PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

FACULDADE DE ENGENHARIA E FACULDADE DE INFORMÁTICA

CURSO DE ENGENHARIA DE COMPUTAÇÃO

Lucas Bondan

Rodrigo Celso Gobbi

Vinícius Morais Fochi

O PROTOCOLO DE AGREGAÇÃO DE ENLACES LACP:
Um Simulador e o Protocolo de Marcação de Pacotes

Trabalho de Conclusão de Curso

Orientador: Ney Laert Vilar Calazans

Porto Alegre

2011/02

LISTA DE FIGURAS

LISTA DE SIGLAS

EIA	Electronic Industries Alliance
FDS	File Descriptor Socket
GPL	General Public License
HAL	Hardware Abstraction Layer
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
ISO	International Organization for Standardization
ITU	International Communication Union
LAC	Link Aggregation Control
LACP	Link Aggregation Control Protocol
LACPDU	Link Aggregation Control Protocol Data Unit
LAG	Link Aggregation Group
LAN	Local Area Network
MAC	Media Access Control
MGN	Management
MPDU	Marker Protocol Data Unit
MRPDU	Marker Response Protocol Data Unit
MUX	Mux Machine
NTT	Need To Transmit
OSI	Open Systems Interconnection
PC	Personal Computer
PUCRS	Pontifícia Universidade Católica do Rio Grande do Sul
PTX	Periodic Transmission Machine
RFC	Request For Comments
RX	Receive Machine
SLOGIC	Selection Logic
TX	Transmit Machine
UCT	Unconditional Transition

SUMÁRIO

1 INTRODUÇÃO

Na área de telecomunicações, cada vez mais se faz necessário o uso de tecnologias capazes de disponibilizar maior confiabilidade, interoperabilidade, robustez e velocidade entre diferentes pontos de redes de comunicação. Tais tecnologias são desenvolvidas a partir de *hardware* utilizado para teste e validação do processo de desenvolvimento de novos produtos, capazes de disponibilizar recursos aos serviços que estão sendo desenvolvidos.

Com relação a infraestruturas de rede de comunicação de médio e grande porte, frequentemente encontramos equipamentos conectados diretamente uns aos outros (*switches, servers* e *routers),* seja para aumentar o número de hospedeiros conectados a uma rede, seja para diminuir a carga em um equipamento. Mas o principal objetivo de redes de comunicação é manter a comunicação ocorrendo com um mínimo possível de oscilações possíveis, ou seja, que a conexão mantenha-se estável. Mas quanto maior a demanda de banda pelos usuários de uma rede, maior a necessidade de gerenciamento desta para que permaneça em pleno funcionamento. Para garantir isto, a prática de políticas que avaliem a demanda com base na topologia de rede facilita o processo de otimização de recursos. Em algumas situações, essas políticas não aumentam o desempenho, o que torna necessária a substituição de toda a rede ou a duplicação da infraestrutura desta. Em uma camada transparente ao usuário, existem protocolos que auxiliam o gerente na manutenção da rede e garantem o bom funcionamento desta última. Estes protocolos podem ter diversas finalidades, tais como descobrir a topologia e os serviços disponíveis, evitar laços entre as conexões, gerenciar mensagens destinadas a determinados grupos de hospedeiros ou ainda a configuração de canais de comunicação especiais entre dois equipamentos.

Geralmente, protocolos como os acima mencionados são descritos em documentos específicos, tais como uma *Request for Comments* (RFC) ou em padrões geridos por algum órgão regulador. Na área de redes de computadores, uma RFC é um documento publicado pela *Internet Engineering Task Force* (IETF) que descreve métodos, comportamentos ou inovações aplicáveis ao funcionamento da *Internet* e sistemas a ela conectados. O IETF adota algumas das propostas publicadas em RFCs como padrões usados em diversas redes de comunicação. Existem órgãos especializados por

normalizar padrões. Alguns destes são: o *Institute of Electrical and Electronics Engineers* (IEEE); a *International Organization for Standardization* (ISO); a *International Telecommunication Union* (ITU); a *Electronic Industries Alliance* (EIA).

A ISO foi uma das organizações pioneiras na definição formal de conexão de computadores. A arquitetura por ela definida recebeu o nome de *Open Systems Interconnection* (OSI). Essa arquitetura define um modelo que estratifica redes de comunicação em sete camadas de protocolos distintas, definindo um conjunto de níveis de abstração usados para organizar hierarquias de redes e facilitar a interoperabilidade de equipamentos de múltiplos fabricantes e fornecedores de serviços de rede. Em cada camada encontram-se protocolos que atendem a determinadas classes de funcionalidades. A Figura 1 ilustra a disposição hierárquica das sete camadas do modelo OSI [WIK11C].

Figura 1: Disposição das camadas do modelo OSI da ISO.

Nas camadas física e de enlace do modelo OSI, integrando as duas camadas, encontra-se a tecnologia *Ethernet*, uma tecnologia de interconexão de rede local (*Local Area Network*) ou LAN, baseada no envio de pacotes de tamanho variável. Esta tecnologia define cabeamento e sinais elétricos para a camada física, o formato de pacotes e protocolos para a camada de enlace, que nesta tecnologia inclui o controle de acesso ao meio (*Media Access Control*) ou camada MAC *Ethernet*. Hoje, a tecnologia

Ethernet está presente em praticamente todos os dispositivos de redes de comunicação. Sendo assim, novas tecnologias vêm sendo desenvolvidas para aprimorar o funcionamento da *Ethernet*, e dentro destas novas tecnologias encontram-se protocolos responsáveis por agregar novas funcionalidades a esta tecnologia. Estudos vêm sendo realizados com o objetivo de expandir o escopo local da tecnologia Ethernet, criando a rede Ethernet metropolitana (*Metro Ethernet Network*) ou MEN, tornando-a capaz de estender os serviços de redes Ethernet para áreas metropolitanas ou subúrbios de cidades [HUY11].

Dentre as tecnologias em desenvolvimento, pode-se citar a tecnologia de agregação de enlaces. Agregação é a capacidade que um equipamento de rede de comunicação possui de unir enlaces físicos, criando um único enlace virtual. Em meados de 1990, várias empresas começaram a produzir *switches* capazes de realizar agregações entre si, tornando possível aumentar a largura de banda entre seus *switches.* Porém, cada fabricante possuía seu método proprietário para realizar a agregação, tornando difícil a configuração de agregações entre *switches* de diferentes fabricantes. Visando um método comum de configuração de agregações, para equipamentos de quaisquer fabricantes, em novembro de 1997 um grupo do IEEE denominado IEEE 802.3 se reuniu e decidiu incluir um recurso de configuração automática entre *switches*, que traria também a possibilidade de redundância na conexão. Esse recurso ficou então conhecido como *Link Aggregation Control Protocol* (LACP) [WIK11A].

A Figura 2 ilustra a disposição do LACP em relação ao modelo de referência OSI.

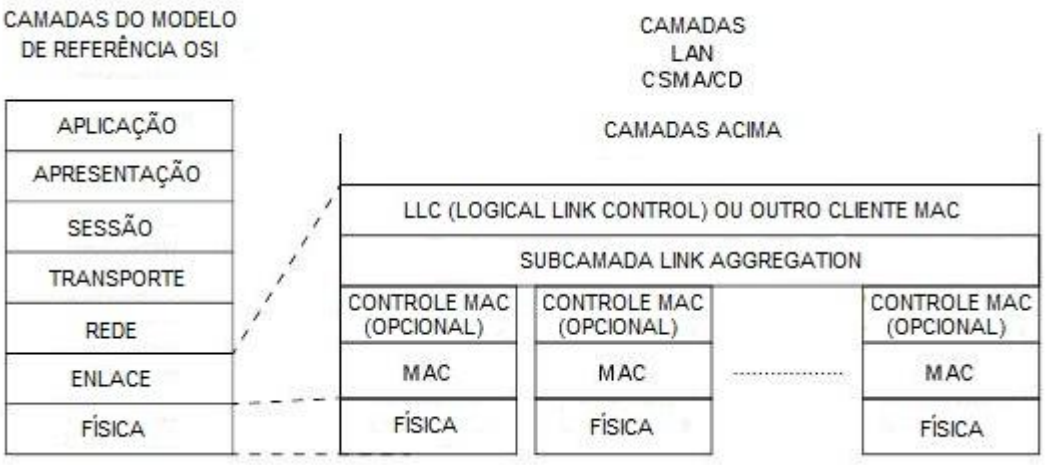


Figura 2: Disposição do LACP no modelo de referência OSI [IEE08].

Existe sempre demanda pelo desenvolvimento de novas soluções para os problemas na área de redes de comunicação. Isto se deve em parte ao alto custo do *hardware* necessário para a execução de determinados serviços. LACP é um destes serviços. Neste contexto, propõe-se aqui a implementação de um simulador capaz de permitir exercitar algumas funções desempenhadas pelo *hardware* de um *switch*, dotado da funcionalidade do protocolo de controle de agregação de enlaces LACP sem a utilização de um *hardware* específico para testes e validação. Também faz parte deste trabalho a implementação do protocolo de marcação de pacotes, denominado *Marker Protocol*, que é importante para o funcionamento correto do algoritmo de agregação.

A implementação consiste na codificação em linguagem de programação C das máquinas de estados e funções do algoritmo que descreve o funcionamento do protocolo, substituindo as chamadas de funções desempenhadas especificamente pelo *hardware* por funções que simulam o funcionamento deste. Assim, o algoritmo desenvolvido independe do dispositivo onde o mesmo será finalmente implantado, pois foi desenvolvido de forma genérica, necessitando apenas poucos ajustes quando migrado para diferentes dispositivos.

Baseado na técnica denominada de *Channel Bonding* [WIK11B], o LACP torna possível a criação de um enlace virtual, formado por duas ou mais conexões físicas reais. O diferencial do LACP está na possibilidade de configurar o canal virtual inserindo as configurações em apenas um dos lados da conexão, o que é uma vantagem significativa em relação a técnicas mais primitivas de *Channel Bonding*, onde os dois lados da conexão precisam ser configurados. O LACP traz ainda outros benefícios, tais como aumento da largura de banda, largura de banda linearmente incrementável (aumento da banda de acordo com a necessidade), maior disponibilidade da conexão (tempo em que a conexão permanece operacional a seus hospedeiros), balanceamento de carga entre enlaces físicos, configuração automática de agregações e configuração e reconfiguração rápida. Hoje, estes benefícios são altamente requisitados para tornar infraestruturas de redes mais robustas, confiáveis e rápidas. Descrito em [IEE08], o LACP trabalha habilitando a agregação de um ou mais enlaces, formando assim um grupo de agregação de enlaces (*Link Aggregation Group*) ou LAG, de tal forma que o cliente MAC possa tratar o LAG como se fosse um único enlace.

Existe hoje no mercado grande interesse nesse protocolo por empresas que atuam no mercado de equipamentos para redes de comunicação. Empresas como 3Com, Bay Networks, Cisco Systems, Extreme Networks, Hewlett-Packard e Sun, já possuem suas tecnologias de agregação de enlaces proprietárias. Na esfera nacional essa tecnologia ainda é escassa e empresas que detém essa tecnologia possuem diferencial, por isso a importância do desenvolvimento do protocolo LACP.

2 MOTIVAÇÃO E OBJETIVOS

O objetivo estratégico deste trabalho é aplicar os conhecimentos adquiridos ao longo do curso de graduação em Engenharia de Computação, tais como: programação utilizando a linguagem C, conceitos de programação orientada a objetos, domínio de estruturas de dados, manipulação de infraestruturas e protocolos de redes de comunicação, operação e configuração de equipamentos de *hardware*, organização e arquitetura de computadores no escopo de redes de comunicação, e programação paralela e concorrente.

Outro objetivo importante a ser atingido ao longo da realização deste trabalho é adquirir experiência prática no decorrer de seu desenvolvimento, uma vez que a área de redes de comunicação é um campo amplo, que desperta interesse dos autores como possível área de atuação profissional. A área de redes de comunicação, por estar em grande expansão, buscando novos produtos e serviços, motiva ir além de conceitos básicos para resolver questões encontradas na área, buscando novas informações e formas mais eficientes de resolver problemas relacionados ao desenvolvimento de novas tecnologias.

O presente trabalho vem sendo desenvolvido no contexto de uma parceria entre a Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) e a empresa Parks S/A Comunicações Digitais. Juntas, universidade e empresa montaram o Laboratório de Inovação Tecnológica, coordenado por membros do Grupo de Apoio ao Projeto de Hardware (GAPH), grupo de pesquisa com interesse em telecomunicações. Os objetivos da criação do laboratório vão de encontro aos da universidade e da empresa, pois o mesmo gera conhecimento e fornece experiência adicional aos estagiários, além de

produzir protótipos de produtos e potencial mão-de-obra para a empresa. Por ter esse vínculo com uma empresa privada, os produtos que são gerados aqui estão sujeitos a prazos para estarem no mercado, o que também se aplica ao nosso projeto.

Com o término deste trabalho, pode-se validar o funcionamento do protocolo de agregação de enlaces LACP em ambiente simulado, funcionando em conjunto com o protocolo de marcação de pacotes. Uma vez validado, o protocolo agora pode ser migrado para a plataforma de *hardware* da empresa envolvida. Espera-se que o protocolo aqui desenvolvido necessite apenas de pequenos ajustes para operar na plataforma final. O benefício primordial da realização deste trabalho será o conhecimento adquirido ao longo do desenvolvimento nas áreas de abrangência da Engenharia de Computação.

3 REFERENCIAL TEÓRICO E PRÁTICO

Definido como uma solução complementar para redes metropolitanas [HUY11], o LACP trabalha utilizando algoritmos de troca de mensagens, a fim de fazer com que os dois lados da conexão se identifiquem e mantenham informações atualizadas um sobre o outro. Diversas definições e identificadores são necessários para que as operações de agregação e desagregação, entre outras, sejam executadas corretamente. Descreve-se a seguir um conjunto mínimo de tais conceitos.

3.1 Conceitos Básicos

O endereço MAC de um *switch* é utilizado para identificá-lo como sistema. O sistema parceiro é o *switch* ao qual o sistema atual está conectado e que está disposto a trabalhar com o LACP. Este trabalha trocando informações entre dois *switches*, estabelecendo uma comunicação entre os dois sistemas, para que cada um conheça e concorde com as informações de seu sistema parceiro. Cada porta pertencente a um dado sistema denomina-se um *ator* e cada porta do sistema parceiro ligada a um ator denomina-se um *parceiro* do sistema em questão. Sendo assim, cada sistema deve armazenar informações de seus atores e respectivos parceiros e estas informações serão utilizadas para a configuração das agregações [IEE08]. A Figura 3 ilustra o conceito de sistema, ator e parceiro, partindo de uma análise do ponto de vista do *Switch* 1.

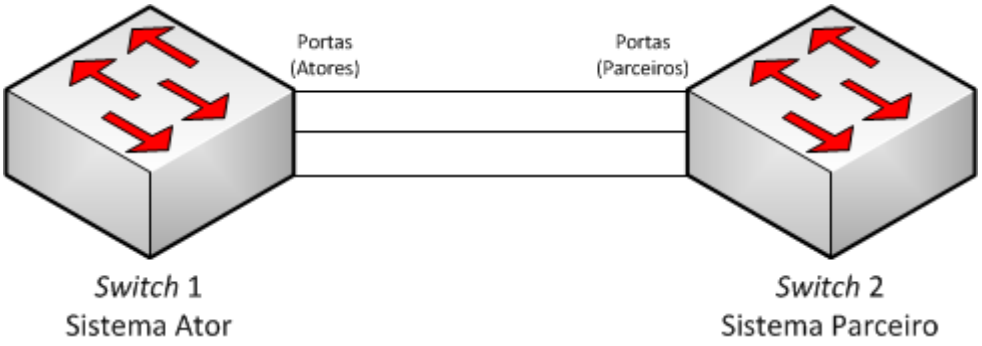


Figura 3: Sistema ator e sistema parceiro na visão do Switch 1 [IEE08].

Da mesma forma que o *Switch* 1 da Figura 1 enxerga o *Switch* 2 como seu sistema parceiro, o *Switch* 2 enxerga o *Switch* 1 como seu sistema parceiro.

A troca de informações entre os sistemas ator e parceiro se dá através de pacotes especiais do tipo *Link Aggregation Control Protocol Data Unit* (LACPDU). Esses pacotes contêm as informações de ator e parceiro presentes em cada sistema, ou seja, o sistema ator monta um LACPDU preenchendo os campos referentes ao ator com as informações da porta que está enviando o pacote; nas informações de parceiro, preenche com as informações atuais que ele possui referentes ao parceiro do ator em questão. Neste pacote também está presente o tempo máximo de entrega de um pacote pela função de coleta de pacotes. Esta informação não é necessária para o controle do LACP, mas sim para o controle do algoritmo de distribuição de pacotes presente no *switch*. A Figura 4 ilustra os campos de um pacote do tipo LACPDU.

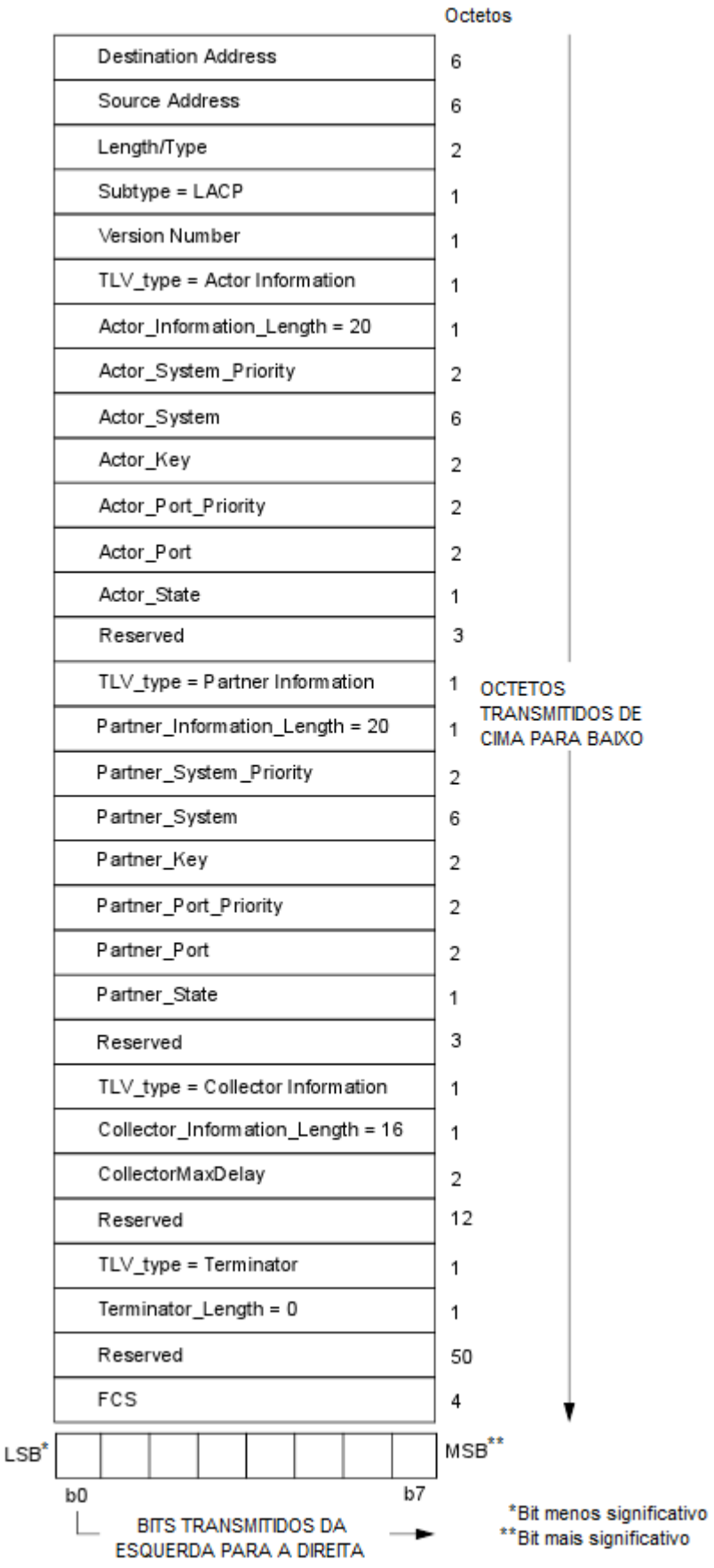


Figura 4: Campos de um LACPDU [IEE08].

Pacotes do tipo LACPDU são um subtipo da classe de protocolos denominados *Slow Protocol*, que determina que a quantidade de pacotes transmitidos dentro do intervalo de um segundo seja de no máximo dez pacotes [IEE07]. Para armazenar as informações referentes aos LACPDUs trocados pelo LACP, foi criada uma estrutura contendo todos os campos vistos na Figura 4. Tal estrutura é mostrada na

```
/* LACPDU */
typedef struct lacpdu_tag {
    struct ether_header eth_h;
    u_int8_t subtype;
    u_int8_t version;
    u_int8_t TLV_type_a; /* Actor Information */
    u_int8_t actor_info_len; /*  = 20 */
    u_int16_t actor_sys_prio;
    u_int8_t  actor_system[ETH_ALEN];  /* Actor System ID */
    u_int16_t actor_key;
    u_int16_t actor_port_prio;
    u_int16_t actor_port;
    u_int8_t actor_state;
    u_int8_t reserved_a[3];
    u_int8_t TLV_type_p; /* Partner Information */
    u_int8_t partner_info_len; /* = 20 */
    u_int16_t partner_sys_prio;
    u_int8_t  partner_system[ETH_ALEN];  /* Partner System ID */
    u_int16_t partner_key;
    u_int16_t partner_port_prio;
    u_int16_t partner_port;
    u_int8_t partner_state;
    u_int8_t reserved_p[3];
    u_int8_t TLV_type_c; /* Collector Information */
    u_int8_t collector_info_len; /* = 16 */
    u_int16_t collector_max_delay;
    u_int8_t reserved_c[12];
    u_int8_t TLV_type_t; /* Terminator */
    u_int8_t terminator_len; /* = 0 */
    u_int8_t reserved[50];
} __attribute__ ((__packed__)) lacpdu_t;
```

Figura 5: Estrutura para armazenar dados de um LACPDU.

Para que seja iniciada a troca de informações entre sistemas, é necessário que um destes receba a configuração desejada para a agregação. Com isto, o sistema que recebe a configuração passará do estado *passivo* para o estado *ativo*. Quando no estado passivo, o sistema tem preferência por não transmitir LACPDUs, a menos que seu parceiro esteja em estado ativo. Quando no estado ativo, a preferência do sistema é por transmitir LACPDUs, desconsiderando o estado do sistema parceiro [IEE08].

3.2 Link Aggregation Control Protocol

O LACP, definido em [IEE08], é um algoritmo composto por um conjunto de máquinas de estados e funções. Estas tornam possível, através de trocas de pacotes de controle e/ou configuração entre os sistemas hospedeiros, a união de múltiplos enlaces físicos em enlaces lógicos. Com isso, obtêm-se diversas vantagens na comunicação, tais como aumento da largura de banda, largura de banda linearmente incrementável, maior disponibilidade da conexão e balanceamento de carga entre os enlaces físicos.

O LACP possui algumas limitações, não tendo suporte a algumas características, tais como: agregação multiponto, ou seja, o LACP só atua em conexões ponto-a-ponto (ligação direta entre equipamentos); MACs fora do padrão IEEE 802.3; operação *half-duplex* (dados trafegam apenas em um sentido no canal: envio ou recebimento); operação em diferentes taxas de dados: todos os enlaces sob controle do LACP devem operar na mesma taxa de transferência.

Como mencionado e descrito na Seção 3.1, pacotes trocados pelo protocolo são denominados LACPDUs, e carregam informações sobre o sistema de origem do pacote, informações do ator que está enviando o pacote e informações que o ator de origem tem sobre seu sistema parceiro.

O LACP pode ser inserido em dois tipos de equipamentos de redes distintos: *switches* ou servidores. Seu funcionamento correto depende da garantia de que o protocolo esteja disponível em ambos os lados da conexão. O LACP também aceita diferentes topologias. Pode ser configurado, por exemplo, entre dois *switches Ethernet*, ou entre um *switch Ethernet* e um servidor. Neste trabalho adota-se inicialmente a topologia da Figura 6, que mostra a ligação de dois enlaces físicos entre dois *switches*, onde os dois enlaces foram configurados pelo LACP para formar uma agregação.

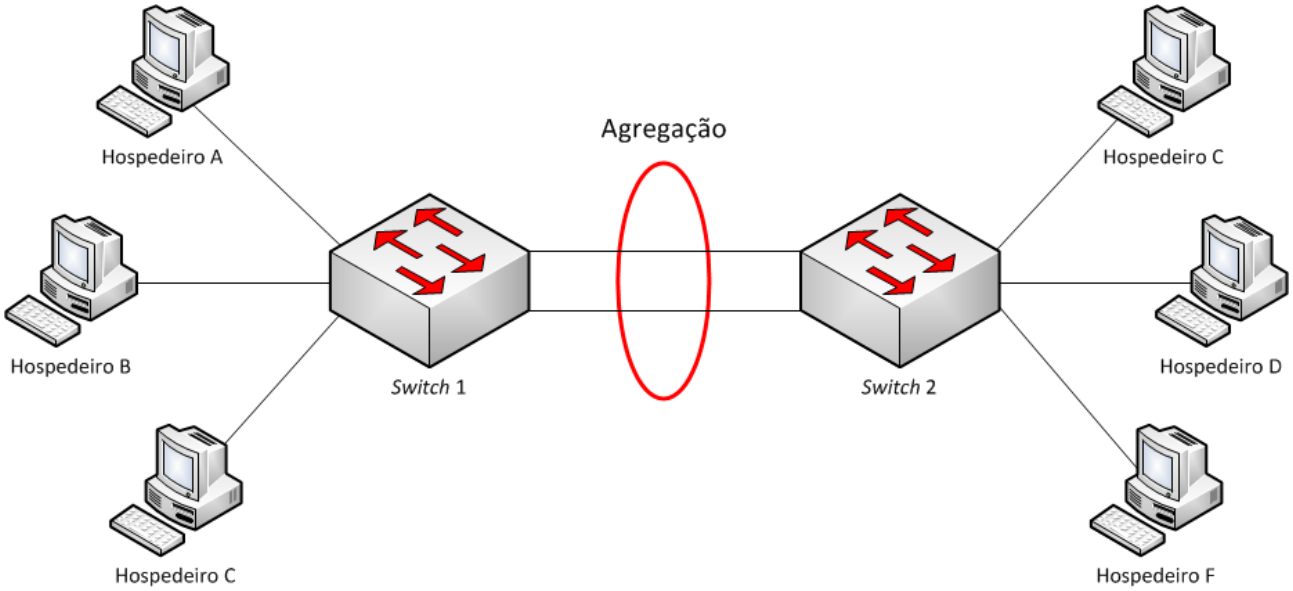


Figura 6: Exemplo de topologia do LACP.

3.2.1 Máquinas de Estado do LACP

Para a implementação do algoritmo, é necessário compreender as seis máquinas de estado que definem de forma resumida o seu funcionamento. Esta seção apresenta estas máquinas de estado, definidas em [IEE08], com uma breve descrição da função desempenhada para cada máquina.

Todas as máquinas de estado do protocolo interagem entre si por meio de determinadas variáveis. Tais variáveis sinalizam qual dos estados de cada máquina entrará em execução, determinando quais funções da máquina em questão serão executadas. A Figura 7 ilustra as iterações entre as máquinas de estado do algoritmo, bem como os sinais de controle por elas utilizados.

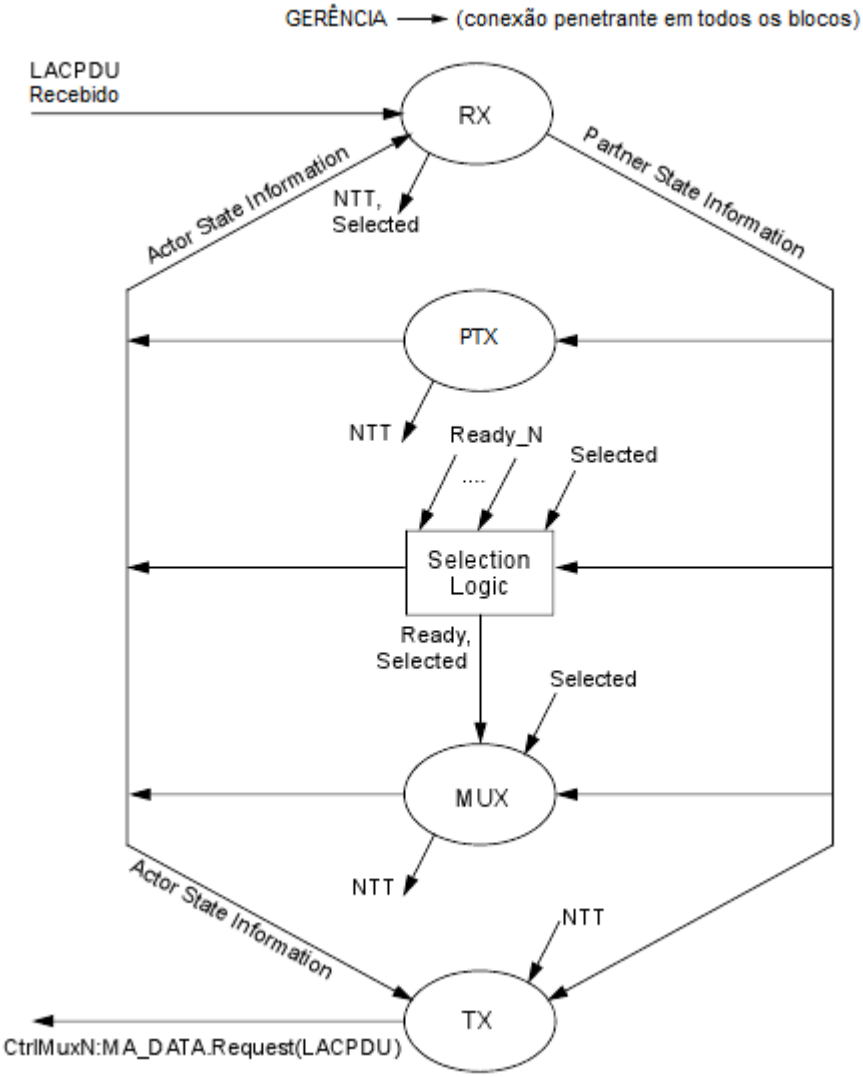


Figura 7: Ilustração das iterações das Máquinas de Estado do LACP [IEE08].

Na Figura 7, cada circulo representa uma das máquinas de estado do protocolo (nota-se que a *Selection Logic* é representada por um quadro, pois esta não é caracterizada como uma máquina de estados, mas sim como um conjunto de funções). As setas que saem dos círculos representam as variáveis que são alteradas pela máquina e que interferem no comportamento das demais. As variáveis que alteram o comportamento das máquinas são as setas que entram em cada círculo.

Além das operações de gerência, outro fator que induz a operação do protocolo é a chegada de um LACPDU. Este ativa primeiramente a *Receive Machine* (RX) que, dependendo dos dados contidos no LACPDU, pode gerar sinais que induzam a operação de outras máquinas de estados. Ao final da execução do LACP decorrido do recebimento de um LACPDU, dependendo da sequência de operações das máquinas de estados, a

Transmit Machine (TX) pode vir a enviar um LACPDU em resposta ao LACPDU recebido, encerrando a sequência de operações do protocolo em decorrência do recebimento de um LACPDU.

No trabalho de Molina [MOL11], cada uma das máquinas de estados do LACP foi implementada através de uma *thread*. Assim, cada uma das máquinas possui funcionamento desacoplado do algoritmo principal (*main*), tornando-se mais independentes entre si. Porém, por executarem em paralelo, foram necessárias estruturas de controle de acesso à regiões críticas, conhecidas como “mutex”. Essas estruturas servem para evitar leituras e escritas mútuas à variáveis compartilhadas pelas máquinas de estado.

3.2.1.1 Receive Machine (RX)

No início de sua execução, a *Receive Machine* ou RX carrega os valores padrão de informações de parceiro. No decorrer de sua execução, é ela a responsável pela captura dos LACPDUs. Esta máquina também é responsável por armazenar as informações presentes nos campos de ator do LACPDU recebido nos respectivos campos locais de parceiro. A RX também compara as informações recebidas no LACPDU com as informações locais. Caso as informações recebidas nos campos de parceiro do LACPDU coincidam com as informações locais de ator, assume-se que os sistemas estão em sincronia. Caso contrário, é gerado um sinal de NTT. Outra função exercida pela RX é a de comparar as informações de ator recebidas no LACPDU com as informações locais de parceiro. Se pelo menos uma das informações comparadas for diferente, o sistema reconhece que ainda não foi selecionado um agregador para o ator.

A RX é composta por seis estados distintos, cada qual com seu conjunto de operações. A Figura 8 mostra em detalhes os estados e as operações realizadas em cada estado da RX.

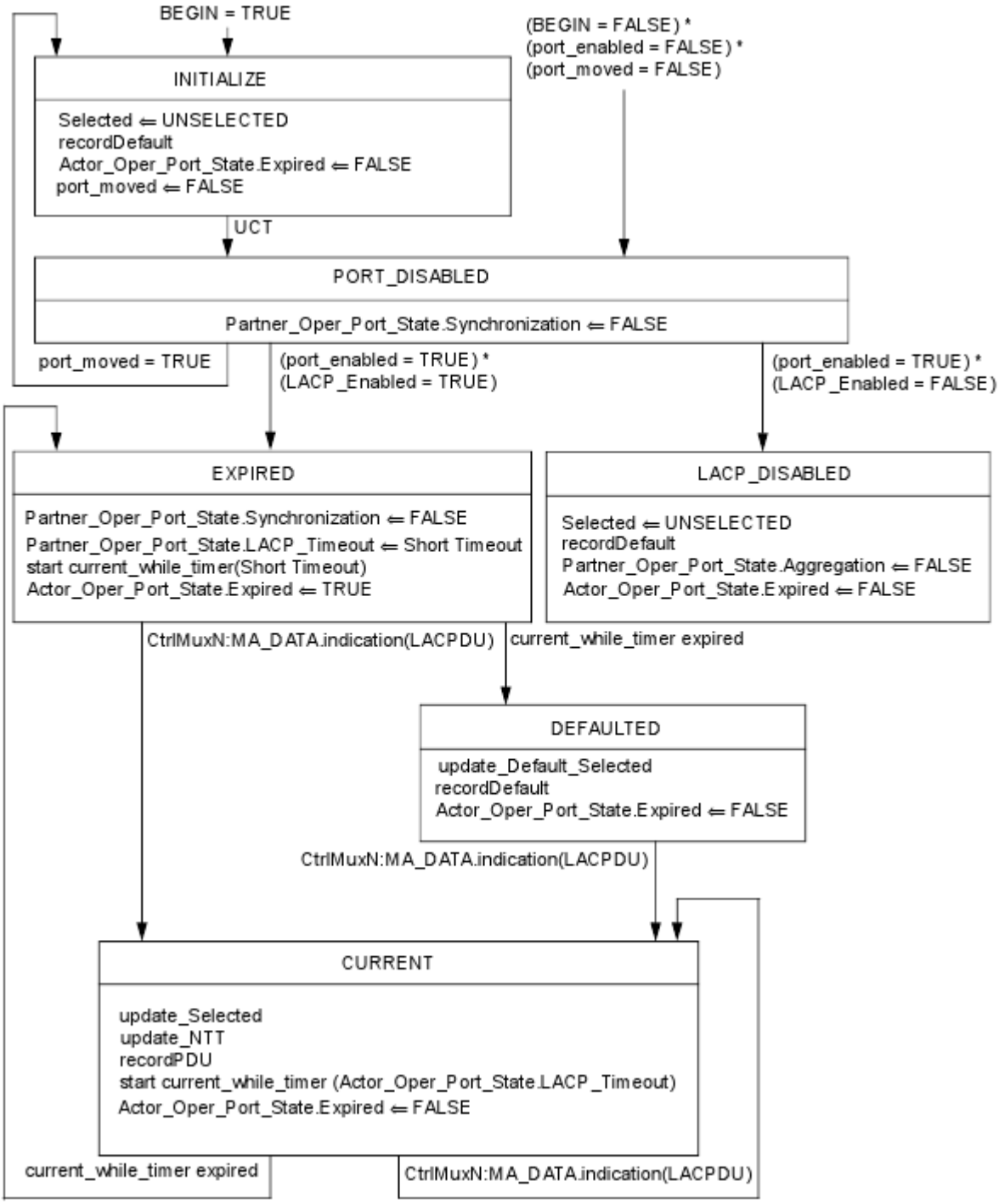


Figura 8: Detalhamento dos estados da *Receive Machine* [IEE08].

Após o sinal de BEGIN se tornar verdadeiro, a RX entra no estado INITIALIZE, onde ela carrega os valores padrão de informações de parceiro, através da função *recordDefault*. Uma Transição Incondicional (*Unconditional Transition*) ou UCT faz com que a *Receive Machine* avance para o estado PORT_DISABLED. Este estado serve apenas para sinalizar que a sincronização com parceiro não está concluída, atribuindo o valor de valso ao bit de sincronismo nas informações locais do parceiro.

No estado PORT_DISABLED, caso o LACP esteja com seu funcionamento desabilitado para a porta em questão (variável LACP_Enabled com valor falso), a RX passa para o estado LACP_DISABLED, onde permanecerá até que a variável BEGIN torne-se verdadeira novamente. Caso contrário, o próximo estado é EXPIRED.

No estado EXPIRED, dentre outras operações, é iniciado o *timer* que, quando excedido, muda o estado para DEFAUTED, onde a função recordDefault é executada novamente, indicando que o LACP está com os valores padrão para a porta em questão. Também é executada a função update_Default_Selected, que compara os valores administrativos dos valores operacionais das informações do parceiro. Caso pelo menos uma dessas informações seja diferente, a variável Selected do parceiro recebe o valor UNSELECTED, indicando que não há agregador selecionado para a porta em questão. Ainda no estado EXPIRED, caso seja recebido um LACPDU antes do *timer* expirar, a RX avança para o estado CURRENT.

No estado CURRENT, dentre outras operações, destacam-se as seguintes: update_Selected: compara as informações de ator recebidas no LACPDU com as informações locais de parceiro. Se pelo menos uma delas divergir, a variável Selected recebe o valor UNSELECTED, caso contrário, seu valor permanece inalterado; update_NTT: compara as informações de parceiro recebidas no LACPDU com as informações locais de ator. Se pelo menos uma delas divergir, a variável NTT recebe o valor TRUE, indicando que há a necessidade de transmitir um LACPDU para o sistema parceiro; recordPDU: armazena as informações de ator recebidas no LACPDU nas respectivas variáveis locais de parceiro. Esta função também atualiza o valor do bit de sincronização do ator local. Caso as informações de parceiro recebidas no LACPDU sejam iguais as informações de ator locais OU o bit de agregação do ator recebido no LACPDU esteja em FALSE (indicando que o parceiro é um enlace individual), o bit de sincronização do ator local recebe o valor TRUE; Também neste estado é inicializado o *timer* que, caso expire, faz com que o estado mude para EXPIRED. Caso chegue um novo LACPDU antes do *timer* expirar, a RX repete seu ciclo de operação, considerando desta vez as informações do LACPDU recebido.

A Figura 9 trás o pseudocódigo da parte principal da RX, detalhando as operações realizadas em cada um de seus estados, com exceção das variáveis de controle externas.

```
enquanto (VERDADEIRO) faça
        caso (estado_rx) em
                INITIALIZE:
                        Selected = UNSELECTED;
                        recordDefault(porta);
                        actor.oper_port_state.Expired = FALSO;
                        port_moved = FALSO;
                        estado_rx = PORT_DISABLED;
                fim;

                PORT_DISABLED:
                        partner.oper_port_state.Synncronization = FALSO;
                        se (port_moved = VERDADEIRO) então
                                estado_rx = INITIALIZE;
                                fim;
                        senão se ((port_enabled = VERDADEIRO) E (LACP_Enabled = VERDADEIRO)) então
                                estado_rx = EXPIRED;
                                fim;
                        senão se ((port_enabled = VERDADEIRO) E (LACP_Enabled = FALSO)) então
                                estado_rx = LACP_DISABLED;
                                fim;
                        fim se;
                fim;

                EXPIRED:
                        partner.oper_port_state.Syncronization = FALSO;
                        partner.oper_port_state.LACP_Timeout = 1;
                        inicia current_while_timer(short timeout);
                        actor.oper_port_state.Expired = VERDADEIRO;
                        se (timer == 0) então           /* timer expirou */
                                estado_rx = DEFAULTED;
                        senão se (recebi_pdu == 1) então
                                estado_rx = CURRENT;
                        senão
                                estado_rx = EXPIRED;
                        fim se;
                fim;

                LACP_DISABLED:
                        Selected = UNSELECTED;
                        recordDefault(porta);
                        partner.oper_port_state.Aggregation = FALSO;
                        actor.oper_port_state.Expired = FALSO;
                fim;

                DEFAULTED:
                        update_Defaulted_Selected(porta);
                        recordDefault(porta);
                        actor.oper_port_state.Expired = FALSO;
                        se (recebi_pdu == 1) então
                                estado_rx = CURRENT;
                fim;

                CURRENT:
                        update_Selected(porta);
                        update_NTT(porta);
                        recordPDU(porta);
                        inicia current_while_timer(actor.oper_port_state.LACP_Timeout);
                        actor.oper_port_state.Expired = FALSO;
                        se (timer == 0) então           /* timer expirou */
                                estado_rx = EXPIRED;
                        senão se (recebi_pdu == 1) então
                                estado_rx = CURRENT;
                        fim se;
                fim;
        fim caso;
fim enquanto;
```

Figura 9: Pseudocódigo da *Receive Machine*.

3.2.1.2 Transmit Machine (TX)

A *Transmit Machine* ou TX tem como responsabilidade enviar LACPDUs sob demanda de outras máquinas de estados ou de princípios periódicos. Esta máquina de estados na verdade se comporta como uma função que é acionada sempre que ocorrer um NTT, sendo este gerado quando um sistema exige a troca de pacotes periódicos para manter

as informações consistentes e atualizadas. É a TX que envia o LACPDU com as informações correspondentes à porta que gerou um NTT, de acordo com as definições do pacote descrito em [IEE07].

Seu funcionamento depende da máquina *Periodic Transmission Machine* ou PTX, pois antes de transmitir um novo pacote, a TX necessita conhecer o estado de periodicidade da porta. Isso se faz necessário quando a máquina periódica se encontra no estado NO_PERIODIC, na qual a TX deve garantir com que nenhum LACPDU seja transmitido. Este cenário é bastante comum em sistemas denominados passivos, na qual a transmissão de pacotes ocorre apenas sob demanda.

Em contra partida, quando as variáveis LACP_Enabled e NTT da porta indicam que o protocolo está presente (LACP_Enabled com valor verdadeiro) e necessitando de uma transmissão de novas informações (NTT com valor verdadeiro), a TX deve garantir que um LACPDU seja transmitido corretamente, sujeito a restrição de que não mais do que três pacotes sejam transmitidos em um intervalo *fast periodic* (um segundo). Este intervalo é representado pelo estado FAST_PERIODIC da PTX. Se este limite estiver em vigor, a transmissão deve ser atrasada até o momento em que a restrição não esteja em vigor. Neste caso, as informações enviadas em um LACPDU correspondem aos valores operacionais da porta no momento da transmissão e não no momento em que o NTT foi gerado. Após transmitir um LACPDU, a variável NTT recebe o valor falso pela própria TX, garantindo que o pacote foi enviado e não há mais a necessidade de transmitir um novo LACPDU.

3.2.1.3 Periodic Transmit Machine (PTX)

A *Periodic Transmission Machine* ou PTX é responsável pela manutenção da agregação. Periodicamente, ela gera um NTT para cada ator vinculado a um agregador, a fim de trocar informações com seus respectivos parceiros, para que estas permaneçam sempre atualizadas. Cada sistema, ator ou parceiro, possui *timers* que controlam a validade das informações que este possui a respeito de cada parceiro. Quando não há troca periódica de informações entre os sistemas ator e parceiro, esses *timers* expiram, fazendo com que o sistema considere que suas informações a respeito de seu parceiro

estão desatualizadas, o que desabilita a agregação. A Figura 10 ilustra detalhadamente os estados da PTX e suas respectivas operações.

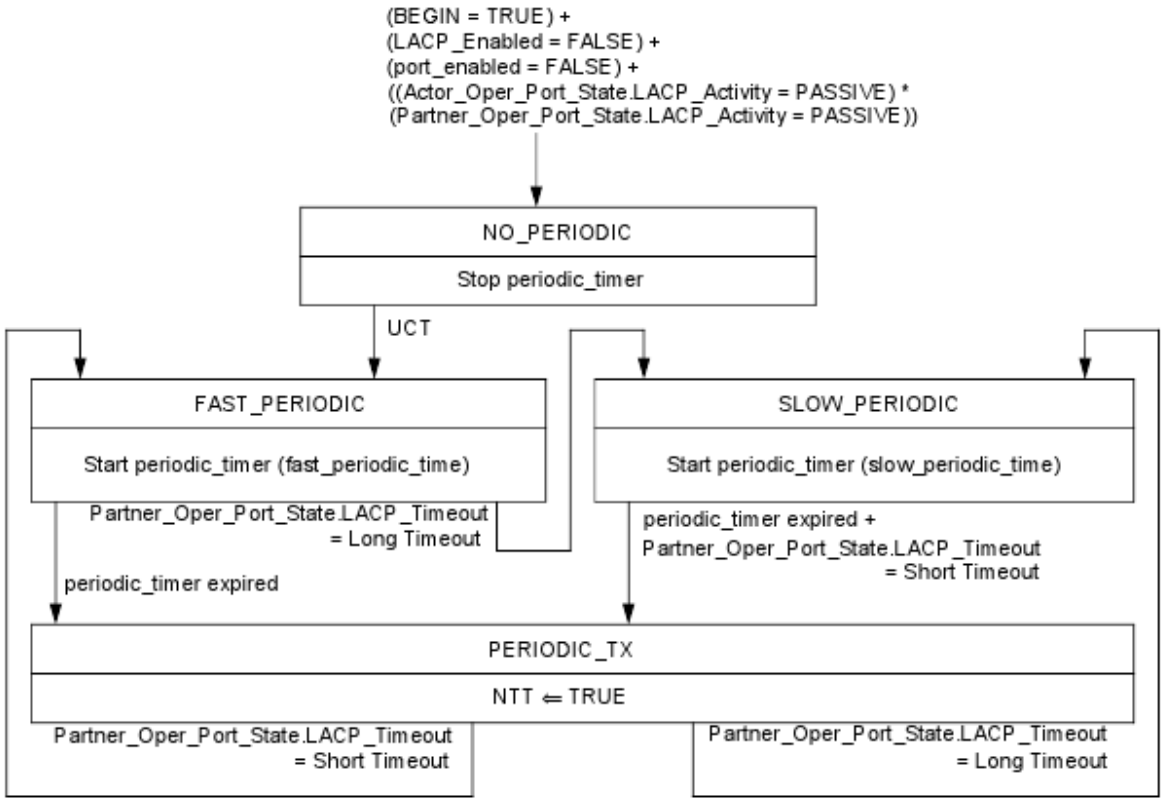


Figura 10: Detalhamento dos estados da *Periodic Transmission Machine* [IEE08].

Depois de iniciada a execução da PTX, devido aos sinais se entrada que podemos observar na Figura 10, esta opera no estado NO_PERIODIC. Neste estado, o *timer* de periodicidade, que indica qual o intervalo de envio entre pacotes, é parado. Após isto, uma UCT leva a PTX para o estado FAST_PERIODIC.

No estado FAST_PERIODIC, as transmissões periódicas estão habilitadas em uma taxa denominada *fast transmission*, na qual os envios de LACPDUs acontecem em um intervalo *fast periodic* (um segundo). Caso o *timer* expire, a PTX avança para o estado PERIODIC_TX, onde um NTT é gerado, requisitando o envio de um LACPDU para pela porta em questão. Caso o bit de LACP_Timeout do parceiro mude de *short* para *long*, a PTX avança do estado FAST_PERIODIC para SLOW_PERIODIC.

Semelhante ao estado FAST_PERIODIC, no estado SLOW_PERIODIC, as transmissões periódicas também estão habilitadas, porém em uma taxa denominada *slow transmission*, na qual os envios de LACPDUs acontecem em um intervalo *slow periodic*

(trinta segundos). A saída deste estado se dá da mesma forma do estado FAST_PERIODIC, porém, não existe transição do estado SLOW_PERIODIC para o estado FAST_PERIODIC sem antes a PTX entrar no estado PERIODIC_TX.

No estado PERIODIC_TX, depois de gerado o sinal de NTT, o qual resultará na criação e envio de um LACPDU, a PTX retorna para o estado FAST_PERIODIC, caso o valor do bit LACP_Timeout do parceiro seja *short timeout* (um); ou avança para o estado SLOW_PERIODIC, caso o valor do bit LACP_Timeout do parceiro seja *long timeout* (zero).

A Figura 11 trás o pseudocódigo da parte principal da PTX, detalhando as operações realizadas em cada um de seus estados, com exceção das variáveis de controle externas.

```
enquanto (VERDADEIRO) faça
        caso (estado_ptx) em
                NO_PERIODIC:
                        parar periodic_timer(porta)             /* Pára o timer periodico */
                        estado_ptx = FAST_PERIODIC;
                fim;

                FAST_PERIODIC:
                        inicia periodic_timer(fast_periodic_timer)      /* inicia timer periodico rápido */
                        se (periodic_timer == 0) então
                                estado_ptx = PERIODIC_TX;
                        senão se (partner.oper_port_state.LACP_Timeout == LONG) então
                                estado_ptx = SLOW_PERIODIC;
                        fim se;
                fim;

                SLOW_PERIODIC:
                        inicia periodic_timer(slow_periodic_timer)      /* inicia timer periodico lento */
                        se ((periodic_timer == 0) OU (partner.oper_port_state.LACP_Timeout == SHORT)) então
                                estado_ptx = PERIODIC_TX;
                        fim se;
                fim;

                PERIODIC_TX;
                        NTT = VERDADEIRO;
                        se (partner.oper_port_state.LACP_Timeout == SHORT) então
                                estado_ptx = FAST_PERIODIC;
                        senão se (partner.oper_port_state.LACP_Timeout == LONG) então
                                estado_ptx = SLOW_PERIODIC;
                        fim se;
                fim;
        fim caso;
fim enquanto;
```

Figura 11: Pseudocódigo da *Periodic Transmission Machine.*

3.2.1.4 Churn Detection Machine (CHURN)

Podem ocorrer situações em que as informações de ator ou parceiro em um sistema sejam perdidas ou corrompidas, o que pode ser causado por perdas de pacotes, por exemplo. A *Churn Detection Machine* ou CHURN detecta esse tipo de erro e altera determinadas informações de ator ou parceiro, a fim de gerar um novo NTT, iniciando uma nova troca de pacotes entre os sistemas, até que estes entrem novamente em acordo. Para isso, a CHURN parte do pressuposto de que se ator ou parceiro

permanecerem muito tempo sem entrar em sincronismo, deve-se alertar o sistema de que algo está errado.

A implementação da CHURN é opcional, pois não é necessário um controle auxiliar sobre as informações contidas nos sistemas. Porém, quando implementada, facilita a manutenção das agregações, uma vez que detecta possíveis problemas nas informações presentes em cada sistema, iniciando uma nova comunicação para ajustar as informações corrompidas.

A CHURN possui duas implementações: uma que monitora o sincronismo dos atores e outra que monitora o sincronismo dos parceiros de cada porta. A Figura 12 e a Figura 13 detalham, respectivamente, os estados e as operações das CHURNs de ator e parceiro.

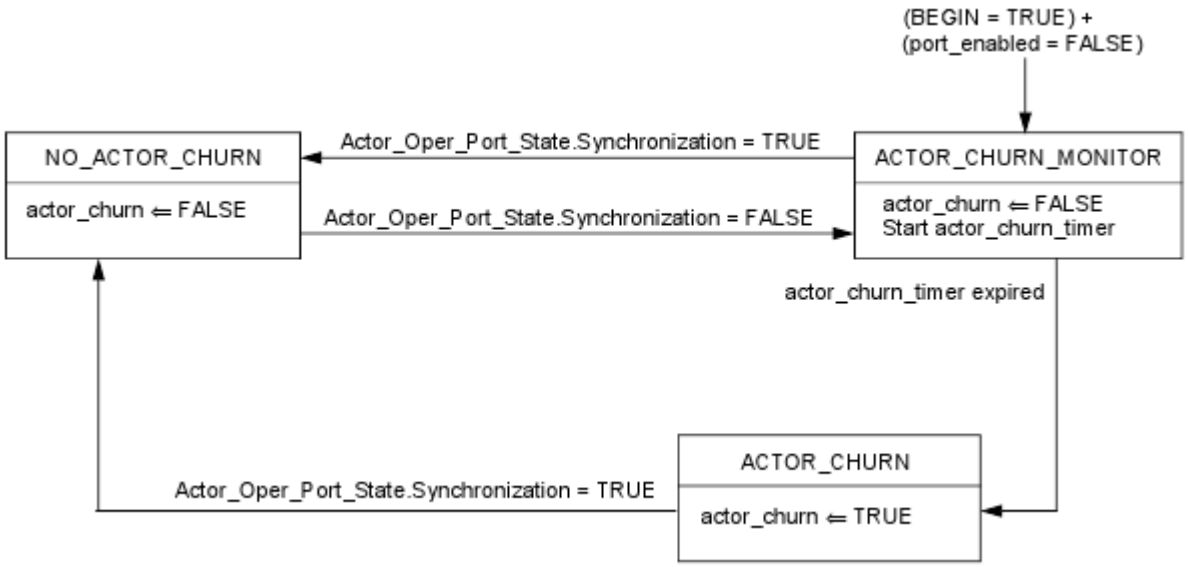


Figura 12: Detalhamento dos estados da *Actor Churn Detection Machine* [IEE08]**.**

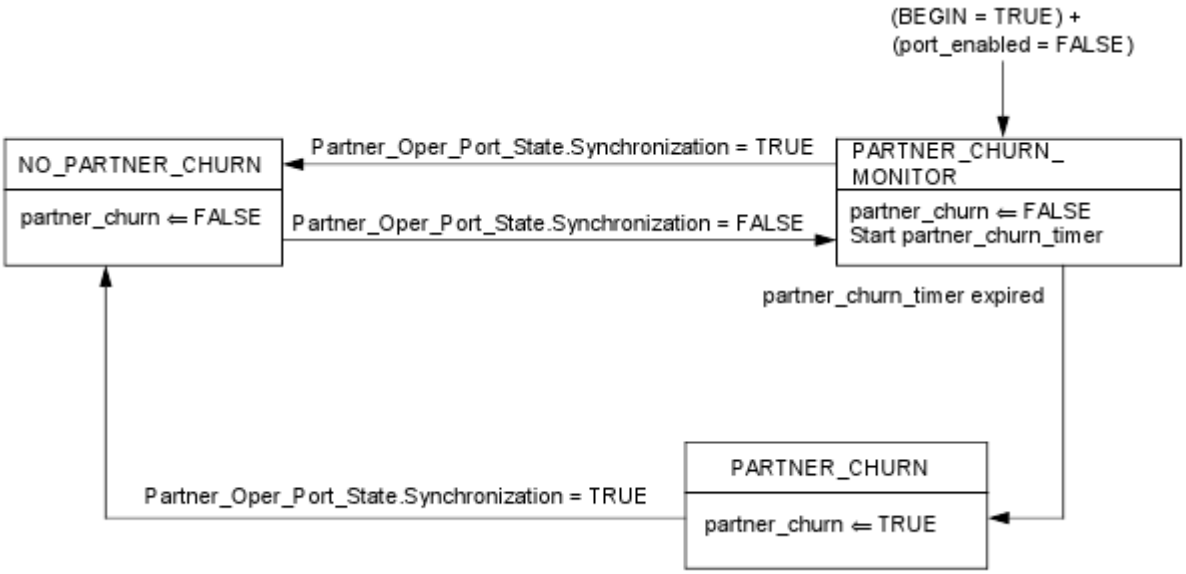


Figura 13: Detalhamento dos estados da *Partner Churn Detection Machine* [IEE08].

A *actor* CHURN inicia sua execução no estado ACTOR_CHURN_MONITOR, atribuindo o valor falso a variável que indica problemas no sincronismo do ator (actor_churn) e iniciando o *timer* para verificação da validade do sincronismo. Caso esse *timer* expire, a CHURN então passa para o estado ACTOR_CHURN, onde atribui o valor verdadeiro a variável actor_churn. Caso o bit de sincronismo do ator trone-se verdadeiro, tanto no estado ACTOR_CHURN_MONITOR quanto no estado ACTOR_CHURN, a CHURN passa para o estado NO_ACTOR_CHURN.

Na estado NO_ACTOR_CHURN, a variável actor_churn recebe o valor falso, indicando que o sincronismo está estabelecido sem problemas. Se a variável de sincronismo do ator tornar-se falsa a CHURN deixará este estado e irá novamente ao estado ACTOR_CHURN_MONITOR, onde novamente iniciará o *timer* de validade do sincronismo.

A sequência de operações da *partner* CHURN é exatamente a mesma que a da *actor* CHURN, com a única diferença que as variáveis monitoradas e alteradas, bem como os estados da máquina de estado, dizem respeito às informações do parceiro e não as do ator.

Na Figura 14 e na Figura 15 podemos observar, respectivamente, os pseudocódigos da *Actor Churn Detection Machine* e da *Partner Churn Detection Machine*.

```
enquanto (verdadeiro) faça
        caso (estado_actor_churn) em:
                ACTOR_CHURN_MONITOR:
                        actor.churn = FALSO;
                        inicia actor_churn_timer(porta);        /* inicia timer de deteccao */
                        se (timer == 0) então
                                estado_actor_churn = ACTOR_CHURN;
                        senão se (actor.oper_port_state.Syncronization == TRUE) então
                                estado_actor_churn = NO_ACTOR_CHURN;
                        fim se;
                fim;

                ACTOR_CHURN:
                        actor.churn = VERDADEIRO;
                        se (actor.oper_port_state.Syncronization == TRUE) então
                                estado_actor_churn = NO_ACTOR_CHURN;
                        fim se;
                fim;

                NO_ACTOR_CHURN:
                        actor.churn = FALSO;
                        se (actor.oper_port_state.Syncronization == FALSO) então
                                estado_actor_churn = ACTOR_CHURN_MONITOR;
                        fim se;
                fim;
        fim caso;
fim enquanto;
```

Figura 14: Pseudocódigo da *Actor Churn Detection Machine.*

```
enquanto (verdadeiro) faça
        caso (estado_partner_churn) em:
                PARTNER_CHURN_MONITOR:
                        partner.churn = FALSO;
                        inicia partner_churn_timer(porta);      /* inicia timer de deteccao */
                        se (timer == 0) então
                                estado_partner_churn = PARTNER_CHURN;
                        senão se (partner.oper_port_state.Syncronization == TRUE) então
                                estado_partner_churn = NO_PARTNER_CHURN;
                        fim se;
                fim;

                PARTNER_CHURN:
                        partner.churn = VERDADEIRO;
                        se (partner.oper_port_state.Syncronization == TRUE) então
                                estado_partner_churn = NO_PARTNER_CHURN;
                        fim se;
                fim;

                NO_PARTNER_CHURN:
                        partner.churn = FALSO;
                        se (partner.oper_port_state.Syncronization == FALSO) então
                                estado_partner_churn = PARTNER_CHURN_MONITOR;
                        fim se;
                fim;
        fim caso;
fim enquanto;
```

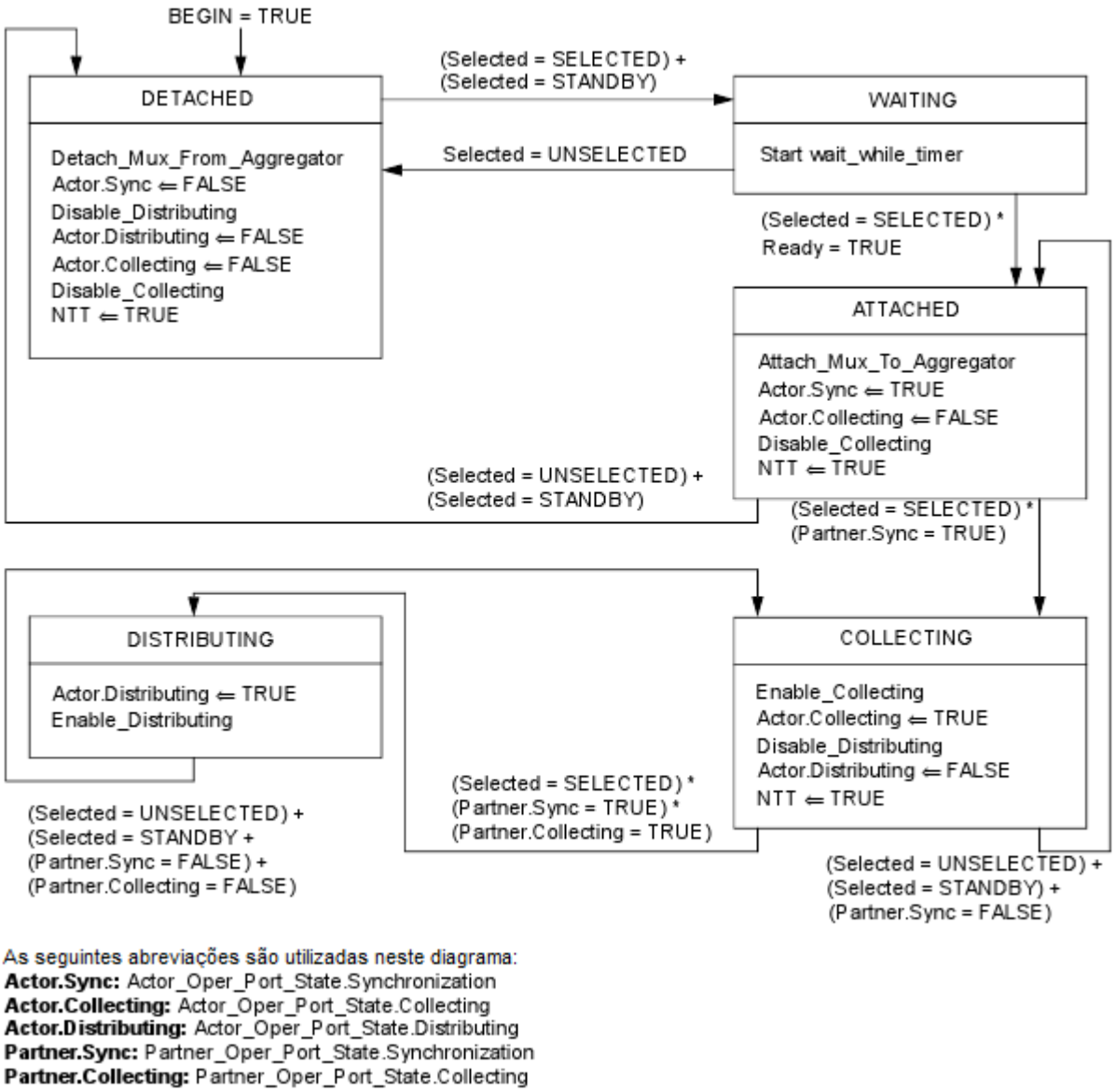
Figura 15: Pseudocódigo da *Partner Churn Detection Machine.*

Da mesma forma que as máquinas de estados, as implementações das *Churn Machines* de ator e parceiro são bastante semelhantes, apresentando como diferença apenas as variáveis monitoradas, uma vez que a *Actor Churn Detection Machine* monitora as variáveis de ator e a *Partner Churn Detection Machine* monitora as mesmas variáveis, porém pertencentes ao parceiro.

3.2.1.5 Mux Machine (MUX)

Para que pacotes possam ser transmitidos entre sistemas, é necessária a correta configuração dos algoritmos de coleta e distribuição de pacotes presentes no *switch*. A *Mux Machine* é responsável pela escolha destas políticas e pelo seu sincronismo. Ou seja, quando um sistema aciona o algoritmo de distribuição, este deve informar ao seu sistema parceiro que está acionando a distribuição de pacotes, utilizando o auxílio do *Marker Protocol*. Isto acontece para que o parceiro acione sua coleta de pacotes, tornando possível a troca de pacotes entre os sistemas.

Existem duas possíveis implementações para a MUX. A primeira trata do gerenciamento desacoplado das funções de coleta e distribuição de pacotes. Esta implementação é a recomendada pela NORMA [IEE08], por diminuir o risco de perda de pacotes. A segunda forma de se implementar a MUX dispões um modelo de gerenciamento de coleta e distribuição de pacotes acoplado, onde ambas as funções são habilitadas e desabilitadas simultaneamente. Neste trabalho, considera-se a implementação da MUX com gerenciamento desacoplado, proposta por Molina [MOL11]. A Figura 16 detalha os estados da MUX, com gerenciamento desacoplado das funções de coleta e distribuição.



As seguintes abreviações são utilizadas neste diagrama:
Actor.Sync: Actor_Oper_Port_State.Synchronization
Actor.Collecting: Actor_Oper_Port_State.Collecting
Actor.Distributing: Actor_Oper_Port_State.Distributing
Partner.Sync: Partner_Oper_Port_State.Synchronization
Partner.Collecting: Partner_Oper_Port_State.Collecting

Figura 16: Detalhamento dos estados da *MUX Machine* [IEE08].

O primeiro estado da MUX é o DETACHED. Neste estado, é executada a função Detach_Mux_From_Aggregator, que desvincula a porta em questão do agregador a ela vinculado atualmente. Ainda neste estado são desabilitadas as funções de coleta e distribuição de pacotes de dados, através das funções Disable_Distributing e Disable_Collecting, respectivamente. Um sinal de NTT também é gerado por esse estado, solicitando o envio das informações locais para o respectivo sistema parceiro. Caso a variável Selected receba os valores SELECTED (indica que um agregador foi selecionado para a porta) ou STANDBY (indica que a porta está aguardando a seleção de um agregador), a MUX parte para o estado WAITING.

No estado WAITING, é iniciado um *timer* que mantém a MUX neste estado até que as demais portas selecionem seus respectivos agregadores. Assim, quando todas as portas tiverem selecionado seus agregadores, a MUX, então, parte para o estado ATTACHED. Porém, caso a porta permaneça sem um agregador, a variável Selected receberá o valor UNSELECTED, fazendo com que a MUX retorne ao estado DETACHED.

Em ATTACHED, a MUX vincula a porta ao agregador selecionado, através da função Attach_Mux_From_Aggregator e desabilita a função de coleta de pacotes de dados, utilizando para tanto a função Disable_Collecting. Neste estado também é gerado um sinal de NTT, requisitando o envio das informações atuais da porta para o sistema parceiro. Caso não a variável Selected receba os valores UNSELECTED ou STANDBY, a MUX retornará ao estado DETACHED. Senão, tendo selecionado um agregador e sincronizado com o parceiro, a MUX partirá para o estado COLLECTING.

No estado COLLECTING, a função de coleta de pacotes de dados é ativada, através da função Enable_Collecting e a função de distribuição de pacotes de dados é desabilitada, através da função Disable_Distributing. Assim, um novo NTT é gerado para informar o sistema parceiro de que este sistema agora está coletando pacotes de dados. Caso um agregador não esteja mais selecionado para a porta ou o sincronismo com o parceiro tenha sido desfeito, a MUX retorna ao estado ATTACHED. Senão, caso o parceiro da porta em questão ative sua função de coleta de pacotes de dados, a MUX parte para o estado DISTRIBUTING.

Uma vez no estado DISTRIBUTING, a MUX ativa a função de distribuição de pacotes de dados através da função Enable_Distributing. A MUX somente deixará este estado se não houver mais um agregador selecionado para a porta, o parceiro da porta sair de sincronismo ou o parceiro desabilitar a função de coleta de pacotes de dados.

A Figura 17 trás o pseudocódigo da parte principal da MUX, detalhando as operações realizadas em cada um de seus estados, com exceção das variáveis de controle externas.

```
enquanto (VERDADEIRO) faça
        caso (estado_mux_em
                DETACHED:
                        Detach_Mux_From_Aggregator(porta);      /* desvincula agregador da porta */
                        actor.oper_port_state.Syncronization = FALSO;
                        Disable_Distributing(porta);            /* desabilita distribuição de pacotes de dados */
                        actor.oper_port_state.Distributing = FALSO;
                        actor.oper_port_state.Collecting = FALSO;
                        Disable_Collecting(porta);              /* desabilita coleta de pacotes de dados */
                        NTT = VERDADEIRO;
                        se ((Selected == SELECTED) OU (Selected == STANDBY)) então
                                estado_mux = WAITING;
                        fim se;
                fim

                WAITING:
                        inicia wait_while_timer();              /* inicia timer de espera pela selecao */
                        se ((Selected == SELECTED) E (Ready == VERDADEIRO)) então
                                estado_mux = ATTACHED;
                        senão se (Selected == UNSELECTED) então
                                estado_mux = DETACHED;
                        fim se;
                fim;

                ATTACHED:
                        Attach_Mux_To_Aggregator(porta);        /* vincula o agregador selecionado a porta */
                        actor.oper_port_state.Syncronization = VERDADEIRO;
                        actor.oper_port_state.Collecting = VERDADEIRO;
                        Disable_Collecting(porta);              /* desabilita distribuição de pacotes de dados */
                        NTT = VERDADEIRO;
                        se ((Selected == SELECTED) E (partner.oper_port_state.Syncronization == VERDADEIRO)) então
                                estado_mux = COLLECTING;
                        senão se ((Selected == UNSELECTED) OU (Selected == STANDBY)) então
                                estado_mux = DETACHED;
                        fim se;
                fim;

                COLLECTING:
                        Enable_Collecting(porta)                /* habilita coleta de pacotes de dados */
                        actor.oper_port_state.Collecting = VERDADEIRO;
                        Disable_Distributing(porta);            /* desabilita distribuição de pacotes de dados */
                        actor.oper_port_state.Distributing = FALSO;
                        NTT = VERDADEIRO;
                        se ((Selected == UNSELECTED) OU (Selected == STANDBY) OU (partner.oper_port_state.Syncronization ==
FALSO)) então
                                estado_mux = ATTACHED;
                        senão se ((Selected == SELECTED) E (partner.oper_port_state.Syncronization == VERDADEIRO) E
(partner.oper_port_state.Collecting == VERDADEIRO)) então
                                estado_mux = DISTRIBUTING;
                        fim se;
                fim;

                DISTRIBUTING:
                        actor.oper_port_state.Distributing = VERDADEIRO;
                        Enable_Distributing(porta);             /* habilita distribuição de pacotes de dados */
                        se ((Selected == UNSELECTED) OU (Selected == STANDBY) OU (partner.oper_port_state.Syncronization ==
FALSO) OU (partner.oper_port_state.Collecting == FALSO)) então
                                estado_mux = COLLECTING;
                        fim se;
                fim;
        fim caso;
fim enquanto;
```

Figura 17: Pseudocódigo da *Mux Machine.*

3.2.1.6 Selection Logic (SLOGIC)

Para que duas ou mais portas possam ser unidas na mesma agregação, é necessário que ambas as portas possuam uma mesma chave, que identificará quais portas pertencerão à mesma agregação. A *Selection Logic* é responsável pela atribuição destas chaves, bem como pela montagem dos identificadores dos agregadores.

A *Selection Logic* associa portas a agregadores do sistema, utilizando o *Link Aggregation Group Identifier* ou LAG ID das portas, de maneira que toda porta faça parte de um *Link Aggregation Group* ou LAG. Esta também configura os agregadores de maneira a garantir os seguintes requisitos [MOL11]:

- Todo sistema deve ter no mínimo um agregador: mesmo que o sistema não possua portas efetivamente agregadas, o mesmo deve possuir ao menos um agregador que o identifique;

- Cada agregador deve ter uma chave operacional: quando uma porta selecionar um agregador, esta deverá possuir sua chave operacional condizente com a chave operacional do agregador selecionado;

- Portas que podem ser agregadas devem ter a mesma chave operacional, assim como as portas que irão operar sozinhas devem possuir chaves operacionais distintas;

- Cada agregador deve ter uma chave operacional e um identificador que o diferencie dos outros agregadores do sistema;

- Uma porta pode ser vinculada somente a um agregador que tenha a mesma chave operacional que ela possui;

- Portas que fazem parte do mesmo LAG devem estar vinculadas ao mesmo agregador;

- Portas com chaves operacionais únicas devem ser gerenciadas por um único agregador.

3.2.2 Gerência

Além das máquinas de estado que descrevem o comportamento do LACP, também é necessário prover de alguma maneira as configurações ao algoritmo. Para isto, foi desenvolvida uma *thread* de gerência, a qual tem por objetivo ler comandos e valores da entrada padrão do sistema e atribuí-los as respectivas variáveis de controle interno do LACP.

Esta *thread* permanece constantemente ativa, monitorando a entrada padrão que, no caso do LACP funcionando com o simulador, é o terminal principal, onde foi inicializado o

LACP. Assim que uma configuração é digitada e a tecla *enter* é pressionada, o LACP realiza a leitura das informações e realiza as devidas operações e atribuições. São exemplos de operações: configurar a atividade do sistema (ativo ou passivo); atribuir a prioridade do sistema e das portas; inicializar as portas; configurar as agregações, bem como as portas que compões as mesmas; dentre outras.

3.3 Marker Protocol

O LACP utiliza duas funções distintas para o transporte de pacotes. Essas funções são conhecidas como função de Coleta de Pacotes (*Frame Collector*) ou FC e função de Distribuição de Pacotes (*Frame Distributor*) ou FD.

A FC é responsável por coletar os pacotes que chegam a cada agregador e por entregá-los ao respectivo cliente MAC. Já à FD cabe o trabalho de distribuir os pacotes recebidos do cliente MAC aos respectivos agregadores. Para o correto funcionamento da comunicação, quando um dos lados da comunicação estiver distribuindo, o outro deve estar coletando. Sendo assim, é necessário certo sincronismo entre estas duas funções. Em outras palavras, quando um dos lados da comunicação iniciar sua FD, este deve informar ao seu parceiro que o está iniciando, para que o parceiro inicie sua FC. Essa conversa estabelecida entre os dois lados para que ambos possam entrar em estados complementares respectivos é provida pelo *Marker Protocol* [IEE08].

3.3.1 Operação do Marker Protocol

O *Marker Protocol* tem a função de gerar e responder com pacotes de controle próprios, os quais são responsáveis pela comunicação entre dois sistemas, a fim de colocar em sincronia as funções de coleta e distribuição de pacotes do LACP.

Quando um sistema ator deseja ativar a FD, este solicita ao *Marker Protocol* que informe isto ao seu sistema parceiro. Então, o *Marker Protocol* gera um pacote do tipo *Marker* PDU (MPDU) e o envia ao seu sistema parceiro. O sistema parceiro, ao receber um MPDU, ativa a FC e gera um pacote do tipo *Marker Response* PDU (MRPDU) e o envia ao seu sistema parceiro (no caso, o sistema ator que solicitou a comunicação). Ao receber o MRPDU, o sistema ator ativa seu FD e dá seguimento à execução do protocolo.

A Figura 18 ilustra a sequência de operação do *Marker Protocol*, iniciando pelo sistema que deseja ativar a função de distribuição de pacotes de dados.

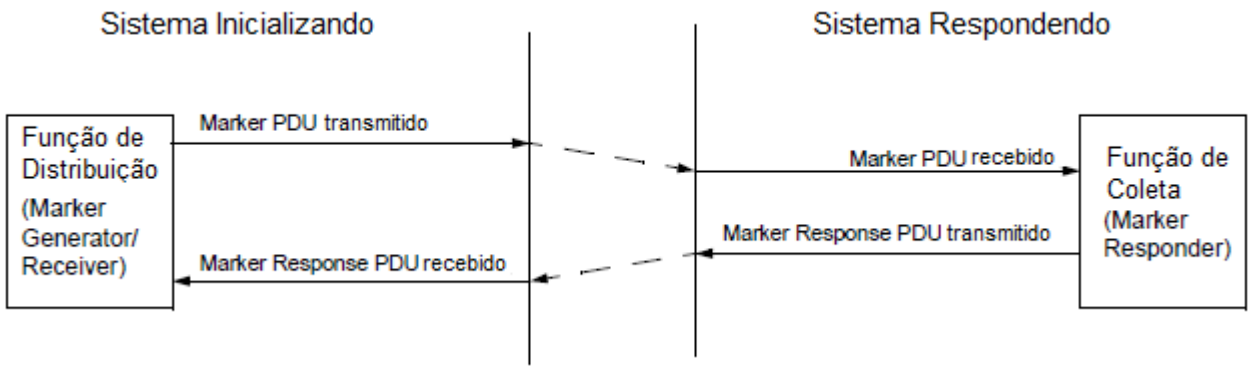


Figura 18: Sequência de operação do *Marker Protocol* [IEE08].

3.3.2 Marker PDU e Marker Response PDU

O *Marker Protocol* comunica-se utilizando dois tipos de pacotes. O primeiro é o *Marker* PDU, ou MPDU, que inicia a comunicação entre os dois lados de uma agregação. O segundo é o *Marker Response* PDU, ou MRPDU. Este, por sua vez, é enviado em resposta ao MPDU, finalizando uma comunicação entre os lados de uma agregação. A Figura 19 detalha os campos dos pacotes MPDU e MRPDU.

Figura 19: Campos dos pacotes MPDU e MRPDU [IEE08].

3.3.3 Marker Generator/Receiver

O *Marker Generator/Receiver* é responsável por duas tarefas distintas: primeiro, ele é responsável pela montagem e envio de um MPDU, quando for solicitado pelo protocolo. Esta solicitação é feita quando o protocolo deseja habilitar a FD. Em um segundo momento, o *Marker Generator/Receiver* recebe a resposta do sistema parceiro, ou seja, recebe o MRPDU enviado pelo seu parceiro e então ativa a FD.

A implementação do *Marker Generator/Receiver* é opcional, uma vez que não é obrigatório que um sistema ator informe ao seu parceiro que deseja ativar a FD. Porém, quando implementado, facilita o entendimento entre os dois lados da agregação, pois o sistema ator informa ao seu parceiro que irá iniciar a FD, preparando o sistema parceiro para iniciar a FC.

3.3.4 Marker Responder

Embora a implementação do *Marker Generator/Receiver* seja opcional, qualquer implementação do *Marker Protocol* tem a obrigação de responder a pacotes gerados pelo *Marker Generator/Receiver*, para que o funcionamento do protocolo que gerou o pacote não fique trancado aguardando o recebimento de uma confirmação.

O *Marker Responder* é responsável por receber um MPDU e responder ao remetente com um MRPDU, para que este possa iniciar a sua FD. Quando o *Marker Responder* recebe um MPDU, este sinaliza ao LACP que o seu sistema parceiro deseja iniciar a FD e que o LACP deve iniciar a FC.

3.4 Estrutura Geral de Simulação do LACP

A simulação do LACP se dá de uma maneira relativamente fácil de ser compreendida. Para que o LACP possa ser simulado corretamente, é necessário que o simulador seja executado em conjunto com o LACP. A comunicação é realizada entre duas máquinas pessoais (*Personal Computer* ou PC) distintas. Cada máquina possui suas próprias cópias do LACP e simulador, idênticas as do seu sistema parceiro. Para iniciar o simulador, basta abrir um terminal no sistema operacional que, obrigatoriamente, deve ser um sistema operacional Linux, e acessar o diretório onde está localizado o simulador. Neste terminal serão exibidas as mensagens de trocas de pacotes do simulador. Também podem ser abertos, em sequência, ***n*** terminais, onde ***n*** é o número de portas que o LACP irá gerenciar. Nestes terminais, são exibidas mensagens referentes aos estados de cada uma das portas.

Ao ser iniciado, o simulador mostra na tela principal uma mensagem de inicio bem sucedido e, a partir daí, passa a mostrar algumas informações do pacotes recebidos e enviados. A Figura 20 ilustra as mensagens exibidas pelo simulador, numa execução simulando 8 portas.

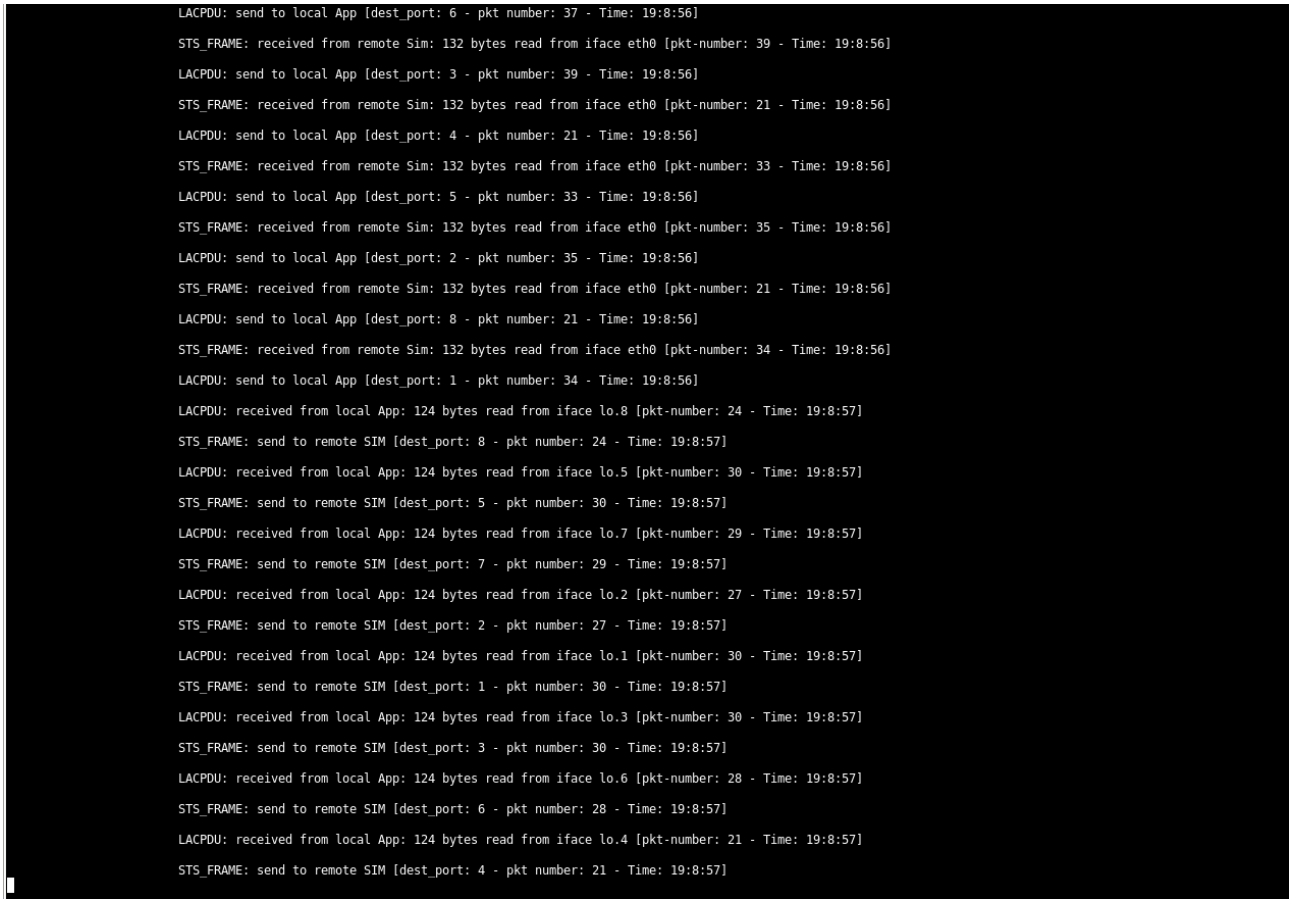


Figura 20: Mensagens exibidas pelo simulador.

Podemos observar na Figura 20 quatro tipos de mensagens exibidas pelo simulador, divididas entre LACPDUs e pacotes de comunicação entre simuladores (STS_FRAME):

- *LACPDU: received from local App: 124 bytes read from iface lo.**n** [pkt-number: **x** – Time: **hh:mm:ss**]* – indica o recebimento de um LACPDU pelo simulador, vindo da aplicação (LACP), com tamanho de 124 *bytes*, através da interface lo.***n***, onde ***n*** indica o índice da interface virtual que enviou o pacote, ***x*** indica o número do pacote e ***hh:mm:ss*** indica a hora, o minuto e o segundo no momento do recebimento;

- *LACPDU: send to local App [dest_port: **n** – pkt-number: **x** – Time: **hh:mm:ss**]* – indica o envio de um LACP do simulador a aplicação, cuja porta destino é dada por ***n***, ***x*** indica o número do pacote e ***hh:mm:ss*** indica a hora, o minuto e o segundo no momento do envio;

- *STS_FRAME: send to remote SIM [dest_port **n** – pkt-number: **x** – Time: **hh:mm:ss**]* – indica o envio de um pacote de controle do simulador local ao simulador remoto, cuja porta destino na aplicação é dada por ***n***, ***x*** indica o número do pacote enviado e ***hh:mm:ss*** indica a hora, o minuto e o segundo no momento do envio;

- *STS_FRAME: received from remote SIM: 132 bytes read from iface eth0 [pkt-number: **x** – Time: **hh:mm:ss**]* – indica o recebimento de um pacote do simulador remoto, cujo tamanho é de 132 *bytes*, lido da interface *eth0*, sendo o número do pacote dado por ***x*** e hora, minuto e segundo do recebimento indicado por ***hh:mm:ss***.

Uma vez iniciado o simulador, as rotinas de envio e recebimento de pacotes de controle já se tornam disponíveis ao LACP. Este, então, pode ser executado em outro terminal na mesma máquina e, após recebida sua configuração ou através da configuração padrão, irá iniciar a comunicação com seu sistema parceiro. Durante a simulação, algumas informações podem ser exibidas pelo LACP, caso este tenha sido compilado em modo *debug*. Caso contrário, o LACP irá apenas ficar aguardando comandos, para manipular valores de duas variáveis e modificar suas configurações.

O *Marker Protocol* inicia sua execução juntamente com o LACP, implicitamente, pois seu código está inserido no mesmo código do LACP, não necessitando abrir outro terminal no sistema operacional e executar um terceiro programa em paralelo com os demais.

4 IMPLEMENTAÇÃO

Este trabalho é concomitante ao trabalho de Molina [MOL11], o qual visa a implementação do LACP com o intuito de inseri-lo em uma plataforma de *hardware* específica. Ambos os trabalhos se complementam, pois antes de ser migrada para a plataforma de *hardware*, a implementação do LACP proposta por Molina [MOL11] será validada com a utilização do simulador proposto neste trabalho. Não é abordada na proposta de Molina a implementação do *Marker Protocol*, o qual faz parte do

funcionamento do LACP e é parte integrante desta proposta de trabalho de conclusão. Neste sentido o trabalho estende a funcionalidade do LACP alvo do trabalho de Molina.

Analisando as necessidades do projeto, a implementação do simulador torna-se prioritária em relação a implementação do *Marker Protocol*, pois tanto o LACP quanto o *Marker Protocol* necessitam de uma infraestrutura para a troca de pacotes, a qual será disponibilizada pelo simulador.

4.1 Visão Geral do Sistema

A arquitetura geral do sistema que compreende o LACP e o simulador foi elaborada baseando-se em um ambiente real de desenvolvimento, onde tem-se à disposição uma plataforma de *hardware* própria para a implementação de um protocolo, substituindo tal plataforma de *hardware* pelo simulador. A Figura 21 ilustra a arquitetura geral do sistema de simulação.

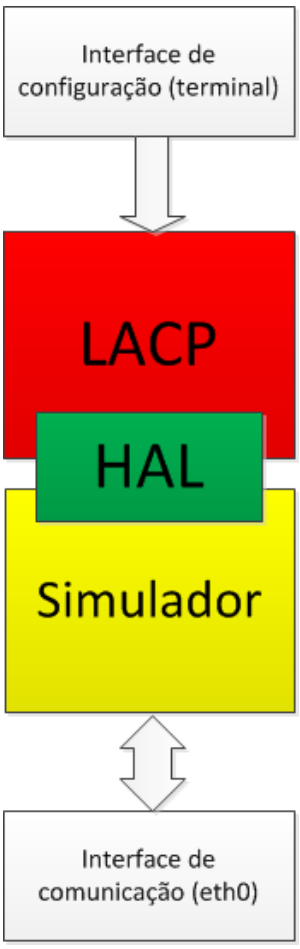


Figura 21: Arquitetura geral do sistema de simulação para o LACP.

LACP e simulador são duas aplicações distintas, descritas na linguagem de programação C. O LACP é o programa que executa as funções do protocolo LACP, recebendo nele as configurações para criação e gerenciamento das agregações. O simulador é o programa que simula as funções de um *switch*, disponibilizando ao LACP as funções desempenhadas por um *switch* real, sendo sua principal função conceber a comunicação entre dois sistemas. Para a comunicação entre o LACP e o simulador foi criada a *Hardware Abstraction Layer* (Camada de Abstração de *Hardware*), ou HAL. Ao implementar um protocolo em *hardware* que deva executar uma camada de *software*, costuma-se criar uma HAL. Todas as ações do protocolo que necessitam manipular estruturas do *hardware*, como registradores ou tabelas internas de um equipamento, são encapsuladas na HAL através de funções em uma linguagem de programação. Isto permite abstrair estruturas de *hardware* específicas que variam em cada equipamento que atende ao protocolo. Assim, o *software* de qualquer equipamento precisa enxergar apenas os protótipos das funções da HAL. Com isto, as implementações do protocolo tornam-se em grande medida portáveis, necessitando poucos ajustes de código, quando este for migrado para diferentes plataformas de *hardware*.

O LACP possui em si a implementação completa do algoritmo descrito em [IEE08]. Para isso, foram criadas nele estruturas responsáveis por armazenar as informações necessárias para o correto funcionamento do LACP. Dentre estas estruturas, pode-se destacar:

- *struct Aggregator:* estrutura responsável por armazenar as informações referentes aos agregadores, tais como identificador e chave do agregador;

- *struct Port:* estrutura responsável por armazenar as informações referentes às portas, tais como número e chave da porta;

- *struct System:* estrutura responsável por armazenar as informações referentes ao sistema, tais como endereço MAC do sistema e identificador do sistema;

- *struct StateMachines:* estrutura responsável por armazenar as variáveis de sistema que alteram os estados das máquinas de estados do LACP;

Como cada sistema possui um número variável de portas, foram desenvolvidas também estruturas dinâmicas para armazenar conjuntos de informações como as acima descritas. Por exemplo, em um cenário onde existem oito portas ativas sendo simuladas, um vetor dinâmico armazena oito estruturas do tipo *Port*, uma para cada porta, cada uma contendo as informações pertinentes a uma determinada porta.

Todas as operações em que o LACP solicita funcionalidades do *hardware* são efetuadas pela HAL. A HAL, por sua vez, para atender às requisições do LACP, aciona o simulador, utilizando as funções nele desenvolvidas para atender às necessidades do LACP. Basicamente, a HAL deve receber um pacote do simulador e repassá-lo ao LACP e também enviar um pacote formado pelo LACP ao simulador. O simulador, por sua vez, deve receber os pacotes de outro simulador, e repassá-los à HAL, bem como transmitir os pacotes repassados pela HAL para outro simulador remoto, conectado diretamente ao simulador local.

Para a compilação dos códigos-fonte, foram utilizados *scripts* da ferramenta *make*. Assim, todo o processo de compilação dos diversos arquivos contendo os códigos-fonte tornou-se mais rápido e simplificado. O arquivo *Makefile* contém os comandos reconhecidos pela ferramenta *make* para a compilação do projeto. Nele, reconhecem-se duas opções: APP, para a compilação dos códigos-fonte exclusivos do LACP; e SIM, para compilação dos códigos-fonte exclusivos do simulador. Utilizando a opção *all*, pode-se compilar ambos, LACP e simulador. Outra opção inserida no *Makefile* é a de *debug*. Esta opção, quando selecionada, faz com que o código seja compilado exibindo todas as mensagens de depuração.

4.2 Detalhamento do Simulador

Quando se implementa protocolos de rede em equipamentos específicos, geralmente é necessária a utilização do próprio equipamento alvo ou de um kit de desenvolvimento. O objetivo é que no decorrer da codificação do protocolo o mesmo possa ser testado e validado, tendo sua funcionalidade comprovada em *hardware*. Como todo protocolo de rede, o LACP será finalmente implantado em um equipamento de rede específico. Esse equipamento deve disponibilizar ao algoritmo LACP as funções que ele necessita para operar corretamente, tais como entrega e envio de pacotes, alertas de recebimento e

envio de pacotes, configuração de políticas de coleta e distribuição de pacotes, entre outras.

A ideia deste trabalho é tornar possível a implementação do protocolo LACP sem a utilização de um equipamento ou kit de desenvolvimento. Para tanto, se faz necessário o desenvolvimento de um simulador em *software* capaz de disponibilizar ao protocolo funções similares às funções presentes nos equipamentos e/ou kits de desenvolvimento.

4.2.1 Funcionamento

O simulador consiste em implementações das funções da HAL onde, ao invés de execução em *hardware*, são realizadas operações equivalentes em funcionalidade em *software*, tornando possível o desenvolvimento do protocolo mesmo sem contar com uma plataforma de *hardware* para ensaios.

A Figura 22 ilustra a arquitetura de comunicação entre simuladores, HAL e o LACP operando entre duas máquinas distintas. O simulador atua em conjunto com o LACP, sendo este executado por último, para evitar que o protocolo faça requisições ao simulador antes deste ter sido posto em execução. A ideia do simulador é criar uma interface virtual de comunicação entre o LACP e o simulador e uma interface de comunicação real entre instâncias de simuladores dos dois lados da comunicação. Cada interface virtual representa um enlace pertencente a um equipamento sob simulação. Assim, para um equipamento com ***n*** portas físicas reais, o simulador terá ***n*** portas virtuais realizando a troca de pacotes entre o LACP e o simulador. A troca de pacotes entre sistemas, porém será realizada apenas por interfaces reais.

O simulador também é responsável pela troca de pacotes de controle entre as máquinas com o LACP em execução. Para tanto, o simulador utiliza a interface real ativa de cada máquina como caminho para a troca de pacotes de controle. Os pacotes de dados que trafegam na rede não são tratados pelo simulador, pois estes não interferem diretamente no funcionamento do algoritmo do LACP.

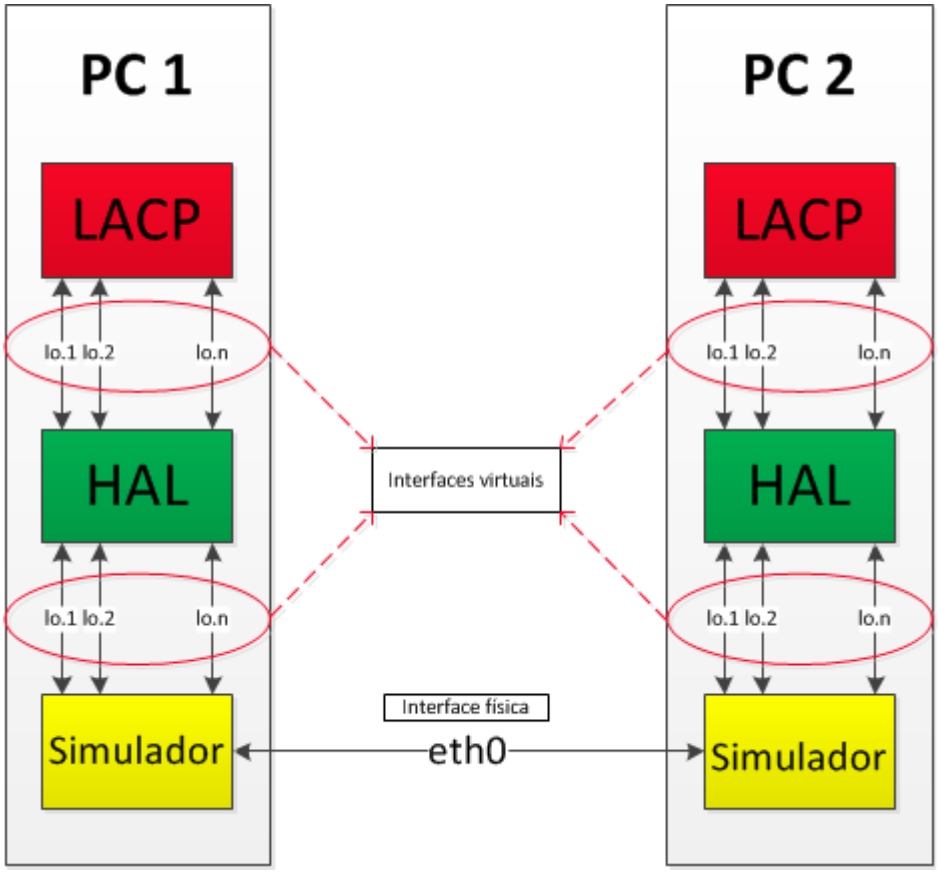


Figura 22: Arquitetura de comunicação entre simulador, HAL e LACP.

Para o fluxo de pacotes entre simuladores foi criado um pacote exclusivamente para que o simulador consiga operar como um *switch*, ou seja, identificar a porta de origem/destino e efetuar a transmissão do pacote. Em equipamentos como *switches* não é necessário esse tipo controle, pois cada porta está conectada diretamente ao seu par. Neste caso, o controle de origem/destino do pacote é de responsabilidade do *hardware*. A Figura 23 ilustra os campos do pacote utilizado na comunicação entre simuladores.

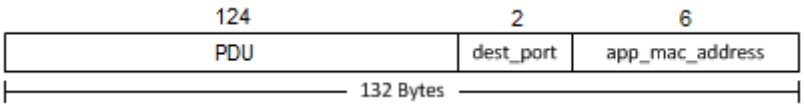


Figura 23: Pacote de comunicação entre simuladores.

O primeiro campo é um PDU. Este pode ser tanto um LACPDU quando um MPDU ou MRPDU, dependendo da necessidade, todos possuindo 124 *bytes*. Na prática, são geradas duas estruturas, uma para LACPDUs e outra para MPDUs ou MRPDUs. Em seguida, vem o campo *dest_port*, de dois *bytes*, responsável por identificar a porta destino em que a aplicação irá receber o pacote. O último campo da estrutura é o *app_mac_address*, de 6 *bytes*. Este armazena o endereço MAC virtual criado para cada

porta na aplicação. Este campo é necessário porque, no momento do envio do pacote, no campo *Destination* do PDU, o qual faz parte do cabeçalho *Ethernet*, é colocado o valor do endereço MAC da máquina onde o LACP está sendo simulado, e não o MAC virtual da porta, que seria o correto a ser feito numa implementação em *hardware* do LACP. Então, quando o pacote chega ao simulador destino, o PDU é desencapsulado do pacote recebido e o campo *Destination* recebe o valor armazenado em *app_mac_address*. Outro campo do PDU que é alterado no momento do envio de um simulador ao outro é o campo *type*, também do cabeçalho *Ethernet*. Tal campo é alterado para que possa ser criado o filtro que evita duplicações de pacotes na comunicação entre simuladores. Antes de encaminhar o PDU ao LACP, o campo *type,* que é recebido pelo simulador com o valor "666" hexadecimal, recebe o valor "8809", identificando o PDU como pertencente ao *Slow Protocol*. O PDU, agora corretamente montado, pode então ser encaminhado ao LACP pelo simulador, utilizando o campo *dest_port* do pacote recebido para identificar a qual porta do LACP o PDU deve ser enviado.

Se tratando de código em linguagem C, as estruturas que armazenam tais informações são detalhadas na Figura 24, mostrando a diferença entre a estrutura para LACPDUs e para MPDUs ou MRPDUs.

Pacote para M/MRPDU

```
struct sts_frame_m
{
    mpdu_t  sts_pdu;
    u_int16_t dest_port;
    u_int8_t app_mac_src[6];
}__attribute__((__packed__));
```

Pacote para LACPDU

```
struct sts_frame
{
    lacpdu_t  sts_pdu;
    u_int16_t dest_port;
    u_int8_t app_mac_src[6];
}__attribute__((__packed__));
```

Figura 24: Estruturas dos pacotes de comunicação entre simuladores.

Para a identificação de pacotes do tipo *Slow Protocol*, presentes no fluxo entre LACP e simulador, e para identificar o pacote proprietário, se fez necessária a utilização de um filtro genérico, capaz de identificar pacotes gerados por ***n*** interfaces (virtuais e físicas), de modo a se trabalhar em conjunto com as interfaces virtuais do LACP e com a interface física do simulador. Utilizando a ferramenta TCPDump [TCP11] é possível gerar o código em linguagem C capaz de executar a filtragem de pacotes. Isto requer que tal código seja associado a uma estrutura que possibilita a comunicação entre dois equipamentos através da rede, o *socket*, mais especificamente, um *socket filter*. Quando associado ao

socket, o filtro passa a capturar apenas os pacotes que estiverem dentro dos parâmetros configurados na criação do filtro pelo TCPDump. Assim, pode-se contornar problemas como o de *loopback*, quando um pacote enviado em uma interface virtual é reconhecido como um novo pacote recebido pela mesma interface que o enviou. São utilizados dois filtros: um no simulador, para comunicação entre simuladores, visando capturar apenas pacotes enviados pelo outro simulador e pelo LACP local, excluindo os demais pacotes; e outro filtro na HAL, responsável por filtrar e dispor apenas pacotes cujo endereço MAC destino seja o *Slow Protocol*. A Figura 25 e a Figura 27 ilustram o comando e a saída resultante do TCPDump para a criação dos filtros para a HAL e para o simulador, respectivamente.

```
parks02:/home/oleiro# tcpdump -dd -s 124 ether dst 01:80:C2:00:00:02
{ 0x20, 0, 0, 0x00000002 },
{ 0x15, 0, 3, 0xc2000002 },
{ 0x28, 0, 0, 0x00000000 },
{ 0x15, 0, 1, 0x00000180 },
{ 0x6, 0, 0, 0x0000007c },
{ 0x6, 0, 0, 0x00000000 },
parks02:/home/oleiro#
```

Figura 25: Comando e saída do TCPDump para criação do filtro da HAL.

Na Figura 25, pode-se observar as opções e os parâmetros passados ao TCDump para a geração do filtro. A opção “-dd” serve para indicar que a linguagem de saída do filtro será a linguagem de programação C. O valor 124 para a opção “-s” indica qual o tamanho dos pacotes aceitos pelo filtro. O parâmetro “ether” indica que será manipulado o cabeçalho do protocolo *Ethernet*, seguido do parâmetro “dst”, indicando qual o valor do campo *destiny* do cabeçalho *Ethernet* irá ser aceito pelo filtro. No caso presente, o valor é o endereço MAC do *Slow Protocol*, o qual abrange os PDUs. Dado o comando, nas linhas abaixo, entre parênteses, estão os valores que devem ser inseridos no *socket* para a criação do filtro. A Figura 26 traz o trecho de código referente à atribuição do filtro gerado pelo TCPDump para a HAL ao *socket* responsável pela filtragem.

```
/*initialize filter using tcpdump*/
struct sock_filter BPF_code[]= {
{ 0x20, 0, 0, 0x00000002 },
{ 0x15, 0, 3, 0xc2000002 },
{ 0x28, 0, 0, 0x00000000 },
{ 0x15, 0, 1, 0x00000180 },
{ 0x6, 0, 0, 0x0000007c },
{ 0x6, 0, 0, 0x00000000 },
};
```

Figura 26: Atribuição do filtro do TCPDump ao *socket filter* da HAL.

```
parks02:/home/oleiro# tcpdump -dd -s 132 '(ether dst 01:80:C2:00:00:02 and not ether src 22:22:22:22:22:22) or
((ether dst 00:22:64:e4:4b:a0 and ether src 00:22:64:e4:59:c5) and ether proto 0x0666)'
{ 0x20, 0, 0, 0x00000002 },
{ 0x15, 0, 6, 0xc2000002 },
{ 0x28, 0, 0, 0x00000000 },
{ 0x15, 0, 14, 0x00000180 },
{ 0x20, 0, 0, 0x00000008 },
{ 0x15, 0, 11, 0x22222222 },
{ 0x28, 0, 0, 0x00000006 },
{ 0x15, 10, 9, 0x00002222 },
{ 0x15, 0, 9, 0x64e44ba0 },
{ 0x28, 0, 0, 0x00000000 },
{ 0x15, 0, 7, 0x00000022 },
{ 0x20, 0, 0, 0x00000008 },
{ 0x15, 0, 5, 0x64e459c5 },
{ 0x28, 0, 0, 0x00000006 },
{ 0x15, 0, 3, 0x00000022 },
{ 0x28, 0, 0, 0x0000000c },
{ 0x15, 0, 1, 0x00000666 },
{ 0x6, 0, 0, 0x00000084 },
{ 0x6, 0, 0, 0x00000000 },
parks02:/home/oleiro#
```

Figura 27: Comando e saída do TCPDump para criação do filtro do simulador.

O filtro para o simulador é um pouco mais complexo que o filtro da HAL. A primeira diferença está no tamanho do pacote. Como o pacote especial do simulador possui 132 *bytes*, este é o valor passado ao TCPDump para a criação do filtro. Na composição dos endereços MAC a serem filtrados no cabeçalho *Ethernet*, a lógica utilizada é: destino igual ao *Slow Protocol* e origem diferente do MAC local da porta (no caso visto na Figura 27, o MAC origem não pode ser 22:22:22:22:22:22), ou destino igual ao endereço da máquina local, tendo como e origem a máquina que enviou. Tudo isso, dentro do protocolo é identificado pelo valor “0666” hexadecimal, que é um valor arbitrário escolhido para a criação do pacote de comunicação entre simuladores. A Figura 27 traz o trecho de código referente à atribuição do filtro gerado pelo TCPDump para o simulador ao *socket* responsável pela filtragem.

```
/*initialize filter using tcpdump*/
struct sock_filter BPF_code[]= {
{ 0x20, 0, 0, 0x00000002 },
{ 0x15, 0, 6, 0xc2000002 },
{ 0x28, 0, 0, 0x00000000 },
{ 0x15, 0, 14, 0x00000180 },
{ 0x20, 0, 0, 0x00000008 },
{ 0x15, 0, 11, 0x22222222 },
{ 0x28, 0, 0, 0x00000006 },
{ 0x15, 10, 9, 0x00002222 },
{ 0x15, 0, 9, 0x64e449fc },
{ 0x28, 0, 0, 0x00000000 },
{ 0x15, 0, 7, 0x00000022 },
{ 0x20, 0, 0, 0x00000008 },
{ 0x15, 0, 5, 0x64e449d8 },
{ 0x28, 0, 0, 0x00000006 },
{ 0x15, 0, 3, 0x00000022 },
{ 0x28, 0, 0, 0x0000000c },
{ 0x15, 0, 1, 0x00000666 },
{ 0x6, 0, 0, 0x00000084 },
{ 0x6, 0, 0, 0x00000000 },
};
```

Figura 28: Atribuição do filtro do TCPDump ao *socket filter* do simulador.

Inicialmente, a implementação do simulador foi pensada utilizando um descritor de *socket* para cada porta emulada. Porém, depois de alguns experimentos, decidiu-se utilizar apenas um descritor, que seria compartilhado por todas as portas. Esta abordagem diminui a quantidade de memória utilizada, bem como facilita o gerenciamento, uma vez que utilizando vários descritores de *sockets*, seria necessário um controle sobre a utilização dos mesmos, os quais deveriam estar armazenados em uma estrutura genérica.

No momento do envio, o *socket* é reservado para a porta que está requisitando o envio do pacote de controle. Uma vez reservado, o *socket* é configurado com as informações necessárias para o envio do pacote de controle e então o pacote é enviado pelo *socket* para o sistema parceiro.

Para o recebimento, também é utilizado apenas um *socket*, o qual identifica a porta da aplicação que deve receber o pacote de controle, através do campo *dest_port*. Este campo está presente no pacote criado exclusivamente para a comunicação entre os simuladores. Recebido o pacote, o simulador desencapsula o pacote de controle do pacote recebido, altera os valores dos campos *Destiny*, atribuindo o valor do endereço MAC virtual da porta que irá receber o pacote de controle; e *type*, atribuindo o valor que

identifica o pacote como pertencente ao *Slow Protocol*. Feito isto, o pacote está pronto para ser enviado pelo simulador à porta destino do LACP. Este envio é realizado de forma bem semelhante ao pacote que é trocado entre simuladores. A HAL fica constantemente lendo de um descritor de *socket* se há um novo pacote válido para ser computado. Para este descritor de *socket*, o simulador encaminha o pacote de controle, utilizando a porta virtual como destino do pacote. Recebido o pacote, o LACP, então, inicia o processamento do mesmo, realizando as operações necessárias.

Toda a lógica implementada no simulador se resume a cinco funções desenvolvidas para tratar tanto LACPDUs quanto MPDUs e MRPDUs. Cada função possui sua sequência própria de operações, descritas a seguir:

- *sim_init_port_info()*: esta função serve para inicializar a estrutura "sim_ports_info", cujo tamanho é dado por NPORTS, que indica o número de portas do sistema. Esta estrutura armazena o nome de cada uma das interfaces virtuais lo.***n***;

- *init_sock()*: cria NPORTS descritores de arquivo (File Descriptor) ou FDs, um para cada interface virtual *lo*; e um descritor de arquivo extra para a recepção de pacotes. Estes FDs são criados com a família PF_PACKET e são usados para a comunicação entre simulador->LACPD e simulador->simulador. Para a recepção, uma combinação de *socket* e o filtro gerado pelo TCPDump são usados para filtrar os pacotes recebidos do LACP/simulador;

- *sim_send_data_to_sim(struct sts_frame *frame, int index)*: Substitui no cabeçalho *Ethernet*: (*i*) o campo *Destiny* pelo endereço MAC do simulador remoto; (*ii*) o campo *Source* pelo endereço MAC do simulador local e (*iii*) o campo *type* pelo valor "0666" hexadecimal. Então, utiliza a função sendto() para enviar o pacote recebido do LACP para o simulador remoto;

- *sim_send_data_to_app(lacpdu_t *pkt, u_int16_t dest_port, u_int8_t * app_mac_src)*: Substitui no cabeçalho *Ethernet*: (*i*) o campo *Destiny* pelo endereço MAC do *Slow Protocols*; (*ii*) o campo *Source* pelo endereço MAC da porta que enviou o pacote e (*iii*) o campo *type* pelo valor do tipo *Slow Protocols,*

"8809" hexadecimal. Então, utiliza a função sendto() para enviar o pacote recebido do simulador remoto para o LACP local;

- *sim_recv_data()*: Função padrão de recebimento de pacotes de controle do simulador. Responsável por receber pacotes do LACP local e do simulador remoto, identificá-los e encaminhá-los para os respectivos destinos. Caso seja um pacote vindo do LACP local, decodifica e ativa a função *sim_send_data_to_sim*, para enviar o pacote ao simulador remoto; Caso seja um pacote vindo do simulador remoto, decodifica e ativa a função *sim_send_data_to_app*, para enviar o pacote ao LACP local;

4.3 Detalhamento do Marker Protocol

Para a implementação do *Marker Protocol*, foi desenvolvida uma biblioteca para dispor as funções por ela exercidas. As funções essenciais para o correto funcionamento do *Marker Protocol* são as funções de recebimento, reconhecimento e envio de MPDUs e MRPDUs.

4.3.1 Funcionamento

As principais operações realizadas pelo *Marker Protocol* estão dentro de uma *thread*, exceto as operações do *Marker Protocol* que são acionadas pelo LACP. Esta *thread* atua apenas em resposta ao recebimento de um MPDU ou MRPDU. O comportamento desta *thread* descreve basicamente o funcionamento do *Marker Responder*, cuja implementação é obrigatória para o funcionamento correto do LACP.

Ao receber um PDU, o *Marker Protocol* identifica qual o tipo: caso seja um MPDU, a *thread* chama a rotina que inicia a função de coleta de pacotes de dados da porta e, em seguida, monta um MRPDU para ser enviado ao seu sistema parceiro, para informá-lo que a função de coleta de pacotes de dados está agora ativa; caso seja um MRPDU, a *thread* simplesmente ativa a função de distribuição de pacotes de dados da porta, continuando sua execução normal, voltando a monitorar os PDUs recebidos. A Figura 29 ilustra o pseudocódigo da *thread* responsável pelas operações descritas.

```
enquanto (VERDADEIRO) faça
        recebe_pdu_hal(pdu_recebido, porta);
        se (pdu == NULO) então
                continua;
        senão
                tipo_pdu = identifica_tipo_pdu(pdu_recebido);
                se (tipo == MRPDU) então
                        ativa_funcao_distribuição(porta);
                senão se (tipo == MPDU) então
                        ativa_funcao_coleta(porta);
                        envia_MRPDU(porta);
                fim se;
        fim se;
fim enquanto;
```

Figura 29: Pseudocódigo da *thread* do *Marker Protocol*.

Na *thread* criada são chamadas três funções pertencentes à HAL. A primeira é a função que recebe pacotes do simulador. A segunda é a função que ativa a distribuição de pacotes de dados. A terceira é a função que ativa a coleta de pacotes de dados. Estas duas últimas funções não são implementadas pelo simulador, pelo fato de que o simulador não pode trabalhar efetivamente com o tráfego de dados que não seja de controle. Esta limitação se dá por que as máquinas onde o simulador atua são computadores pessoais, não havendo como uma máquina deste tipo realizar agregações com mais de um enlace, visto que cada máquina possui apenas uma interface de rede ativa. Porém, quando o LACP for transferido para uma plataforma de *hardware*, o mesmo deve ter as funções de distribuição e coleta de pacotes de dados devidamente implementadas. Estas configuram o *hardware*, para que este esteja apto a realizar o tráfego de dados pelos enlaces configurados na agregação.

O *Marker Protocol* também é ativado quando o LACP deseja iniciar sua função de coleta de pacotes de dados. Neste momento, ele ativa o *Marker Generator*, implementado neste trabalho como uma função responsável por montar e enviar um MPDU. Este MPDU deverá ser recebido pelo sistema parceiro do LACP local, o qual ativará sua função de coleta de pacotes de dados e responderá ao remetente com um MRPDU, para que o mesmo possa efetivar a inicialização da sua função de distribuição de pacotes de dados. Uma vez terminada esta comunicação, os dois sistemas podem iniciar a troca de pacotes de dados.

Quem solicita a ativação da função de distribuição de pacotes de dados e, consequentemente, o envio de um MPDU é a *MUX Machine*. Tal máquina, conforme visto

na Seção 3.2.1.5, é responsável pelo controle de ativação das funções de coleta e distribuição de pacotes de dados. Logo, ela trabalha em conjunto com o *Marker Protocol*.

Assim como para armazenar os LACPDUs, foi criada uma estrutura para armazenar as informações dos MPDUs e MRPDUs. Valendo-se do fato de que um MPDU é muito semelhante a um MRPDU, uma única estrutura foi criada para armazenar ambos os tipos de PDUs, variando apenas o campo que define o tipo do pacote. A Figura 30: Estrutura para armazenar informações de MPDUs e MRPDUs.Figura 30 ilustra a estrutura criada para armazenar MPDUs e MRPDUs.

```
/* MPDU and MRPDU */
typedef struct mpdu_tag {
    struct ether_header eth_h;
    u_int8_t subtype; /* Marker = 0x02*/
    u_int8_t version;
    u_int8_t TLV_type_a; /* Marker/Marker Response Information */
    u_int8_t marker_info_len; /*  = 16 */
    u_int16_t requester_port;
    u_int8_t requester_system[ETH_ALEN];
    u_int32_t requester_transaction_id;
    u_int16_t pad;  /* = 0 */
    u_int8_t TLV_type_t; /* Terminator */
    u_int8_t terminator_len; /* = 0 */
    u_int8_t reserved[90];
} __attribute__ ((__packed__)) mpdu_t;
```

Figura 30: Estrutura para armazenar informações de MPDUs e MRPDUs.

5 EXPERIMENTOS

Na fase experimental de desenvolvimento do projeto, foram realizados vários testes para validação das funcionalidades do LACP e *Marker Protocol* com o simulador, cada um operando individualmente ou ambos atuando em conjunto. Nesta etapa, algumas ferramentas auxiliaram no processo de depuração.

Assim que as rotinas de recebimento de pacotes do LACP foram desenvolvidas, bem como as respectivas rotinas de recebimento do simulador, foram realizados alguns testes referentes ao comportamento do LACP e simulador quando do evento de recebimento de um pacote de controle, no caso, um LACPDU. Para tanto, como as rotinas de envio de pacotes ainda estavam sendo desenvolvidas, foi utilizada a ferramenta Packeth [PAC11],

capaz de gerar um pacote com os dados desejados e injetá-lo na rede. A Figura 31 mostra a tela de configuração dos pacotes no Packeth.

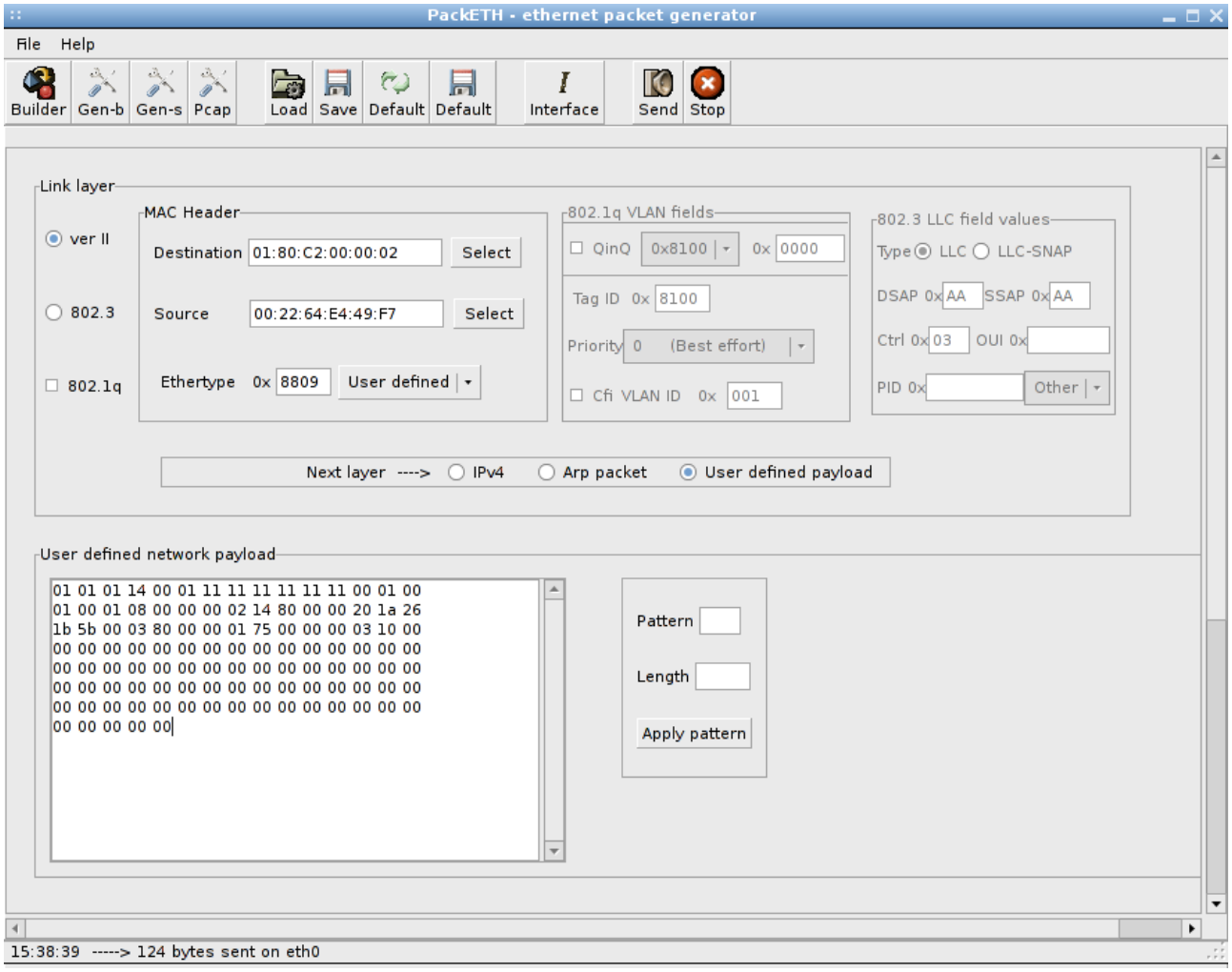


Figura 31: LACPDU montado no Packeth.

Inicialmente, inserem-se os campos de endereço MAC origem e destino. Depois, atribui-se o valor “8809”, que identifica o pacote como pertencente ao *Slow Protocol*. Para o preenchimento do *payload* (carga útil ou dados do pacote), tendo em vista o fato de que o Packeth não possui uma pré-configuração para a montagem de LACPDUs, foi necessário utilizar o campo *user defined network payload* (dados de rede definidos pelo usuário), permitindo assim que fossem inseridas as informações desejadas no pacote. Nesse campo, os dados do pacote devem ser inseridos em linguagem hexadecimal. Depois de inseridos os dados, basta selecionar a interface na qual o pacote será transmitido, no botão *Interface* e depois, enviar o pacote, clicando no botão *Send*.

Para comprovar o envio deste pacote na rede e a validade dos dados por ele enviados, outra ferramenta foi utilizada. A ferramenta em questão é o Wireshark [WIR11], um *software* capaz de monitorar interfaces de rede ativas em uma máquina, coletando e exibindo graficamente pacotes que trafegam pela interface de rede desta. Assim, quando um pacote é enviado pelo Packeth, pode-se comprovar seu envio através da captura do pacote pelo Wireshark. A Figura 32 mostra uma tela de captura do Wireshark, onde se monitora a comunicação entre o LACP trabalhando com o simulador e um *switch* com uma implementação proprietária do LACP. Os testes realizados com o simulador e um equipamento real serão abordados mais adiante nesta Seção.

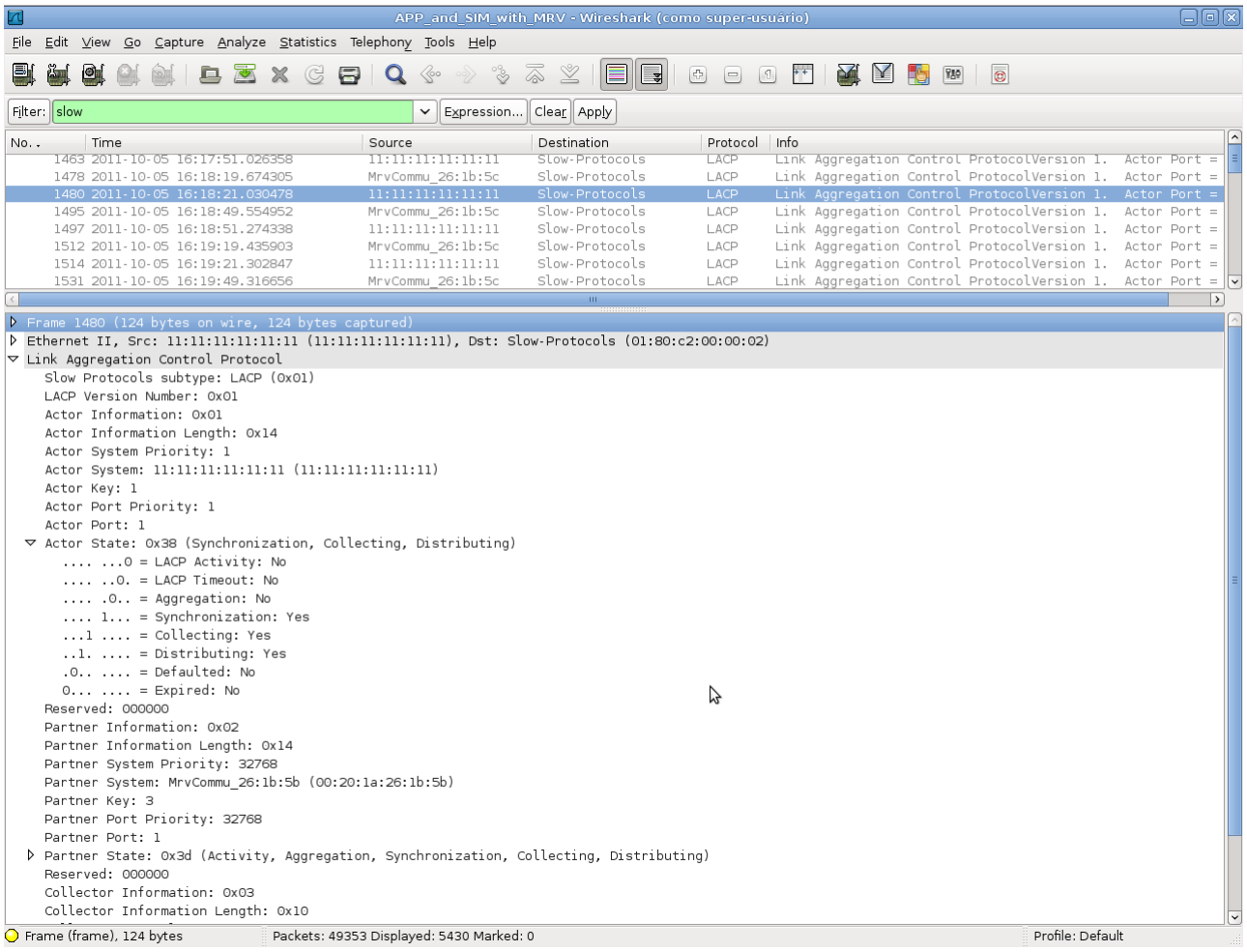


Figura 32: Tela de captura de um LACPDU pelo Wireshark.

Depois de verificado o envio do pacote, analisa-se o recebimento deste pelo simulador e, em caso positivo, o comportamento da máquina de recebimento de pacotes do LACP, a RX. Depois de validado o funcionamento da recepção de pacotes de controle pelo simulador e o correto comportamento do LACP, foi testada, a máquina de transmissão de

pacotes de controle, a TX. Do mesmo modo, utiliza-se o Wireshark para comprovar o envio do pacote pelo simulador.

Na Figura 32, pode-se observar também a existência do campo *Filter*. Neste campo, podem ser definidos filtros, os quais definem quais tipos de pacotes ou protocolos serão capturados, ignorando pacotes que não pertençam ao tipo definido no filtro. Para os testes realizados, insere-se a palavra *slow*, capturando apenas pacotes pertencentes ao protocolo *Slow Protocol*, protocolo este que engloba LACPDUs, MPDUs e MRPDUs. Um processo de filtragem semelhante a este é realizado com a ajuda da ferramenta TCPDump para a criação de um filtro para o simulador, explorado em detalhes na Seção 4.2.

Depois de testadas e validadas as rotinas de envio e recebimento de pacotes do simulador, concluiu-se que o mesmo apresentava as funcionalidades necessárias para a simulação do LACP. Com isso, um novo cenário de testes foi realizado, utilizando dois computadores pessoais (PC), cada um com uma cópia do simulador e do LACP sendo executados, interagindo uma com a outra por meio da troca de pacotes de controle. Este tipo de teste denomina-se teste simulador/simulador, ou teste SIM/SIM.

Nos testes SIM/SIM, foi possível comprovar a efetividade na comunicação entre os simuladores, novamente observando o Wireshark e, através de funções para exibir os pacotes recebidos no simulador, sendo possível verificar se os pacotes recebidos e enviados pelos simuladores são idênticos aos que trafegam pela rede, capturados pelo Wireshark. Com este teste também é possível verificar o funcionamento do LACP utilizando o simulador para a comunicação entre sistemas.

Na fase final de testes, o LACP executando através do simulador é usado para estabelecer uma comunicação com um *switch* físico real. O *switch* utilizado foi o MRV 2228-S2C [MRV11] Este dá suporte ao LACP e ao *Marker Protocol*, entre outros protocolos. Com este equipamento, pode-se testar principalmente a troca de pacotes de controle entre o simulador e o *switch*, a fim de validar o funcionamento da implementação com um equipamento comercial em um esquema de emulação/simulação conjunta. A Figura 33 mostra uma foto do *switch* em questão.

Figura 33: *Switch* MRV MR2228-S2C utilizado para validação.

Para comprovar o funcionamento correto do simulador e do LACP, foi adicionada uma configuração no *switch* para criar uma agregação entre a porta do switch e a máquina conectada à porta do mesmo. O *switch* oferece dois tipos de configuração: por porta serial, através de linha de comando; e por interface gráfica, acessada por um navegador de internet através do endereço IP previamente configurado no *switch*. Adicionada a configuração, está é então analisada, utilizando novamente o Wireshark, para verificar se a comunicação entre o *switch* e a máquina executando o simulador e o LACP estava ocorrendo como esperado. A Figura 32 ilustra a captura de um LACPDU durante a comunicação entre o *switch* e o LACP, utilizando o simulador rodando em um PC. Em destaque na Figura 32 está um LACPDU gerado pelo LACP utilizando o simulador.

Na Figura 34, pode-se observar o recebimento dos LACPDUs pelo *switch*, o que comprova a validade dos LACPDUs gerados pelo LACP e enviados através do simulador para o *switch*. Logo depois de estabelecida a comunicação e a troca de pacotes de controle periódicos, foi constatada a criação e manutenção da agregação entre o *switch* e o LACP utilizando o simulador.

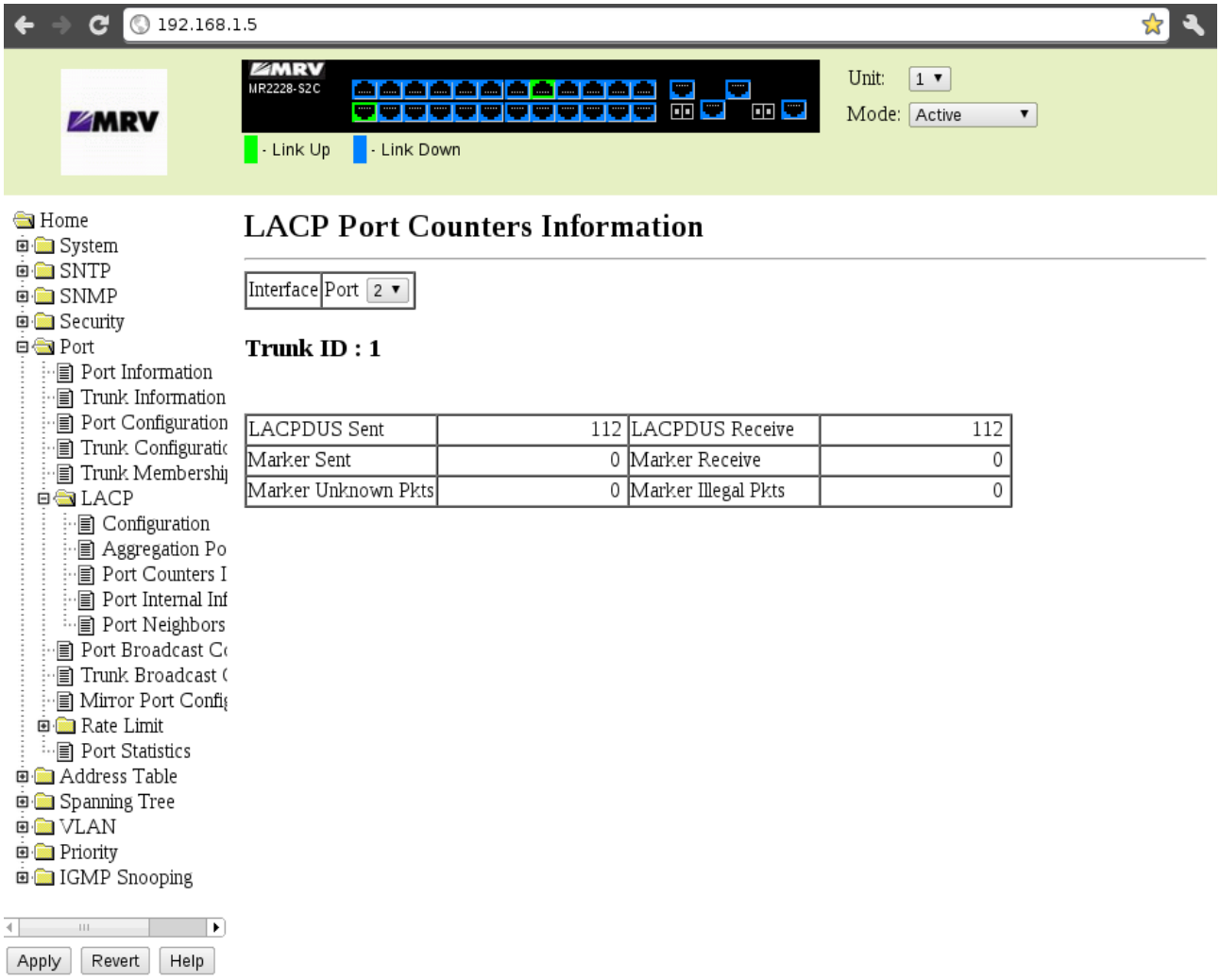


Figura 34: Interface gráfica de configuração do *switch* MRV MR2228-S2C.

A fase final de experimentação coloca novamente dois PCs executando o LACP e o simulador, estabelecendo agregações entre si. As mais diversas composições de agregações são realizadas, sempre mudando o número de portas ativas e o número de agregadores, bem como o número de portas em cada agregador. O número padrão de portas ativas nos testes foi de oito portas, uma vez que o este número é próximo ao número de portas de equipamentos reais no mercado.

Também foram realizados testes exaustivos, onde um número grande de portas foi gerenciado pelo LACP. No maior dos testes, vinte portas formaram uma agregação. Testes com um maior número de portas só não foram possíveis devido à escassez de recursos das máquinas, visto que para serem simuladas vinte portas, o consumo de memória ficou consideravelmente elevado, devido os recursos solicitados pelo simulador e pelo LACP. Destes testes pode-se concluir pela eficiência do simulador e do LACP

trabalhando com um número elevado de portas sendo simuladas e gerenciadas. O simulador claramente foi capaz de dispor os recursos solicitados pelo LACP sem problemas.

6 CONCLUSÕES E TRABALHOS FUTUROS

Com a realização deste trabalho, foi possível comprovar que é possível o desenvolvimento do protocolo de agregação de enlaces LACP sem a utilização de *hardware* específico, através da utilização de um simulador, implementado para suprir as necessidades que o protocolo possui em relação a chamadas de funções exercidas pelo *hardware*. Foi possível também, com a utilização do simulador, a implementação do *Marker Protocol*, parte integrante do LACP e essencial para seu correto funcionamento.

Espera-se concluir o trabalho de Molina [MOL11], inserindo o LACP desenvolvido em ambiente simulado na plataforma de *hardware* na qual o mesmo irá ser comercializado, cumprindo com objetivos da empresa. Na proposta da Parks Comunicações Digitais S/A, o LACP será inserido em seus equipamentos, tornando-se um produto integrante da gama de serviços disponibilizados pelos equipamentos da empresa. Com isso, a empresa agrega valor a seus produtos, uma vez que os torna aptos a realizar agregações utilizando o LACP.

Por se tratar da primeira versão do protocolo criada até hoje, futuramente podem surgir novas versões, trazendo melhorias em seu funcionamento, ou então novas propostas de operabilidade do protocolo, o que abre espaço para novos estudos e futuras implementações do LACP. Uma abordagem interessante seria a de que, em versões futuras, o protocolo contasse com mecanismos de auto ajuste, ou seja, que o próprio LACP analisasse a necessidade de criar agregações, seja para melhorar o fluxo de dados, seja para diminuir a carga em determinado canal. Essa abordagem, porém, requer uma quantidade significativa de estudos, tanto de viabilidade como de operabilidade, uma vez que a gerência de canais de comunicação de uma rede muitas vezes não é uma tarefa simples nem mesmo para o mais experiente gerente de rede.

Outro trabalho futuro possível é a análise do funcionamento conjunto do LACP com o RSTP (*Rapid Spanning Tree Protocol*) [IEE04]. O protocolo RSTP é uma evolução do

STP (*Spanning Tree Protocol*). O STP possibilita incluir na arquitetura da rede enlaces redundantes, a fim de fornecer caminhos alternativos caso um enlace ativo falhe, sem o perigo de ocorrerem laços lógicos entre *bridges* (conexão de vários dispositivos de rede), ou a necessidade de uma ação manual de ativação ou desativação de enlaces físicos alternativos. Laços lógicos entre *bridges* devem ser evitados, pois esta situação resulta em *flooding* de pacotes na rede, ou seja, os pacotes são repassados de maneira *broadcast* (um para todos), sem atingir o destino [PIN09].

O protocolo STP foi projetado quando a recuperação de conectividade após um minuto ou mais era considerada adequada (caso uma *bridge* ou determinado enlace conectado a mesma seja retirado de operação). Com o advento da camada 3 do modelo OSI em *bridges* de rede, estas agora competem com soluções roteadas onde protocolos, como *Open Shortest Path First* (OSPF) e *Enhanced Interior Gateway Routing Protocol* (EIGRP), estão capacitados a fornecer um caminho alternativo em menor tempo.

A ideia de integração entre LACP e RSTP é de que os dois possam atuar em conjunto, sem que hajam interferências na funcionamento de ambos. O desafio nesta integração está no gerenciamento das portas do *switch*, mais especificamente na forma de como LACP e RSTP vão manipulá-las, surgindo questões como:

- Quem tem prioridade de execução?

- Para a criação de um caminho alternativo pelo RSTP, o mesmo deve considerar portas físicas ou agregações?

- Depois de criados os caminhos alternativos, o LACP pode selecionar as portas dos caminhos alternativos para formar agregações?

Na Figura 35, tem-se a ilustração de uma topologia de rede montada com três *switches Ethernet* ligados entre si formando uma conexão em anel. Esta topologia é um exemplo a ser estudado da interação entre LACP e RSTP, pois agrega algumas das questões pertinentes ao funcionamento em conjunto dos dois protocolos.

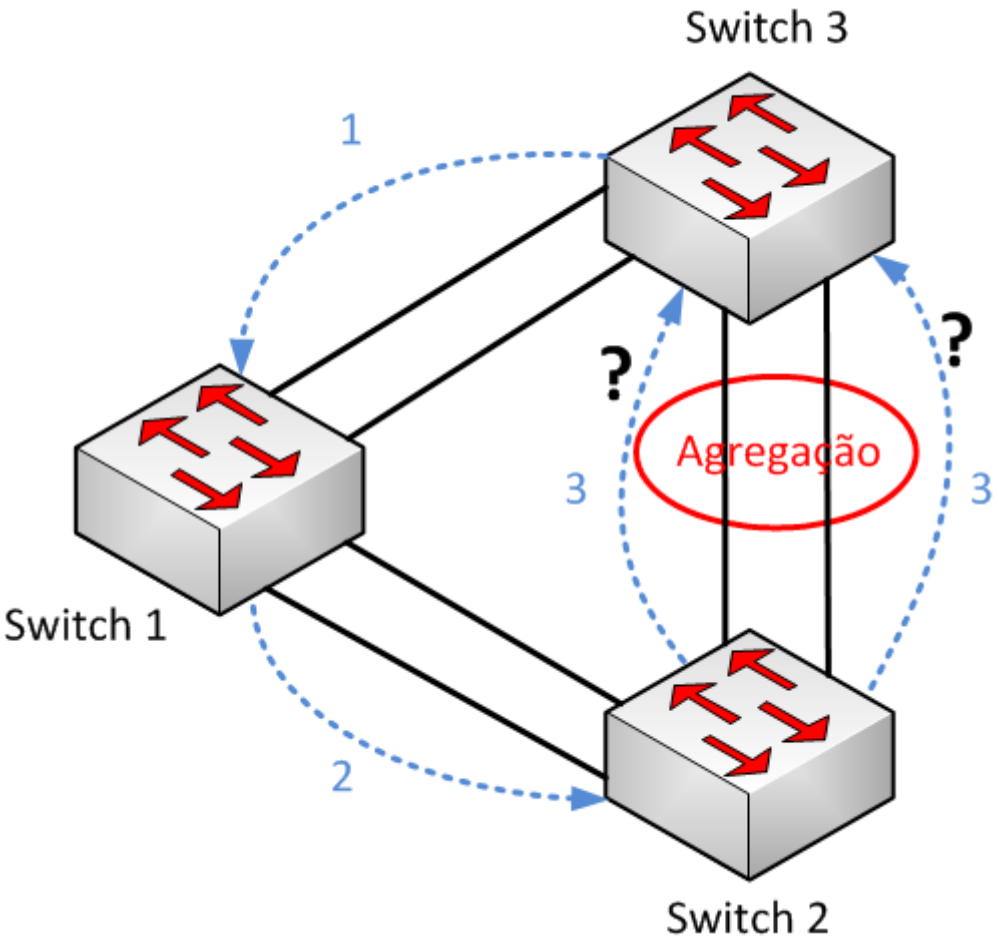


Figura 35: Exemplo de topologia de rede utilizando LACP e RSTP em conjunto.

Considere-se que na topologia da Figura 35 deseja-se utilizar o RSTP para definir os caminhos de comunicação do sistema, representados pelas setas pontilhadas, e também deseja-se utilizar o LACP para formar uma agregação entre os dois enlaces que ligam o Switch 2 ao Switch 3, representado pelos enlaces circulados. A problemática neste exemplo é descrita nos itens acima nesta sessão: não sabemos se o RSTP irá considerar a agregação como um enlace único ou desconsiderará a agregação e montará seu caminho considerando apenas os enlaces físicos reais. Ou ainda, havendo outros canais disponíveis, que não pertençam a qualquer agregação, tais canais sejam utilizados como caminhos alternativos.

Estas questões resumem-se a qual dos algoritmos teria a maior prioridade na configuração das portas, o que definiria a visão que um protocolo teria em relação às portas. Por exemplo, caso o LACP realize a configuração das portas inicialmente, este poderia dispor ao RSTP as configurações que foram atribuídas às portas. Então, o RSTP realizaria suas operações tendo como base as agregações realizadas pelo LACP. Em

outro caso, se o RSTP realizar a descoberta dos caminhos antes que o LACP entre em funcionamento, o LACP teria que levar em consideração os caminhos estabelecidos pelo RSTP quando for realizar as agregações.

Considerando que a empresa Parks Comunicações Digitais S/A já possui o RSTP devidamente implantado em seus produtos e deseja que o mesmo funcione em conjunto com o LACP, dentre outros protocolos, tem-se aí uma nova linha de pesquisa, que pode dar seguimento ao presente trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS

[ETH11] ETHERNET TODAY. “Metro Ethernet Definition”. Capturado em: http://www.ethernettoday.com/articles/metro-ethernet-definition.php, Agosto 2011.

[HUY11] Huynh, M.; Mohapatra, P. “Metropolitan Ethernet Network: A move from LAN to MAN”, Computer Networks, 51(17), Dec. 2007, pp. 4867–4894. Acesso em: http://www.sciencedirect.com/science/article/pii /s1389128607002101, Agosto 2011.

[IEE04] IEEE COMPUTER SOCIETY. “Media Access Control (MAC) Bridges”, STD 802.1d, 2004 Edition, pp 57 – 65.

[IEE07] IEEE COMPUTER SOCIETY. “Requirements for support of Slow Protocols”. Capturado em: http://www.ieee802.org/3/axay/comments/D1.2/8023-57a_b_SG15_response.pdf, Setembro 2007.

[IEE08] IEEE COMPUTER SOCIETY. “Link Aggregation”. Capturado em: http://standards.ieee.org/getieee802/download/802.1AX-2008.pdf, Março 2008.

[MOL11] MOLINA, Diego C. “Estudo e Implementação do Protocolo de Agregação de Enlaces”, Trabalho de Conclusão de Curso I, Curso de Ciência da Computação, FACIN, PUCRS, 2011, 26p.

[MRV11] MRV COMMUNICATIONS. “Datasheet MRV 2228 S2C L2/4 Stackable Switch”. Capturado em: http://www.mrv.com/dl.php?prod=os&type=a4pdf72&file=mrv-os-mr-2228, Setembro 2011.

[PIN09] PINOTTI, Igor K. “Desenvolvimento do Protocolo RSTP – Rapid Spanning Tree Protocol”, Trabalho de Conclusão de Curso, Curso de Engenharia de Computação, FENG/FACIN, PUCRS, 2009, 80p.

[PAC11] PACKETH. “PackETH - Ethernet Packet Generator”. Acesso em: http://packeth.sourceforge.net/, Setembro de 2011.

[TAN03] Tanembaum, Andrew S.. “Computer Networks”. Upper Saddle River: Prentice Hall, 2003, 4 ed., 891p.

[TCP11] TCPDUMP. “Tcpdump man page”. Acesso em: http://www.tcpdump.org/tcpdump_man.html, Setembro de 2011.

[VAL11] VALGRIND. “About Valgrind”. Acesso em: http://valgrind.org/info/about.html, Setembro 2011.

[WIK11a] WIKIPÉDIA. “Link Aggregation”. Acesso em: http://en.wikipedia.org/wiki/link_aggregation, Agosto 2011.

[WIK11b] WIKIPEDIA. “Channel Bonding”. Acesso em: http://en.wikipedia.org/wiki/channel_bonding, Agosto 2011.

[WIK11c] WIKIPÉDIA. “Modelo OSI”. Acesso em: http://pt.wikipedia.org/wiki/modelo_osi, Agosto 2011.

[WIR11] WIRESHARK. “Wireshark User's Guide”. Acesso em: http://www.wireshark.org/docs/wsug_html_chunked/, Setembro 2011.