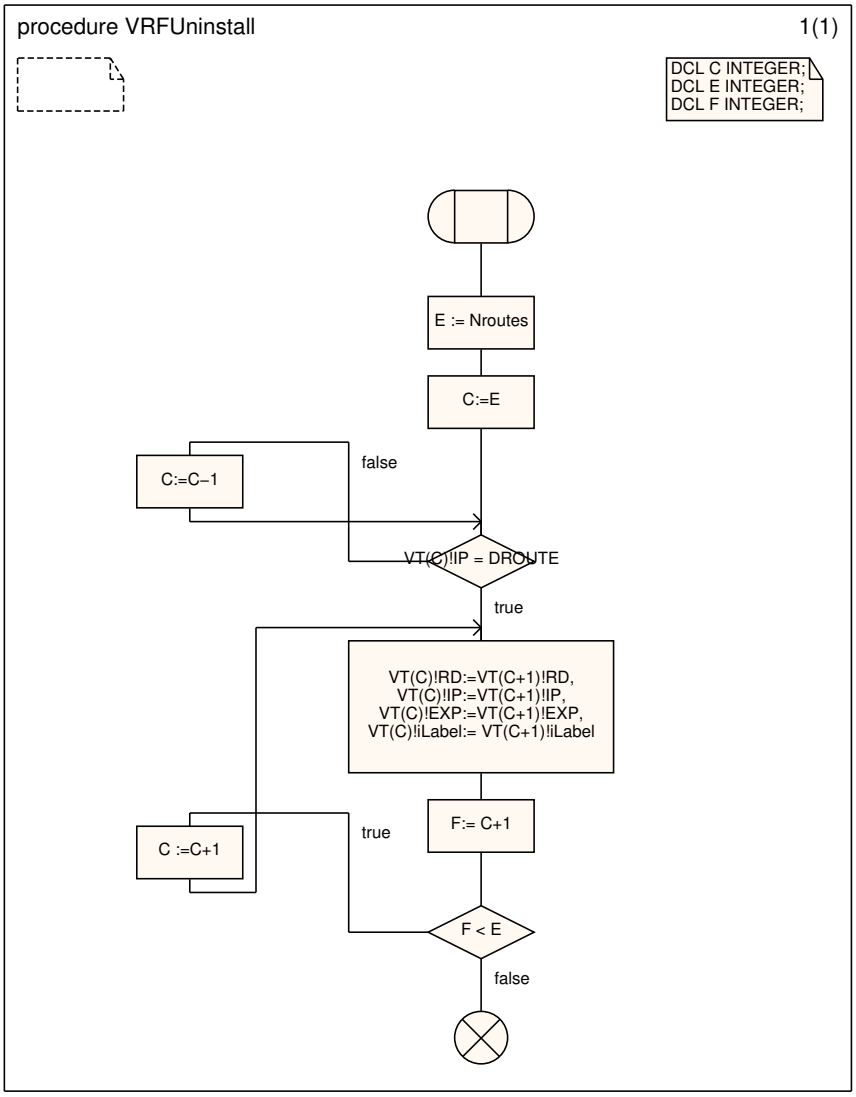


Figura C.23: Procedimento *vrf-uninstall* do processo *prouter-pe*

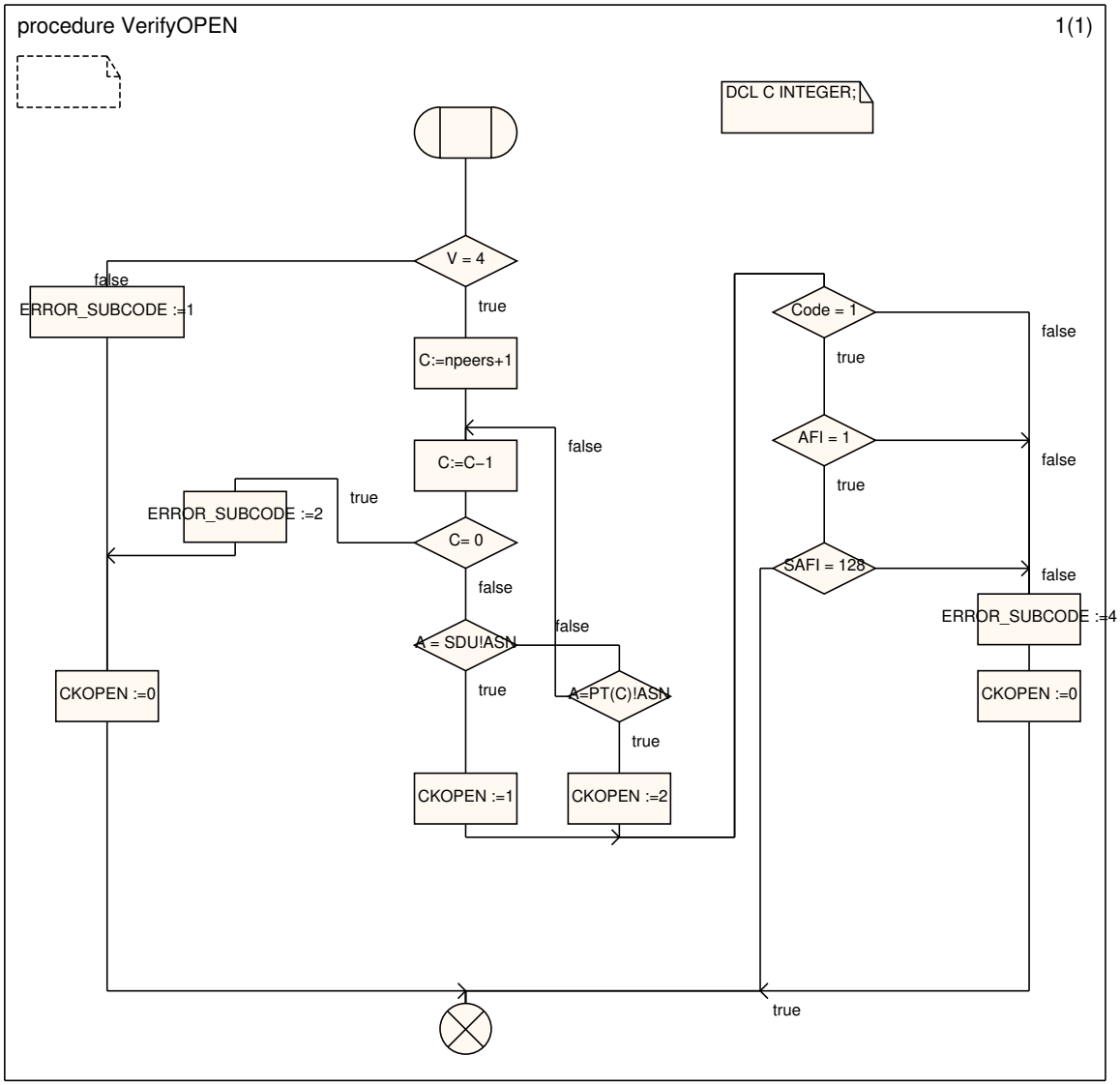


Figura C.24: Procedimento *verify-open* do processo *prouter-pe*

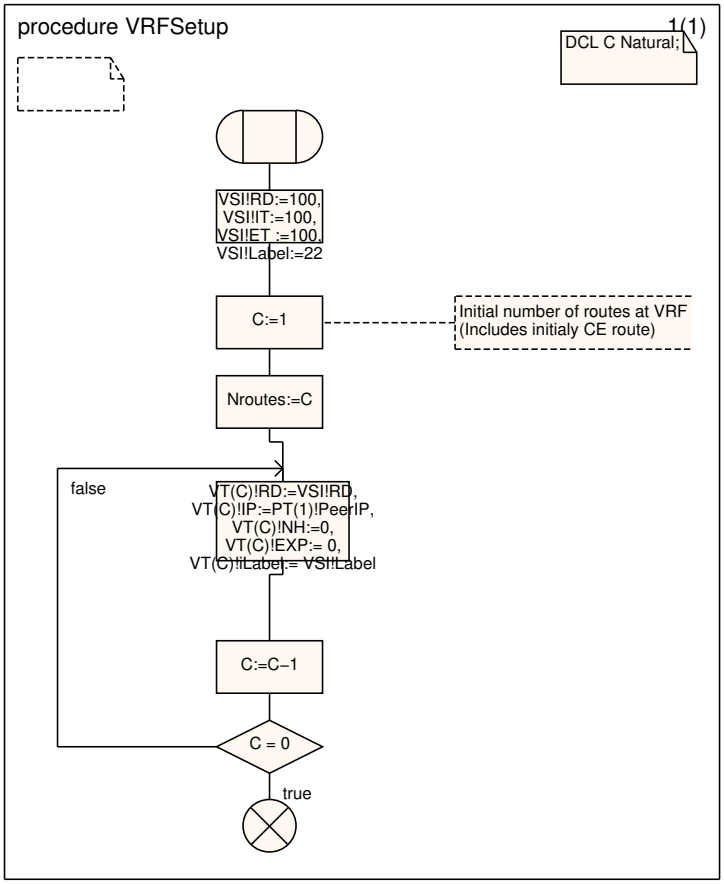


Figura C.25: Procedimento *vrf-setup* do processo *prouter-pe*

C.2 Sistema *basicarquitecture-scenario2*

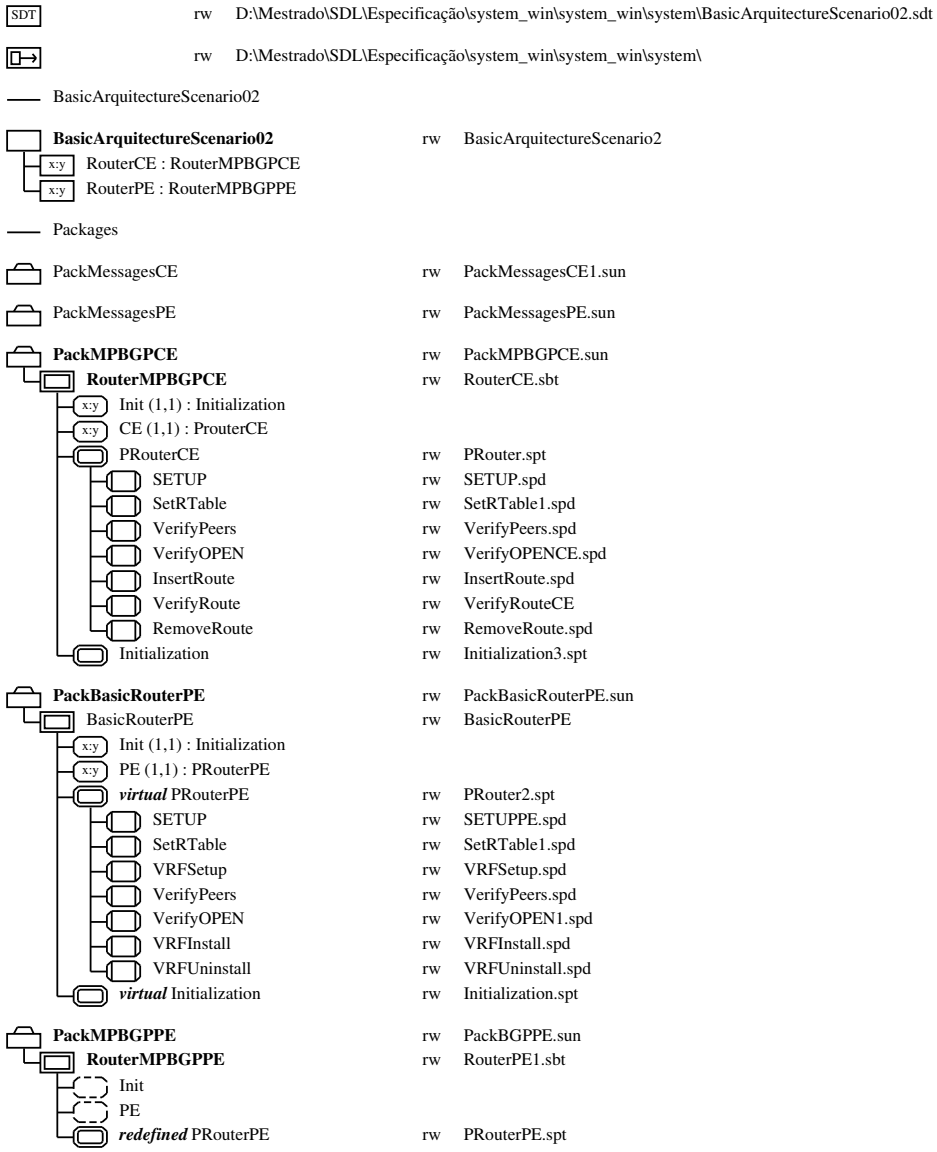


Figura C.26: Visão do Organizer(*SDL TAU Suite*) do sistema *basicarquitecture-scenario2*

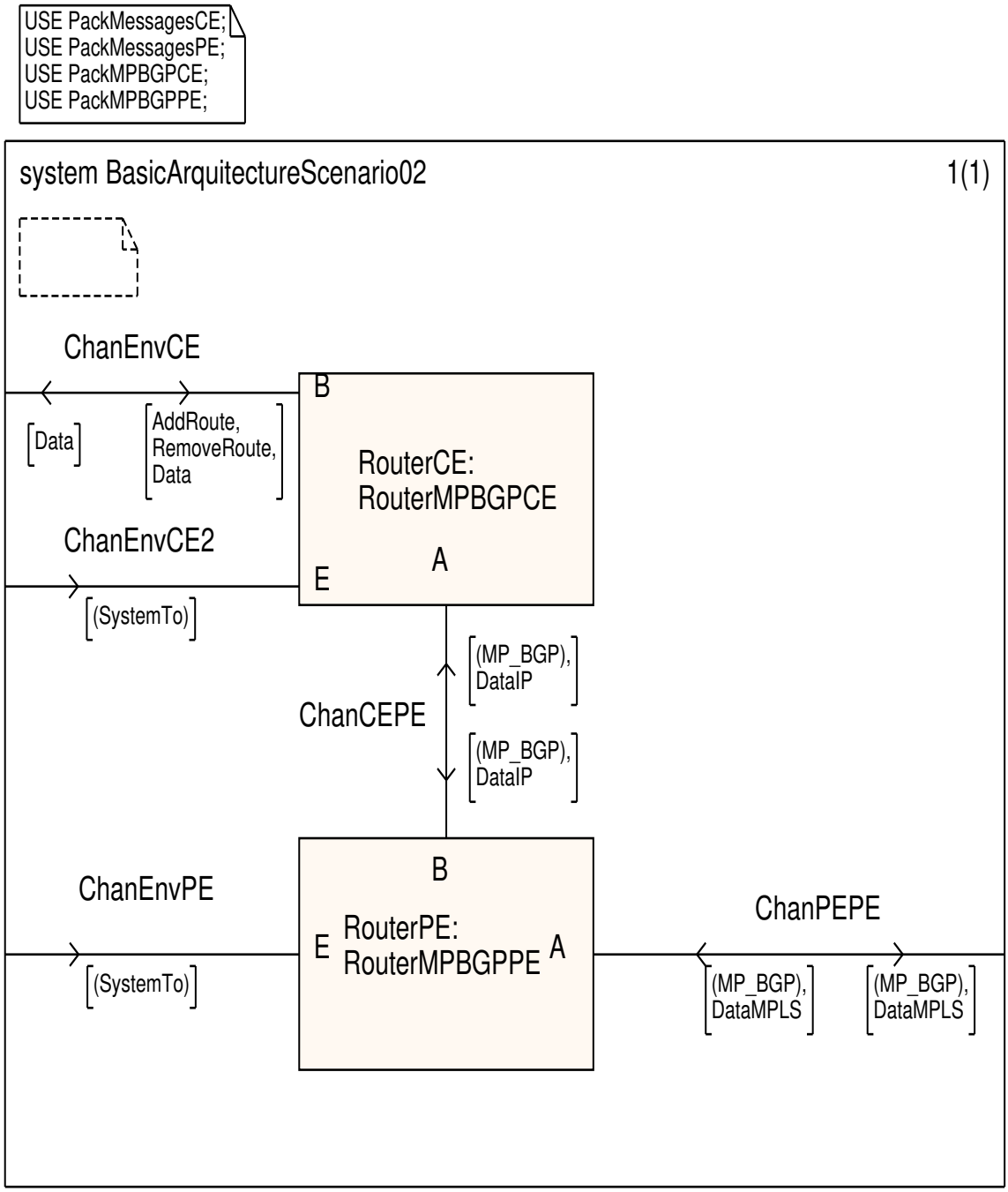


Figura C.27: Sistema *basicrquitecture-scenario2*

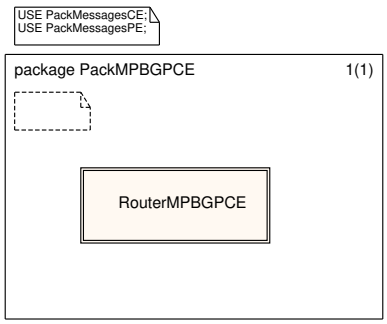


Figura C.28: Pacote *packmpbgp-ce*

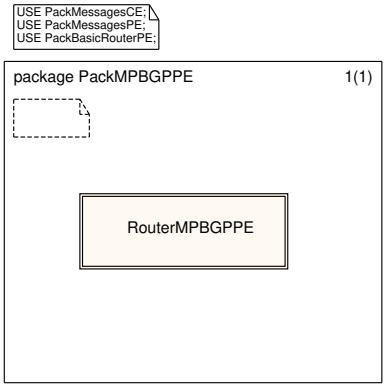


Figura C.29: Pacote *packmpbgp-pe*

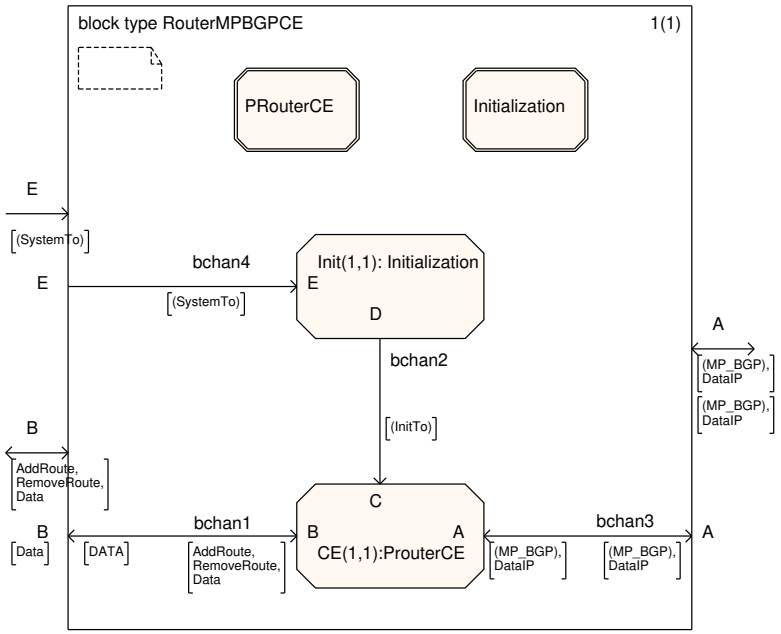


Figura C.30: Bloco *routermpbgp-ce* do sistema *basicarquitecture-scenario2*

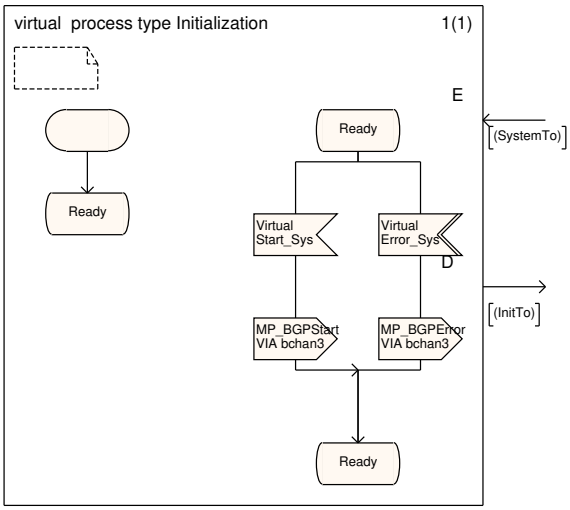


Figura C.31: Processo *Initialization* do bloco *routermpbgp-ce*

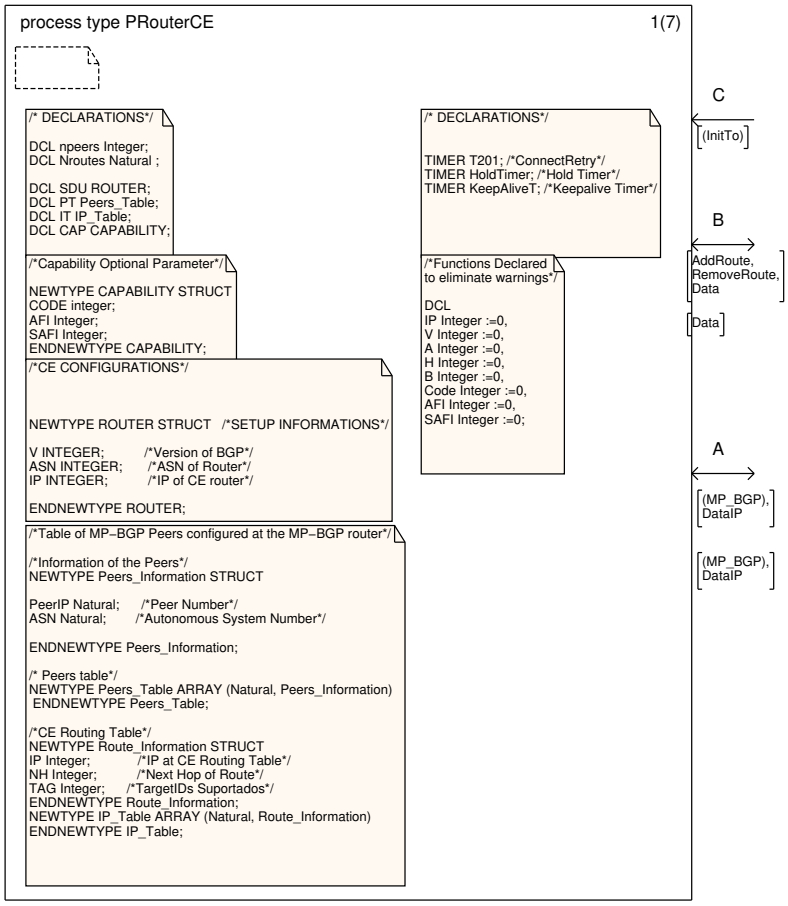


Figura C.32: Declaração das variáveis do processo *prouter-ce* pertencentes ao bloco *routermpbgp-ce*

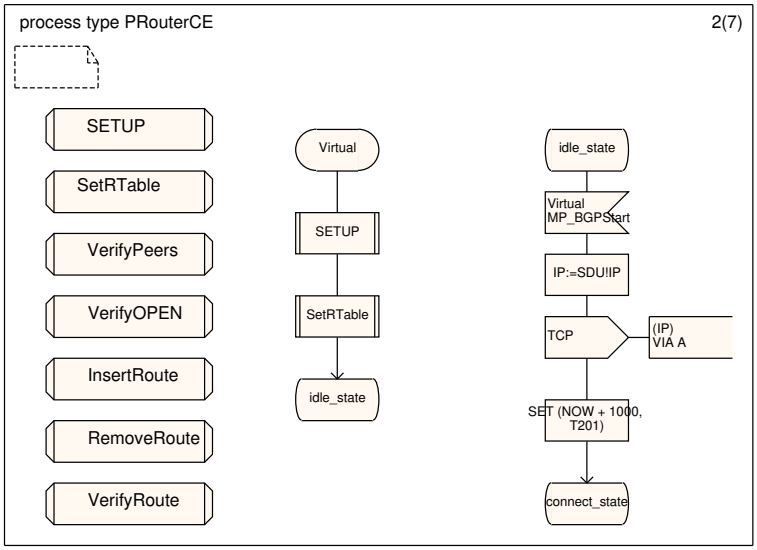


Figura C.33: Estado *idle*, processo *prouter-ce* do bloco *routermpbgp-ce*

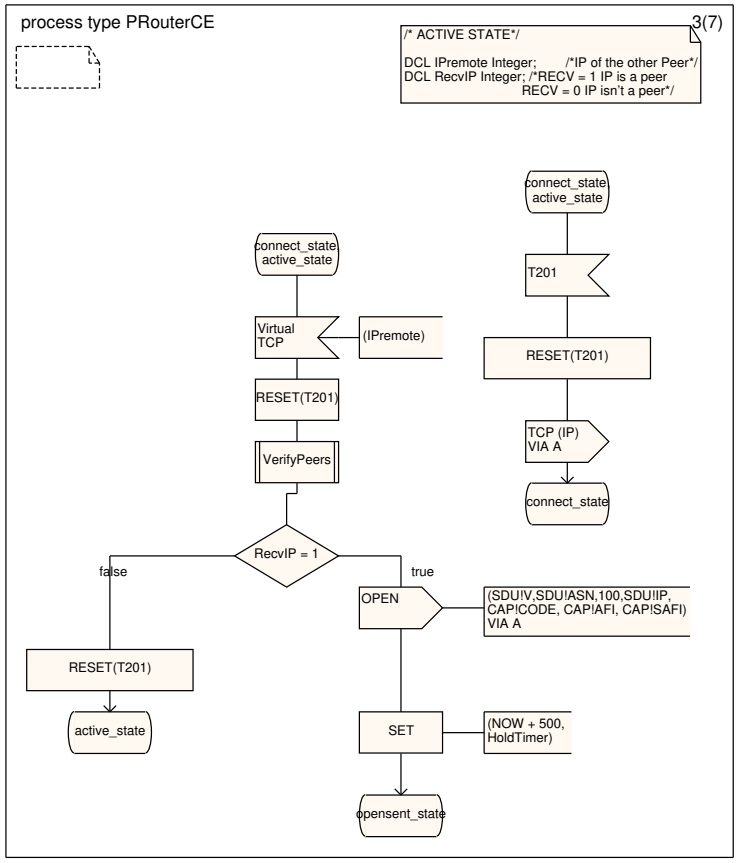


Figura C.34: Estado *connect e active*, processo *prouter-ce* do bloco *routermpbgp-ce*

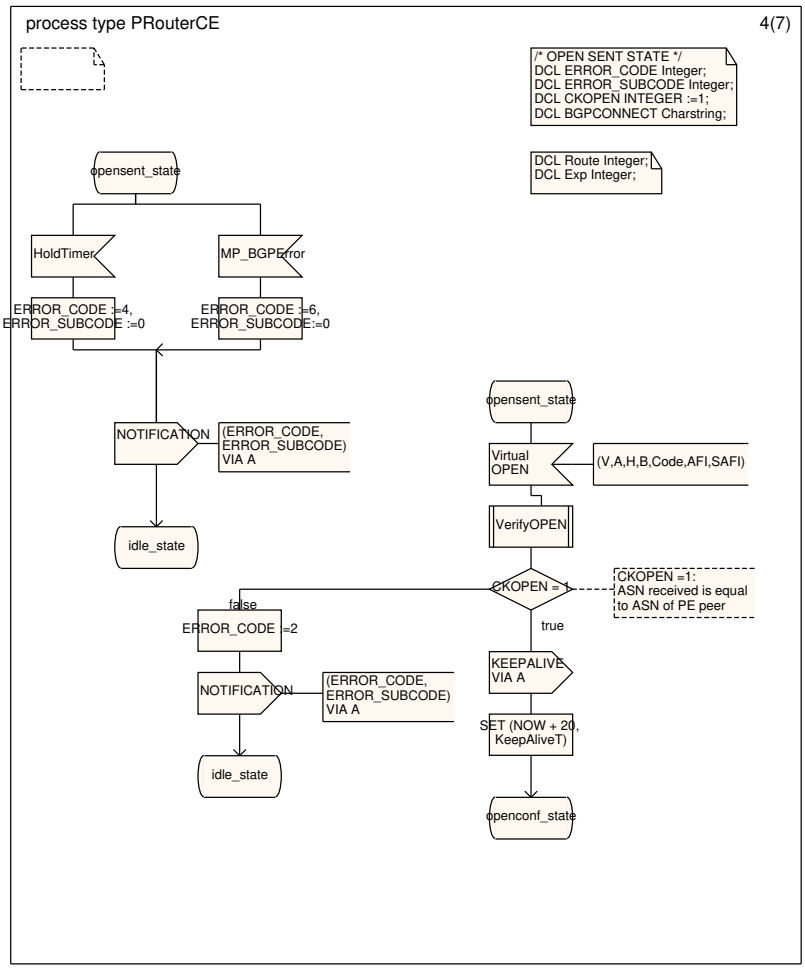


Figura C.35: Estado *open sent*, processo *prouter-ce* do bloco *routermpbgp-ce*

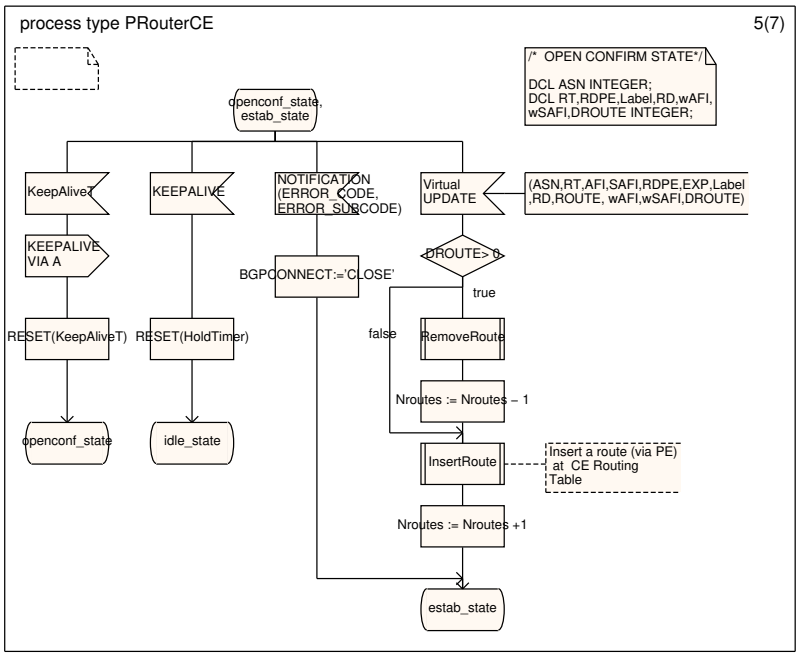


Figura C.36: Estado *open confirm* e *established*, processo *prouter-ce* do bloco *routermpbgp-ce*

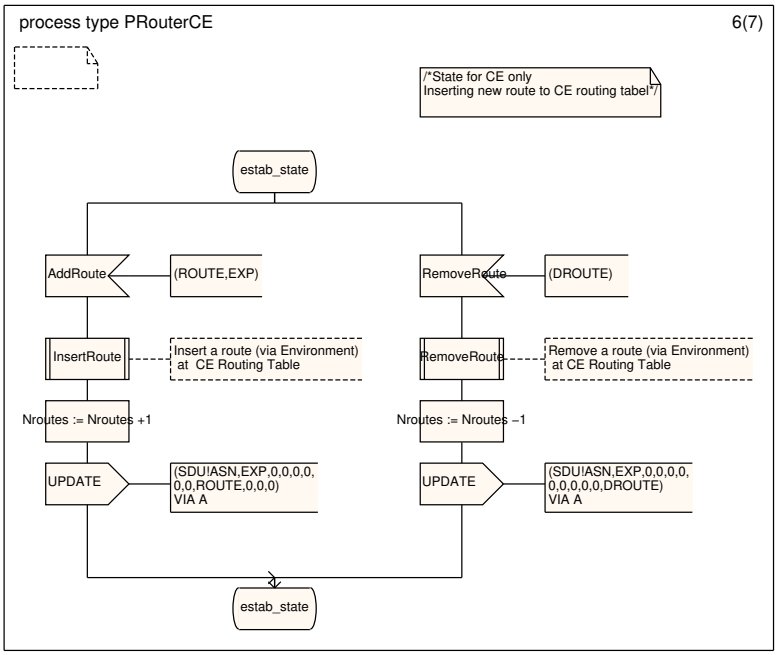


Figura C.37: Estado *established*, processo *prouter-ce* do bloco *routermpbgp-ce*

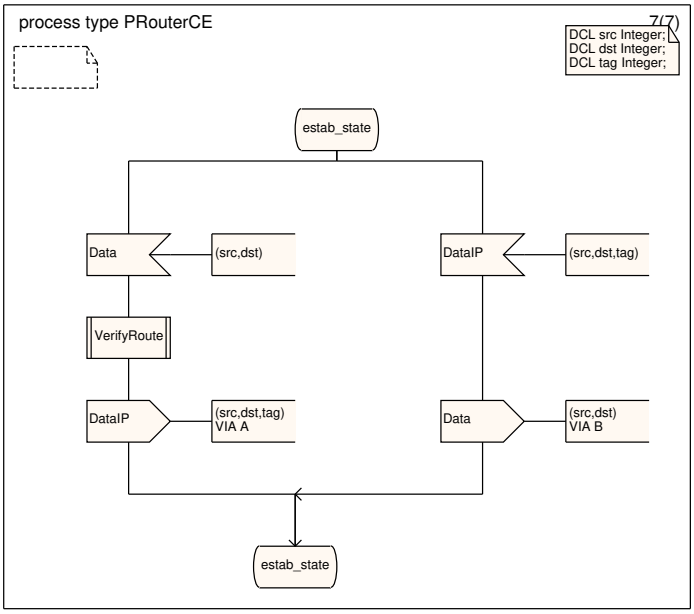


Figura C.38: Estado *established*, processo *prouter-ce* do bloco *routermpbgp-ce*

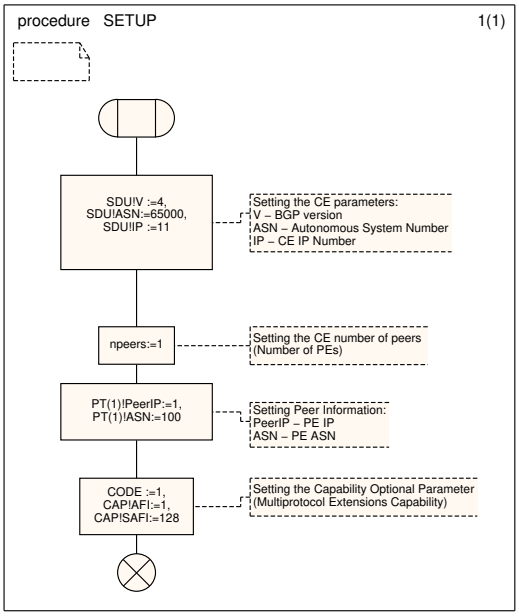


Figura C.39: Procedimento *setup* do processo *prouter-ce*

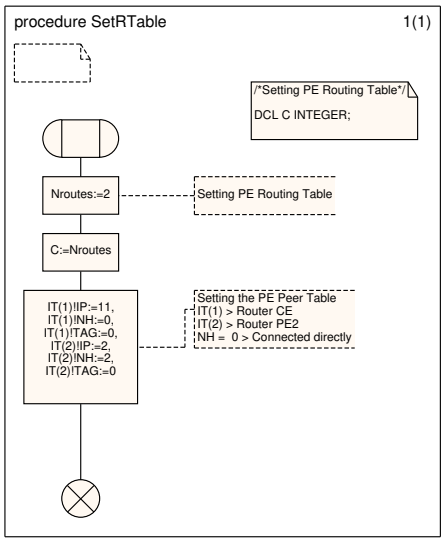


Figura C.40: Procedimento *setrtable* do processo *prouter-ce*

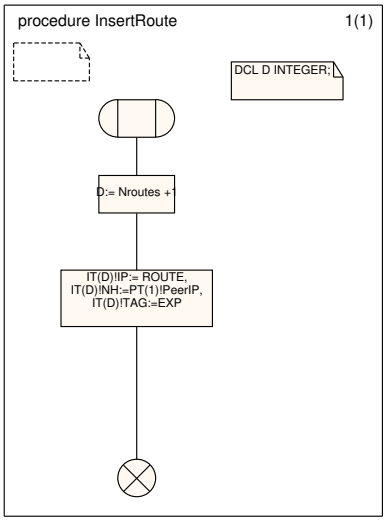


Figura C.41: Procedimento *insert-route* do processo *prouter-ce*

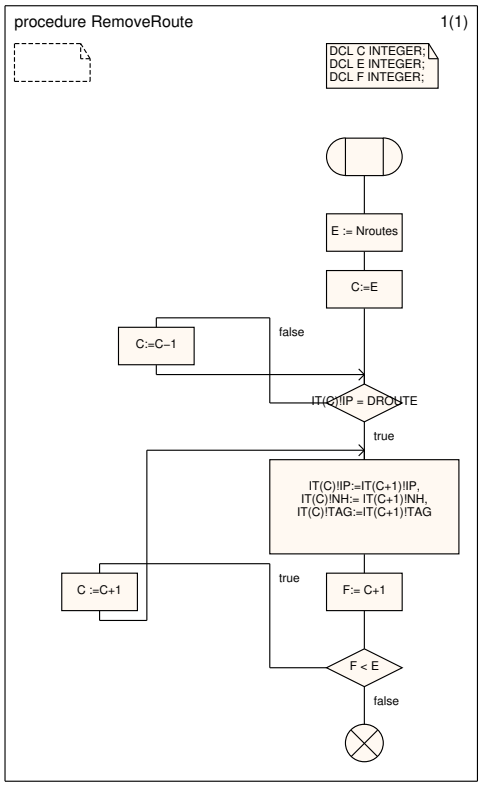


Figura C.42: Procedimento *remove-route* do processo *prouter-ce*

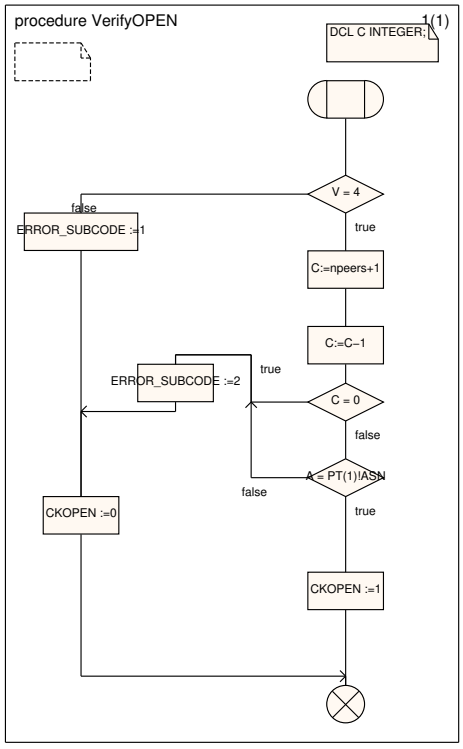


Figura C.43: Procedimento *verify-open* do processo *prouter-ce*

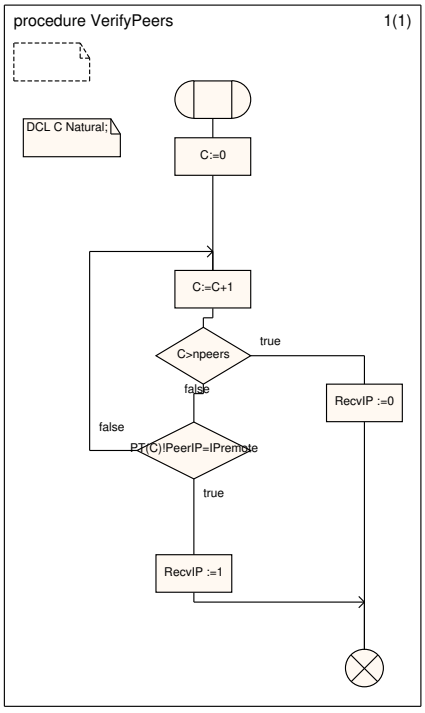


Figura C.44: Procedimento *verify-peers* do processo *prouter-ce*

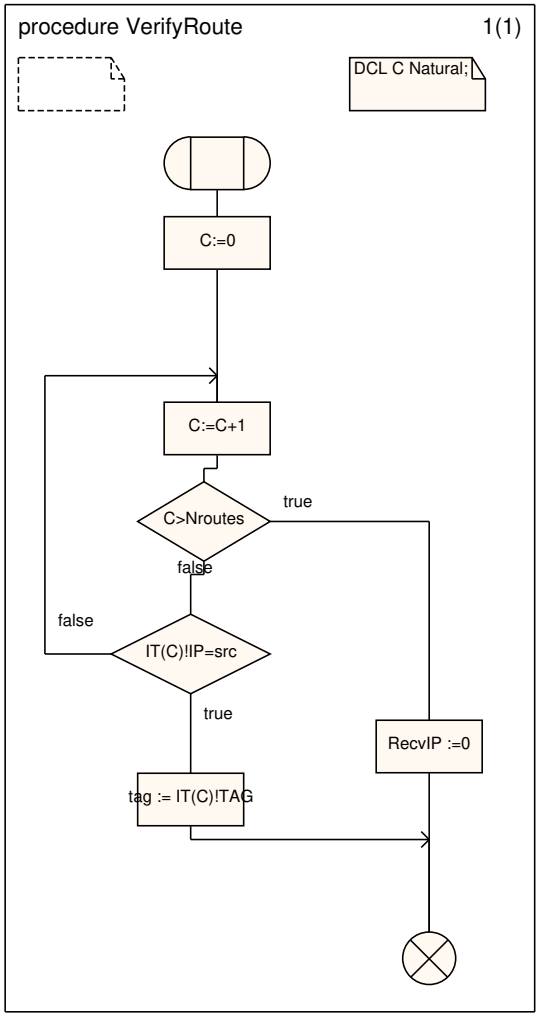


Figura C.45: Procedimento *verify-route* do processo *prouter-ce*

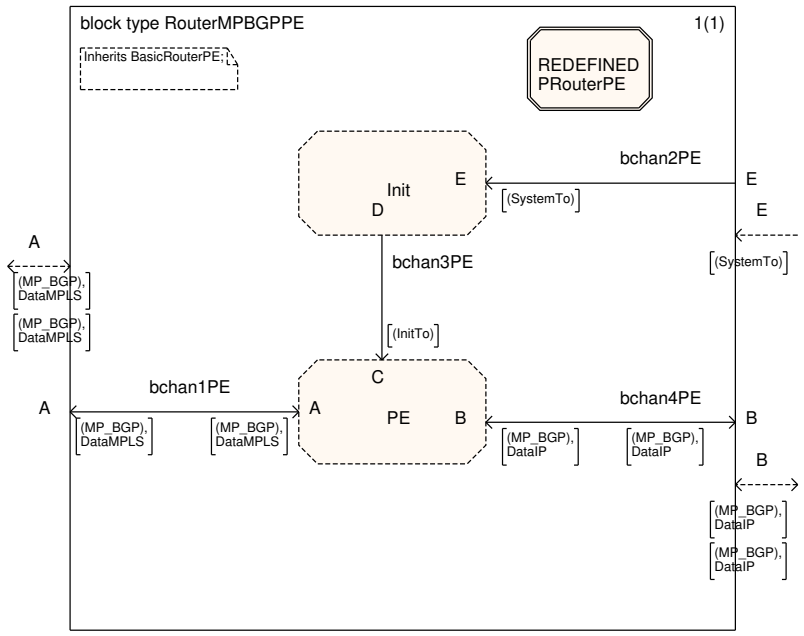


Figura C.46: Bloco *routermpbgp-pe* do sistema *basicarquitecture-scenario2*

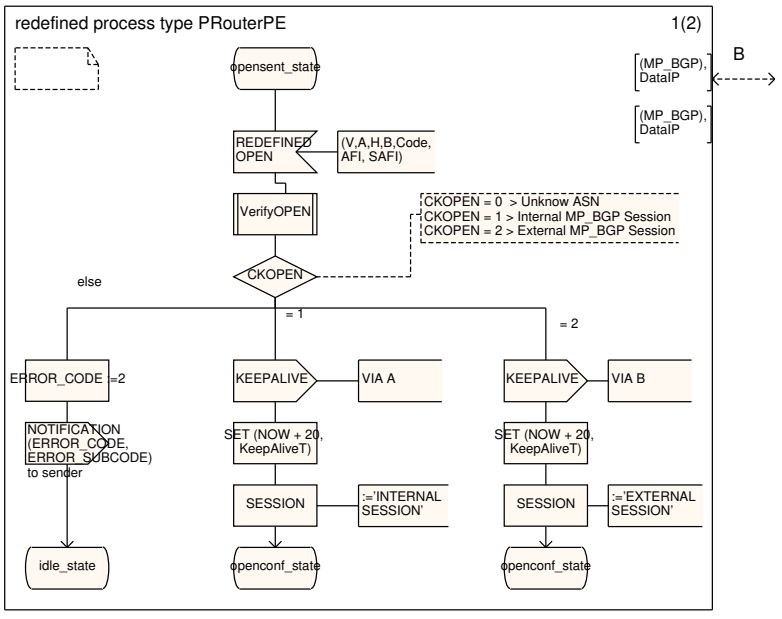


Figura C.47: Redefinição do processo *prouter-pe* - estado *open sent*

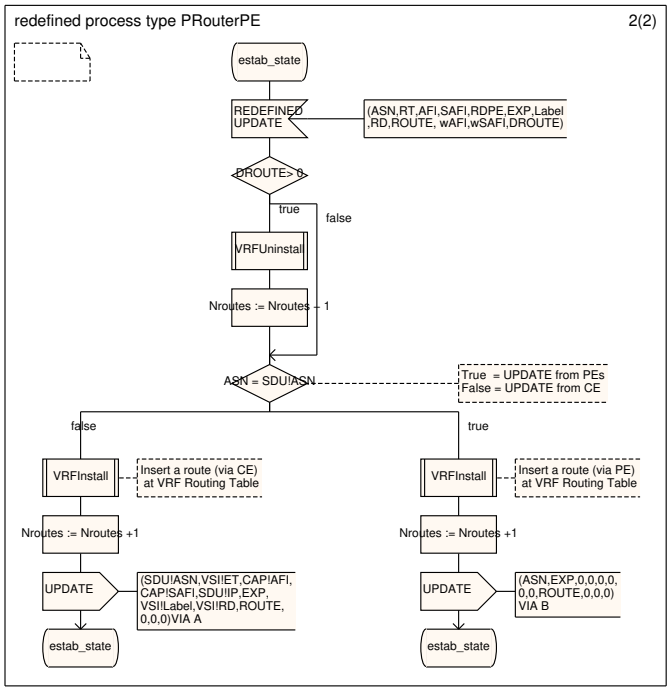


Figura C.48: Redefinição do processo *prouter-pe* - estado *established*