PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE ENGENHARIA
FACULDADE DE INFORMÁTICA
CURSO DE ENGENHARIA DA COMPUTAÇÃO

Desenvolvimento do Protocolo RSTP – Rapid Spanning Tree Protocol

IGOR KRAMER PINOTTI

TRABALHO DE CONCLUSÃO DE CURSO

Prof. Dr. Fernando Gehm Moraes
Orientador

Porto Alegre, 18 de dezembro de 2009.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1. INTRODUÇÃO

Antes do advento das redes de computadores [RED09] baseadas em algum tipo de sistema de telecomunicação, a comunicação entre máquinas calculadoras e também computadores antigos era realizada por pessoas, através do transporte manual (força humana) de instruções entre eles.

Em setembro de 1940, George Stibitz usou uma máquina de teletipo para enviar instruções para um conjunto de problemas a partir de seu Model K na Faculdade de Dartmouth em Nova Hampshire para a sua Calculadora de Números Complexos em Nova Iorque e recebeu os resultados de volta pelo mesmo meio. Conectar sistemas de saída como teletipos a computadores era um interesse da Advanced Research Projects Agency (ARPA) quando, em 1962, J. C. R. Licklider foi contratado e desenvolveu um grupo de trabalho o qual ele chamou de a "Rede Intergaláctica", um precursor da ARPANet.

Em 1964, pesquisadores de Dartmouth desenvolveram o Sistema de Compartilhamento de Tempo de Dartmouth para usuários distribuídos de grandes sistemas de computadores. No mesmo ano, no MIT, um grupo de pesquisa apoiado pela General Electric e Bell Labs usou um computador (DEC PDP-8) para rotear e gerenciar conexões telefônicas.

Durante a década de 1960, Leonard Kleinrock, Paul Baran e Donald Davies, de maneira independente, conceituaram e desenvolveram sistemas de redes os quais usavam datagramas ou pacotes, que podiam ser usados em uma rede de comutação de pacotes entre sistemas de computadores.

Em 1969, a Universidade da Califórnia em Los Angeles, SRI (em Stanford), a Universidade da Califórnia em Santa Bárbara e a Universidade de Utah foram conectadas com o início da rede ARPANet usando circuitos de 50 kbits/s.

Atualmente, redes de computadores são o núcleo da comunicação moderna. O escopo da comunicação cresceu significativamente na década de 1990 e essa explosão nas comunicações não teria sido possível sem o avanço progressivo das redes de computadores.

Uma rede de computadores proporciona um meio poderoso de trocas de informação devido à velocidade, confiabilidade e compartilhamento de recursos que proporciona. Assim, um dos objetivos de uma rede de computadores é tornar disponíveis aos seus usuários todos os aplicativos, dados e outros recursos, independente de sua localização física e, ainda, proporcionar uma maior flexibilidade ao sistema, dada a possibilidade de migração da operação para outros equipamentos quando algum dispositivo apresentar defeito.

Dada a importância que a informação vem adquirindo na sociedade atual, tê-la rapidamente ao alcance das mãos e repassá-la é uma necessidade urgente. Neste contexto, a sua democratização

passa necessariamente pelo seu compartilhamento através de Redes de Computadores. A sua importância para o mundo se traduz na capacidade de colocar pessoas em contato muito mais rapidamente, possibilitar a comunicação individual e a coletiva (conferências e encontros), possibilitar a resolução de problemas de trabalho e até de natureza social e política, interação do trabalho das pessoas a longa distância, compartilhamento de recursos como impressoras, discos, acesso remoto de programas, e meio para troca de mensagens.

Com essa necessidade constante de conexão com a rede mundial de computadores (nuvem da internet) e o aumento intenso da complexidade das redes de computadores, crescem as preocupações de gerenciar e administrar os pacotes que trafegam pela rede, além de exigir uma melhoria na segurança e no fluxo de dados contínuo, sem haver perdas. Assim, o trabalho realizado consiste no desenvolvimento do protocolo RSTP, visando à conexão de vários dispositivos de rede (*bridges)*, evitando possíveis laços lógicos de pacotes causados por conexões físicas redundantes entre os dispositivos.

1.1 Motivação

A rede *Ethernet* [ETH09] é uma tecnologia de interconexão para redes locais - *Local Area Networks* (LAN) - baseada no envio de pacotes. Ela define cabeamento e sinais elétricos para a camada física, e formato de pacotes e protocolos para a camada de controle de acesso ao meio (*Media Access Control* - MAC) do modelo OSI. A *Ethernet* foi padronizada pelo IEEE como 802.3.

Normalmente, em uma rede *Ethernet* local duas estações estão logicamente interligadas entre si através de um caminho único. Em tempos passados, este tipo de situação era mandatório, sob pena de ocorrerem laços na rede, o que atrapalhava o mecanismo de encaminhamento de pacotes de *bridges Ethernet* conectadas à rede, uma vez que os mesmos poderiam detectar pacotes de um mesmo endereço de máquina chegando por duas portas diferentes, o que normalmente não é possível, pois tais endereços são únicos. A interligação de dispositivos em uma rede *Ethernet* realizada por vários caminhos lógicos sem um controle específico gera um acumulo de pacotes sendo repassados incessantemente pelos dispositivos, sem atingir o destino proposto. Este resultado acarreta na inutilização da rede *Ethernet*, pois a conexão é totalmente ocupada por pacotes *broadcast* (gera um *flood* de pacotes), na tentativa de achar o destino para os mesmos.

Para a finalidade de interligar redes *ethernet*, existem dois tipos de dispositivos que operam em nível de enlace (camada 2 OSI), o *switch* e a *bridge*. Antigamente, estes dispositivos tinham especificações distintas. A *bridge* possuía no máximo três portas, separando apenas três enlaces *Ethernet*. O *switch* já possui várias portas, separando vários enlaces *Ethernet*. Hoje em dia, a função destes aparelhos está unificada, sendo indiferente o termo utilizado para designá-los, pois em ambos a finalidade é a mesma. Neste trabalho, será adotado o termo *bridge*.

Com o aumento da quantidade de dispositivos (*hosts*) de rede, o uso de uma única *bridge* torna-se impraticável, ou seja, existem muitos computadores para a capacidade suportada nas

bridges, gerando a necessidade de interligação entre as *bridges*. É comum duas ou mais *bridges* interligando segmentos de uma rede local. A Figura 1 ilustra esta situação. Pode-se observar na Figura 1 que os segmentos de rede estão ligados entre si por vários caminhos. A vantagem deste tipo de ligação reside no fato de se criarem caminhos alternativos entre os segmentos. Esta redundância torna-se útil quando ocorre a falha de uma *bridge* na rede. Neste caso, os pacotes podem ser encaminhados por uma *bridge* que esteja em um caminho redundante. Por outro lado, topologias deste tipo podem causar laços lógicos bem como duplicidade de pacotes na rede.

Um dispositivo *Ethernet* pode estar conectado a rede por diversos enlaces físicos. Mas para que uma rede *Ethernet* funcione corretamente, só deve existir um caminho lógico que conecte dois dispositivos, sob pena de ocorrer laços lógicos de dados. Para resolver o problema dos laços lógicos gerado por *flood* de pacotes *broadcast*, problema originado por várias ligações físicas entre *bridges*, ilustrado na situação apresentada na Figura 1, e ainda utilizar a vantagem da redundância de caminhos, o IEEE propôs o *Spanning Tree Protocol and Algorithm* (STP), definido em IEEE 802.1D – MAC *bridges* [INS04].

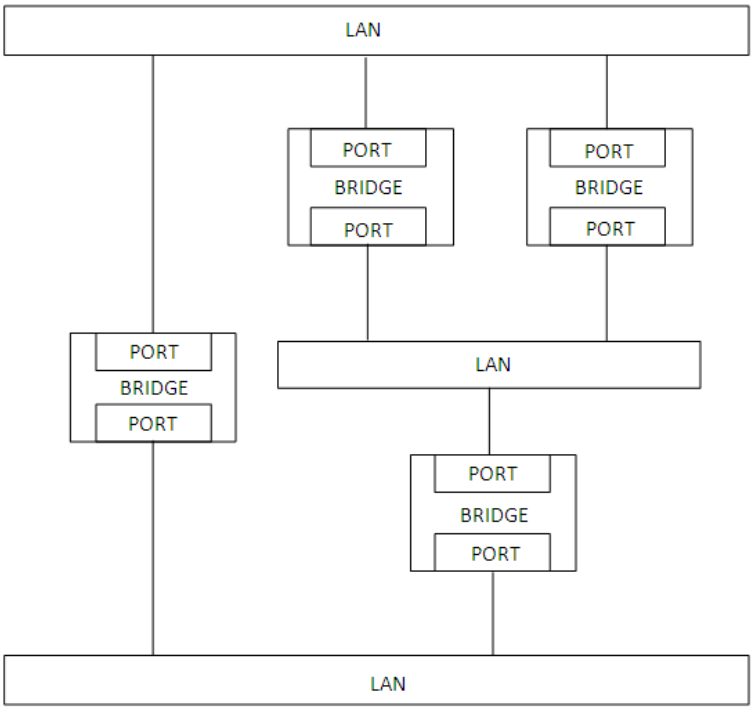


Figura 1 – Estrutra de rede interconectada por várias *bridges*.

Além da importância deste protocolo para a administração das redes de computadores e seu correto funcionamento em topologias com diversas *bridges* e caminhos redundantes, a empresa PARKS, através de um termo de cooperação com a PUCRS, está financiando este trabalho com o objetivo de utilizar este protocolo em seus dispositivos de nível de enlace.

1.2 Objetivo

O presente trabalho tem por objetivo descrever o desenvolvimento, em software, do protocolo RSTP (Rapid Spanning Tree Protocol) – IEEE Standard 802.1w [INS01]. A implementação do protocolo foi realizada como uma aplicação executando sob sistema operacional Linux, em espaço de usuário, criando-se uma camada de abstração de acesso ao *hardware* (HAL) da *bridge* (*switch Ethernet* - conjunto de 4 portas ethernet da plataforma de desenvolvimento). A plataforma de desenvolvimento é a MPC8323E *Modular Development System* (MDS), da FreeScale. O processo de desenvolvimento do protocolo RSTP foi conduzido utilizando a linguagem C++, nos sistemas operacionais Linux Ubuntu e Debian, com versão de *kernel* 2.6. O resultado é um software de gerência de *bridges*, compatível com o protocolo RSTP, executando como se fosse um *deamon* em Linux embarcado.

1.3 Arquitetura Desenvolvida e Estrutura do Documento

O protocolo RSTP foi desenvolvido de forma concomitante em dois ambientes diferentes, conforme ilustrado na Figura 2. Primeiro, executando em uma estação Linux, onde o enlace entre os vários terminais com RSTP é o *Simulador de Conexões* (simulador referenciado na Figura 2), um programa que faz a função de virtualizar os fios entre *bridges*. O segundo ambiente executa na plataforma de desenvolvimento da Freescale, com *drivers* adaptados para fazer a ligação entre o protocolo e o *hardware* da plataforma, executando como uma aplicação de usuário (camada 7 OSI).

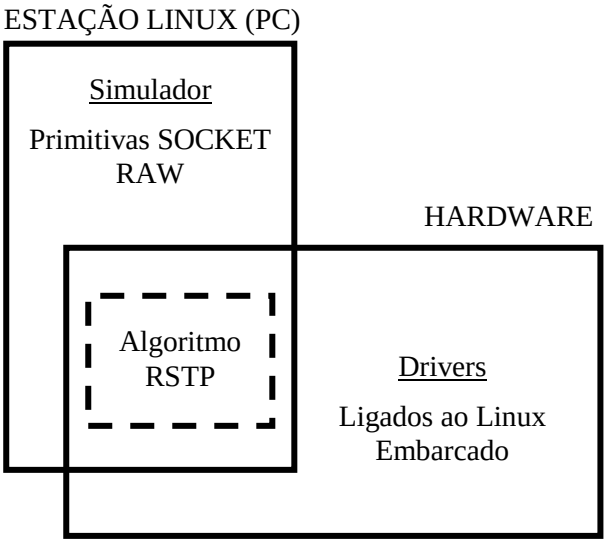


Figura 2 – Abstração dos ambientes Simulador de Conexões, algoritmo RSTP em sí e a implementação em *hardware* do RSTP.

O Capítulo 2 apresenta o protocolo RSTP e suas características. O Capítulo 3 apresenta a implementação do algoritmo RSTP, detalhando o funcionamento do algoritmo. O Capítulo 4 apresenta o algoritmo RSTP sendo simulado no *Simulador de Conexões*, mostrando seu

funcionamento e as características do sistema de comunicação entre o simulador e o algoritmo. O Capítulo 5 descreve a integração do algoritmo RSTP com a plataforma de desenvolvimento MPC8323E descrevendo o *hardware*, e resultados da execução do algoritmo em *hardware*. O Capítulo 6 aborda conclusões finais do trabalho.

2. PROTOCOLO RSTP

O protocolo *Spanning Tree* possibilita incluir na arquitetura da rede enlaces redundantes a fim de fornecer caminhos alternativos caso um enlace ativo falhe, sem o perigo de ocorrer laços lógicos entre as *bridges*, 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*, sem atingir o destino.

O protocolo STP, em inglês *Spanning Tree Protocol* (norma IEEE 802.1D) [INS04], 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 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.

O protocolo RSTP, em inglês *Rapid Spanning Tree Protocol* (norma IEEE 802.1w) [INS01] é uma evolução do STP, que manteve a terminologia do padrão 802.1D, sendo que a maioria dos parâmetros ficou inalterada. Assim, usuários familiarizados com o modelo 802.1D podem rapidamente configurar o protocolo RSTP. O padrão 802.1w pode também retroceder para o padrão 802.1D com a idéia de interoperar com todo o legado de *bridges* que não possuem o novo sistema.

No sistema do protocolo RSTP, pacotes de controle denominados BPDU, são trocados entre as *bridges* para transmitir informações referentes ao estado de topologia do protocolo, contendo dados sobre as portas da *bridge*, endereço MAC, prioridade, custo do enlace, além da garantia de que o dado seja entregue. A seção 2.1 detalha estes pacotes.

As *bridges* possuem várias portas para conectar enlaces de outras *bridges* ou terminais, como PCs. Assim, no RSTP, cada porta tem suas características que interagem com o protocolo, sendo peça fundamental para a eliminação de laço na rede. A seção 2.2 detalha as portas da *bridge* no protocolo.

O protocolo RSTP funciona com base em um sistema de convergência de topologia, que elege uma *bridge* principal (*bridge* raiz – *root bridge*), portadora da melhor configuração dentre as demais do segmento. Após a convergência (configuração das portas das *bridges* permanecem inalteradas por certo tempo), é gerado um caminho lógico único entre os dispositivos através de uma topologia em forma de árvore, para as rotas dos pacotes que trafegam na rede. O algoritmo RSTP opera de forma distribuída, sendo executado em cada *bridge* do sistema. A seção 2.3 detalha o sistema de topologia.

2.1 BPDU (*Bridge Protocol Data Unit)*

O RSTP funciona a partir da troca de pacotes chamados BPDU (*Bridge Protocol Data Unit).* Este pacote transporta informações de controle que viabilizam às *bridges* coordenar suas ações para implementar a funcionalidade do RSTP, garantindo assim um correto funcionamento das ligações lógicas redundantes. Sintetizando, os pacotes BPDUs viabilizam a administração automatizada da conectividade lógica do protocolo RSTP nas *bridges*.

BPDUs são enviados pelas *bridges*, por todas as suas portas, a cada <*hello-time*> onde o tempo padrão segundo a norma IEEE é dois segundos. A *bridge* envia um BPDU com suas informações correntes a cada <*hello-time*> segundos, mesmo que ela não receba BPDUs da *root bridge*.

Em uma dada porta, se BPDUs não são recebidos ao longo de três <*hello-time*>, a informação armazenada (configuração) na porta referente a esses BPDUs é imediatamente expirada (ou se *max_age* expirar – tempo de vida máximo da informação). BPDUs são usados como um mecanismo de *keep-alive* ("manter vivo", tradução literal) entre *bridges*. Uma *bridge* considera que perdeu conectividade com a *root bridge* ou *designated bridge* se ela perder três BPDUs em seqüência. Este rápido envelhecimento de informação permite rápida detecção de falhas. Se uma *bridge* falhar em receber BPDUs de um vizinho, é certo que a conexão com o vizinho estará perdida. Caso um enlace seja desconectado da *bridge*, ou ocorra algum problema físico com o enlace, esta falha é imediatamente refletida para a execução local do algoritmo do protocolo RSTP.

Estes pacotes exercem função fundamental para a convergência, pois é pela comparação entre os diferentes BPDUs enviados por todas as *bridges* na rede, que se elege a raiz da árvore da topologia, a *root Bridge* (*bridge* portadora das melhores configurações). A *root bridge* é a *bridge* que envia o melhor BPDU dentre as demais, ou seja, com menor valor numérico nos campos do pacote BPDU.

Existem três tipos de BPDUs utilizados no protocolo RSTP. O BPDU padrão RSTP versão dois, tem sua estrutura dada na Figura 3 [INS04] enquanto suas *flags* aparecem na Figura 4 [CIS09], correspondendo ao campo *FLAGS* da Figura 3. Os outros dois tipos de BPDU são utilizados para compatibilidade com o protocolo STP, o BPDU padrão STP versão zero e o BPDU TCN (BPDU Topology Change Acknowledgment), utilizado para sinalizar mudança de topologia na rede por *bridges* STP.

Descrição dos campos do BPDU RSTP versão dois:

- ***Protocol Identifier:*** identificador do protocolo utilizado;
- ***Protocol Version Identifier***: identificador da versão do protocolo utilizado;
- ***BPDU Type***: identifica o tipo de BPDU utilizado, versão STP 802.1d ou RSTP 802.1w;
- ***Flags***: segundo a Figura 4, guarda as *flags* correspondentes ao estado da porta, papel da porta, mudança de topologia e sistema de "*handshake"* para ocorrer mudança de estado nas portas, chamado *Proposal/Agreement*;

- ***Root Identifier*:** identifica a *root bridge* da topologia. O campo é composto por um ID, que por sua vez, é composto por dois dados:
    - **MAC**: Atributo que representa o endereço de cada *bridge*. O MAC de uma *bridge* é único na rede, ou seja, em nenhuma outra *bridge* da rede se encontrará o mesmo valor de MAC.
    - **Prioridade**: Número que define a prioridade que cada *bridge* possui na rede. Pode ser alterado pelo administrador. A alteração deste atributo pode influenciar diretamente na topologia que será definida pelo algoritmo.

  Cada *bridge* deve possuir um identificador único, composto então pelo agrupamento da prioridade com o MAC da mesma:

  EX: Prioridade = 32000 MAC = 001731bad8ab ID = 32000001731bad8ab

- ***Root Path Cost***: armazena a soma dos custos de enlace até a *root bridge* da topologia.
- ***Bridge Identifier:*** identifica a *bridge* que está enviando o BPDU. É também composto por um ID, com seus campos de MAC e prioridade agrupados.
- ***Port Identifier***: identifica a porta que enviou o BPDU;
- ***Message Age***: contador de "tempo de vida" da informação carregada no BPDU;
- ***Max Age***: tempo máximo ajustável para que a informação carregada no BPDU seja expirada;
- ***Hello Time***: intervalo de tempo em que o BPDU é enviado pela *bridge*, 2 segundos por padrão*;*
- ***Forward Delay***: tempo estipulado para a porta exercer o estado *forward*;
- ***Version 1 Length***: atribuído valor zero, significando que o BDPU não corresponde à versão 1 (STP).

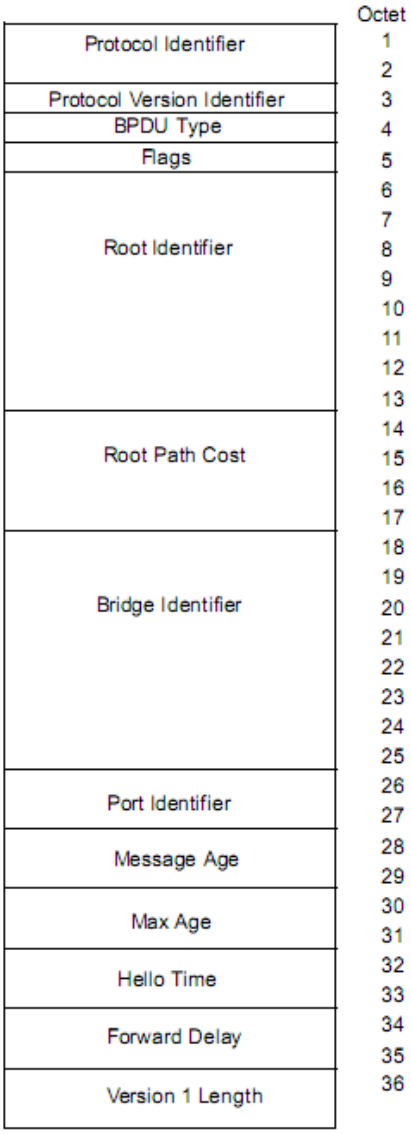


Figura 3 – Campo de dados do BPDU.

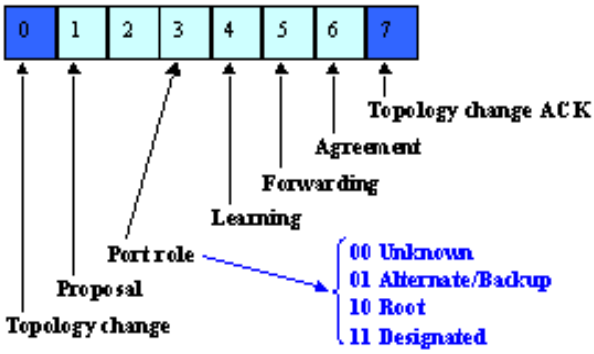


Figura 4 – *Flags* do BPDU.

O BPDU STP versão zero utiliza a mesma estrutura do BPDU RSTP, excluindo somente o campo *Version 1 Length*. No campo de *flags*, é considerado apenas o primeiro (zero) e o último (sete) *bit* do octeto de *flags* (destacados na Figura 4). O BPDU TCN utiliza estrutura distinta, apresentado na Figura 5 [INS04]. Os campos apresentados no BPDU TCN têm o mesmo propósito dos campos existentes no BPDU RSTP e STP, apenas configurados com valores diferentes. O BPDU TCN é enviado pelo protocolo STP em casos de mudança de topologia, sinalizando esta alteração as demais *bridges* do segmento.

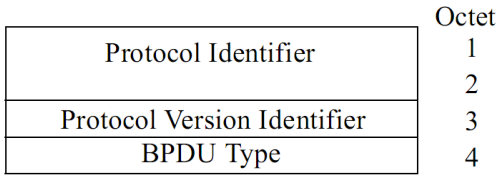


Figura 5 – Campo de dados do BPDU TCN.

2.2 Portas

Cada porta da *bridge* tem **papéis** e **estados** que lhe são atribuídos para o funcionamento do algoritmo RSTP, ação que é coordenada pelos BPDUs recebidos pelas *bridges*. Os estados das portas no RSTP podem ser Discarding, significando que a porta está devolvendo pacotes de dados diferentes de BPDUs, *Learning*, significando que a porta está aprendendo informações de pacotes de configuração, e *Forwarding*, significando que a porta está repassando pacotes de dados diferentes de BPDUs. No modelo 802.1D - STP, os estados *Blocking e Listening* existem de forma separada. No modelo 802.1w - RSTP, eles são fundidos no estado denominado *Discarding*. A Tabela 1 [CIS09] a seguir ilustra os estados das portas, comparando o protocolo STP com o protocolo RSTP em questão.

Tabela 1 – Descrição dos estados das portas do STP/RSTP.

STP (802.1D) Port State	RSTP (802.1w) Port State	A porta está incluída na topologia ativa?	A porta está aprendendo endereços MAC?
Disabled	Discarding	Não	Não
Blocking	Discarding	Não	Não
Listening	Discarding	Sim	Não
Learning	Learning	Sim	Sim
Forwarding	Forwarding	Sim	Sim

O algoritmo determina o papel de uma porta baseado em BPDUs. O BPDU recebido é analisado com as informações armazenadas na porta da *bridge*. O papel da porta que recebeu o BPDU é eleito a partir da comparação entre os valores armazenados na porta e no BPDU recebido.

Os parágrafos a seguir descrevem os estados das portas. Cada estado é ilustrado por uma figura, sendo a legenda para a compreensão dos mesmos apresentados na Figura 6.

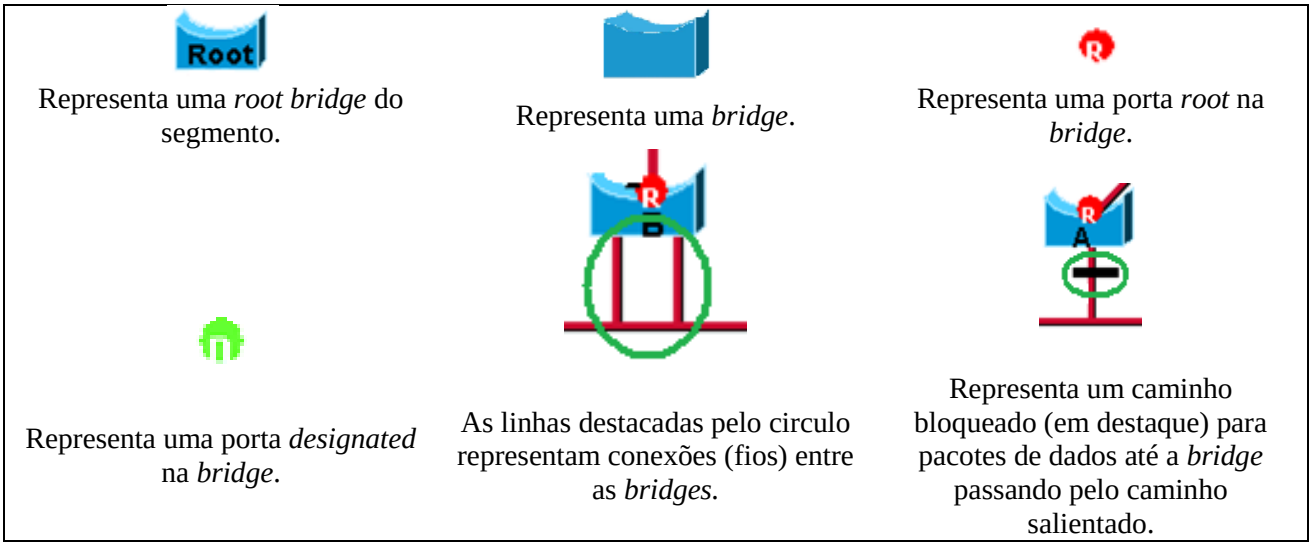


Figura 6 – Legenda utilizada para a discussão do conceito de estado de portas.

- ***Root Port:*** A porta que recebe o melhor BPDU (portador de melhor configuração) em uma *bridge* é a porta *root* (raiz) (Figura 7) [CIS09]. Esta é a porta que está mais próxima da *root bridge* em termos de custo de caminho (*path cost*), e está presente em todas *bridges designated*. O STA

(*Spanning Tree Algorithm*) elege uma única *root bridge* em todo segmento de rede contendo *bridges*. A *root bridge* transmite BPDUs com informações de melhor valor que outros BPDUs enviados por qualquer outra *bridge*. A *root bridge* possui propriedades de ser a única *bridge* na rede que não possui uma porta *root*.

Figura 7 – Exemplo de porta *Root*.

- ***Designated Port:*** Uma porta é *designated* se a mesma transmite o melhor BPDU no segmento em que está conectada (Figura 8) [CIS09]. O modelo STP de *bridges* une diferentes segmentos, como segmentos *Ethernet*, para criar um domínio *bridged*. Em um dado segmento, só pode haver um caminho lógico por dispositivo até a *root bridge*. Se existirem dois ou mais caminhos, existirão laços na rede (*bridging loops*). Todas as *bridges* conectadas em um dado segmento escutam os BPDUs de cada uma e concordam com a *bridge* que transmite o melhor BPDU como a *designated bridge* para este segmento. A porta correspondente desta *bridge* é a *designated port* para este segmento.

Figura 8 – Exemplo de porta *Designated*.

- ***Backup* e *Alternate Port*:** Uma porta *alternate* (alternada) recebe BPDUs mais eficientes de outra *bridge*, bloqueando a porta em questão (porta *blocked)*. Uma porta *backup* recebe BPDUs mais eficientes da própra *bridge* em que está conectada, bloqueando esta porta (porta *blocked*). Estes papéis estão sempre atrelados ao estado *blocked*. Uma porta *blocked* é definida como não sendo a porta *designated* ou *root*, e recebe um BPDU mais eficiente que os BPDUs que envia no seu segmento. Esta porta *blocked* precisa receber BPDUs a fim de se manter no estado *blocked*.

  A diferença entre estes papéis pode ser resumida da seguinte forma, ilustrada na Figura 9 [CIS09] e Figura 10 [CIS09]:

  - Porta *alternate*: fornece um caminho alternativo para a *root bridge* e, portanto, pode

substituir a porta *root* se esta falhar.

- Porta *backup*: fornece conectividade redundante para a mesma *bridge* (mesmo segmento) e não garante uma conectividade substituta (*alternate)* para a *root bridge*.

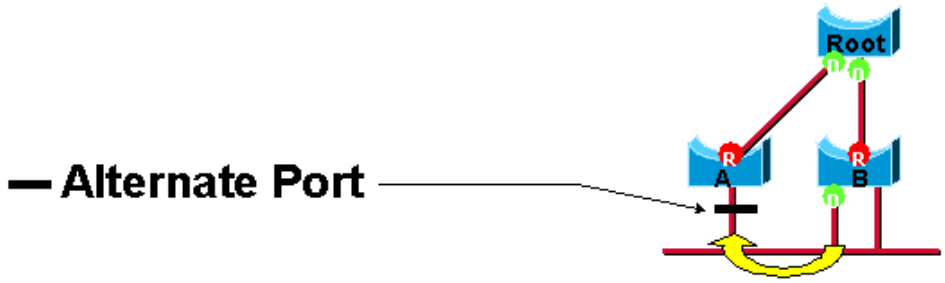


Figura 9 – Exemplo de porta *Alternate*.

Figura 10 – Exemplo de porta *Backup*.

- ***Edge Port***: Todas as portas diretamente conectadas a estações finais (hosts) não podem criar laços lógicos entre *bridges* na rede. Portanto, uma porta *edge* transita diretamente para o estado *forwarding*, ignorando os estágios de *listening* e *learning*. Portas *edge* não geram mudanças na topologia quando o enlace é ativado ou desativado, sendo assim, não influenciam no protocolo RSTP. Porém, quando uma porta *edge* recebe um BPDU, imediatamente perde o estado de porta *edge*, tornando-se uma porta com configurações *spanning tree*.

Como resultado, o RSTP calcula a topologia final para o *spanning tree* que utiliza o mesmo critério do modelo STP. Não há nenhuma mudança na maneira como diferentes *bridges* e prioridades de portas são utilizadas. O nome *blocking* (STP) é usado para o estado *discarding* (RSTP - ou seja, ignorando pacotes de dados). Entretanto, a nova função gera uma diferença entre o papel que o protocolo determina para uma porta e seu estado corrente. Por exemplo, é perfeitamente válido para uma porta ser *designated* e *blocking* ao mesmo tempo. Esta situação ocorre tipicamente por um pequeno período de tempo, e significa que esta porta está em um estado transitório, em direção ao estado *designated forwarding*.

2.3 Definição da Topologia

Nesta seção é apresentado um exemplo de topologia com conexões redundantes, onde é aplicado o algoritmo de *spanning tree,* na parte inicial, para eleição da *Root bridge* do segmento. As Figuras abaixo (Figura 11, Figura 12, Figura 13 e Figura 14) apresentam os símbolos utilizados nos gráfico das topologias que serão apresentadas. Esta convenção é a mesma utilizada na norma 802.1D-2004 [INS04].

A Figura 15 [INS04] mostra um exemplo de topologia com laços. As *bridges* contêm um identificador (ex: 555) e portas (ex: 1). As portas de cada *bridge* interligam outras *bridges* através de enlaces. Entre uma *bridge* e outra podem existir redes (Exemplos: E, D, A, B).

Na inicialização, todas as portas estão descartando pacotes, mas logo passam a transmitir e receber BPDUs com as informações necessárias para construir a *Spanning Tree*. As *bridges* guardam em cada porta o melhor BPDU recebido para fazer o cálculo dos papéis das portas (*root*, *designated*, *alternate* ou *backup*). As *bridges* ficam aprendendo as informações da rede através de constantes trocas de BPDUs (pacotes de controle), sempre atualizando as informações de cada porta quando recebem uma informação melhor do que a atual armazenada.

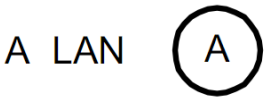


Figura 11 – Representação de uma LAN denominada "A".

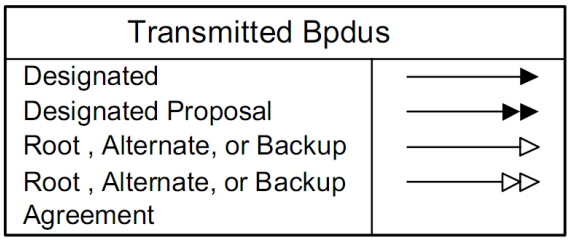


Figura 12 – Legenda para os papeis de portas e tipo de BPDUs transmitidos.

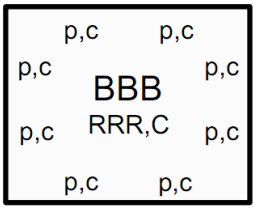


Figura 13 – Representação de uma *bridge*, mostrando o ID (BBB), ID da *root bridge* e o *root path cost* (RRR, C), e identificadores das portas (p) e seus *port costs* (c).

Port Role	Port State	Legend
Designated	Discarding	
	Learning	
	Forwarding	
& operEdge	Forwarding	
Root Port	Discarding	
	Learning	
	Forwarding	
Alternate	Discarding	
	Learning	
	Forwarding	
Backup	Discarding	
	Learning	
	Forwarding	
Disabled	-	

Figura 14 – Legendas para as combinações (papel, estado) de cada porta.

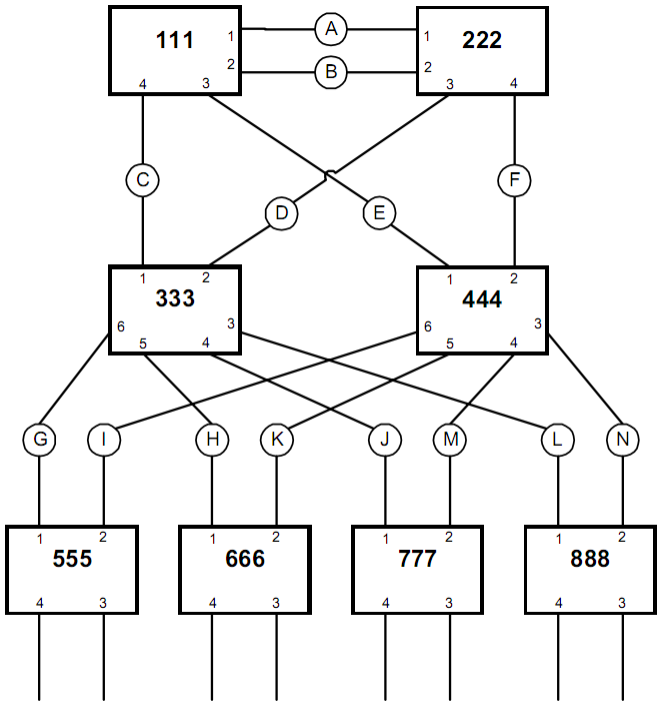


Figura 15 – Exemplo de topologia com laços.

O melhor BPDU recebido em cada porta é armazenado no vetor de prioridade (PPV – *Port Priority Vector*) que cada porta possui. O PPV é composto pelos campos:

- *Root Identifier* (ID): Identificador da *root bridge* do segmento.
- *Root Path Cost*: Soma do custo dos enlaces que levam a *root bridge*.
- *Designated Bridge Identifier* (ID): *Bridge* emissora do BPDU recebido.
- *Designated Port Identifier* (ID): Porta da *bridge* emissora do BPDU recebido.
- *Bridge Port Identifier* (ID): Porta da *bridge* em questão que recebeu o BPDU.

É feita uma comparação entre os PPVs das portas, e o melhor é armazenado no vetor de prioridade da *bridge* (BPV – Bridge Priority Vector). Em uma *designated bridge* (qualquer *bridge* diferente de *root bridge)*, a porta correspondente aos dados do BPV será a porta *root*.

Considerar a *bridge* 111 conectada à *bridge* 222, ativas e descartando pacotes. A Figura 16 mostra as configurações iniciais das *bridges* 111 e 222. Também se considera que cada enlace tenha custo 10. Na inicialização, todas as *bridges* assumem inicialmente serem elas a *root bridge* do segmento.

Após a inicialização das *bridges*, elas passam a aprender as informações da rede por troca de mensagens. Supondo que a *bridge* 222 é a primeira a transmitir os BPDUs, ela informa em suas mensagens as configurações iniciais que estão armazenadas na *bridge* (Figura 16).

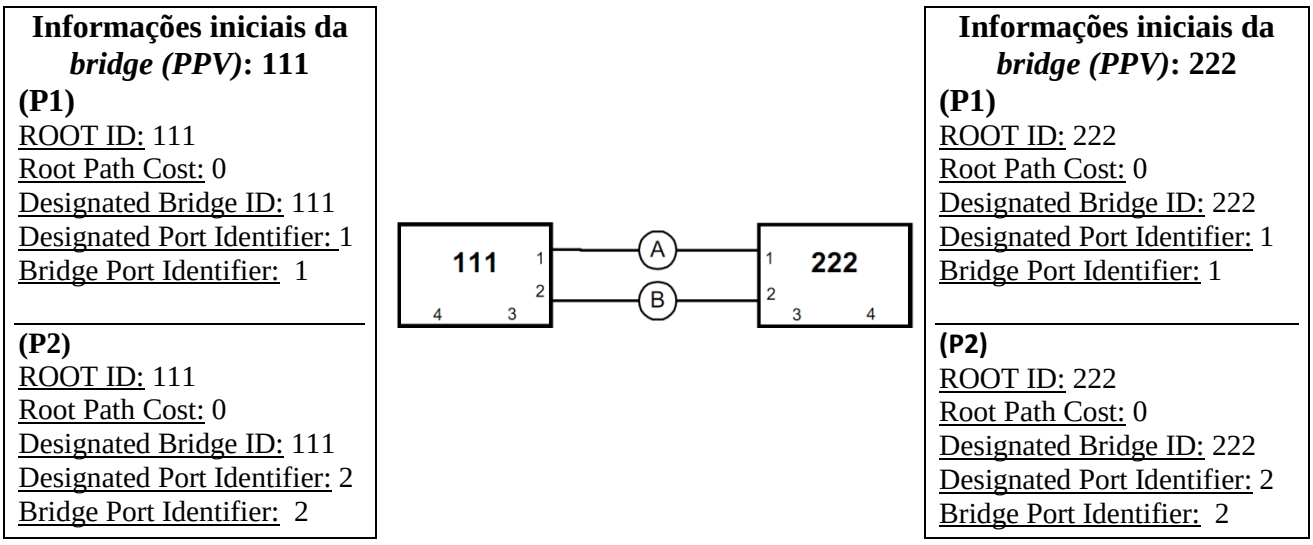


Figura 16 – Configurações inicias das *bridges*.

A Figura 17 mostra a *bridge* 222 informando suas informações para seus vizinhos. No caso, para a *bridge* 111.

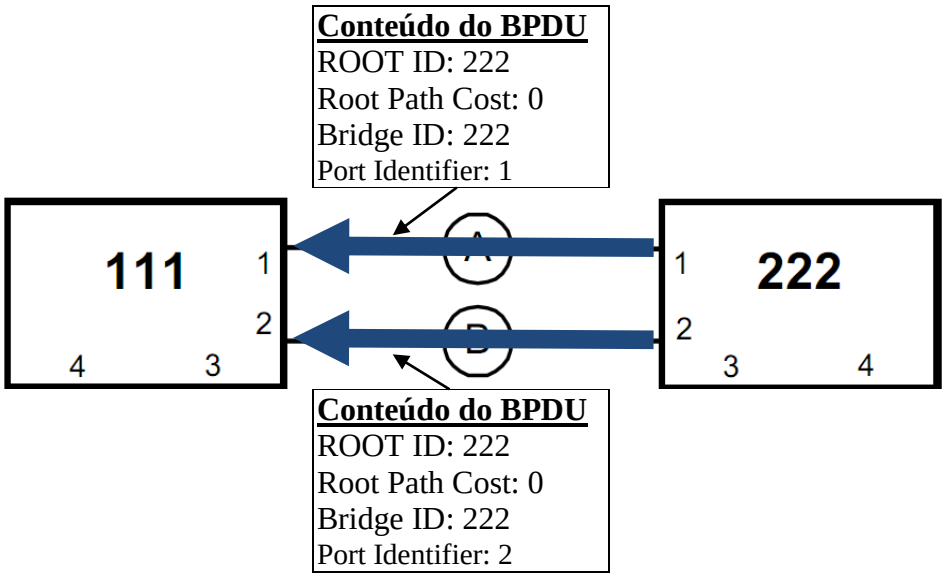


Figura 17 – *Bridge* 222 enviando suas informações iniciais.

Os BPDUs que a *bridge* 222 envia contêm informações piores (informações de valores mais altos), comparando com as informações guardadas no PPV da *bridge* 111:

Bridge 111 ➔ porta 1: 111 – 0 – 111 – 1 <-> porta 2: 111 – 0 – 111 – 2

Bridge 222 ➔ porta 1: 222 – 0 – 222 – 1 <-> porta 2: 222 – 0 – 222 – 2

Então, baseado na comparação, a *bridge* 111 não altera suas informações e imediatamente informa à *bridge* 222 as suas informações superiores, como é exemplificado na Figura 18.

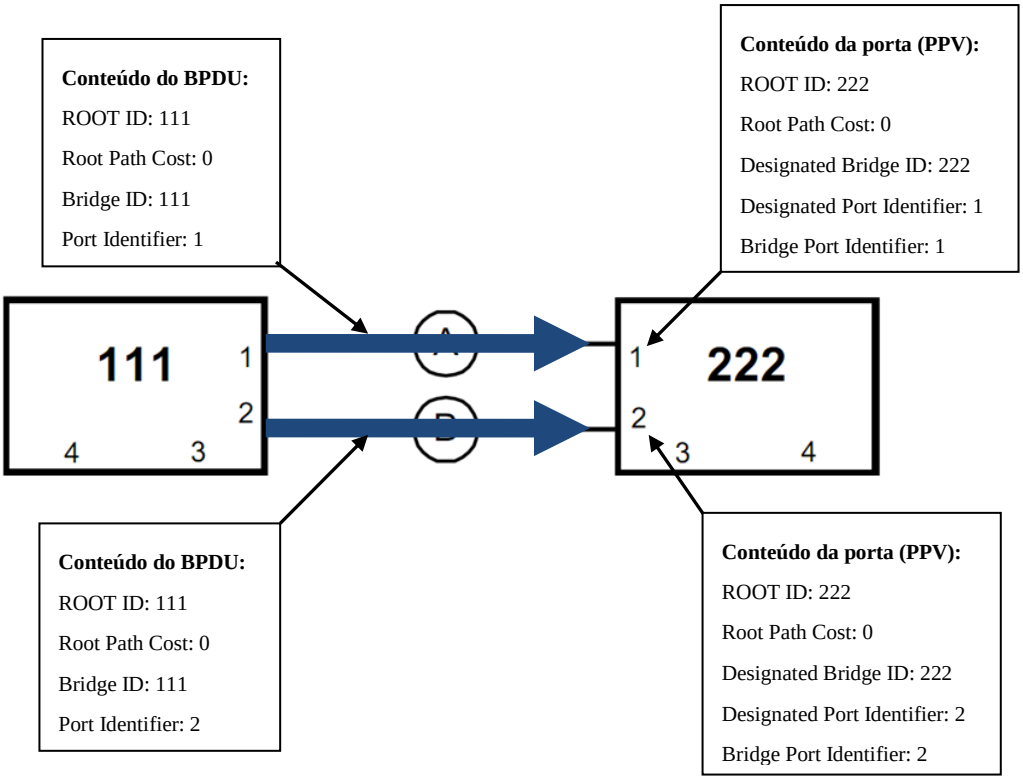


Figura 18 – *Bridge* 111 enviando suas informações.

A *bridge* 222, tendo recebido BPDUs melhores (com valores numéricos menores) do que a informação guardada nos seus PPVs realiza a atualização de suas informações. Caso suas outras portas também estejam conectadas em outras RSTP BRIDGES, a *bridge* 222 envia por essas portas as novas configurações. Esta atualização está ilustrada na Figura 19, destacando as mudanças na *bridge* 222.

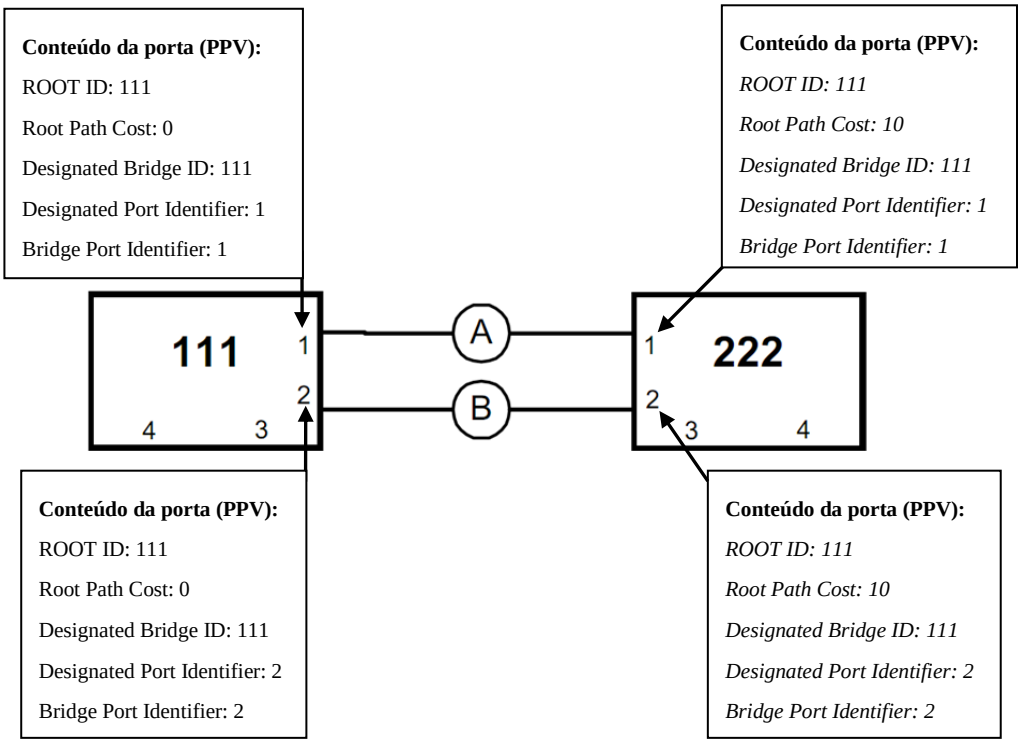


Figura 19 – Atualização do conteúdo da porta (PPV) da *bridge 222.*

Neste momento todas as *bridges* possuem as informações necessárias para que possam calcular os papéis das portas. Primeiro deve-se estabelecer a ROOT PORT. Para isto, a *bridge* verifica todas as suas portas a fim de encontrar a porta que possui o melhor PPV (vetor de prioridade com os valores mais baixos). Feito isto, a porta com melhor resultado é eleita como ROOT PORT, e os dados do PPV desta porta são armazenados no BPV. Caso a *bridge* receba um BPDU com informações melhores que as armazenadas no BPV, elas são armazenadas no BPV, A única *bridge* que não possui ROOT PORT é a *bridge* eleita como ROOT BRIDGE, ou seja, ela não executará este passo. A Figura 20 mostra os dois casos e também os vetores guardados pela *bridge* 222. Pode-se perceber claramente qual PPV tem o melhor valor (menor) simplesmente concatenando as informações contidas em cada vetor de determinada porta de cada *bridge* e comparando-as. Como abaixo, se utilizou das informações dos PPVs das portas da Figura 19 acima para exemplificar:

Exemplo: *Bridge* 111→ Porta1: 111-10-111-1-1 < Porta2: 111-10-111-2-2

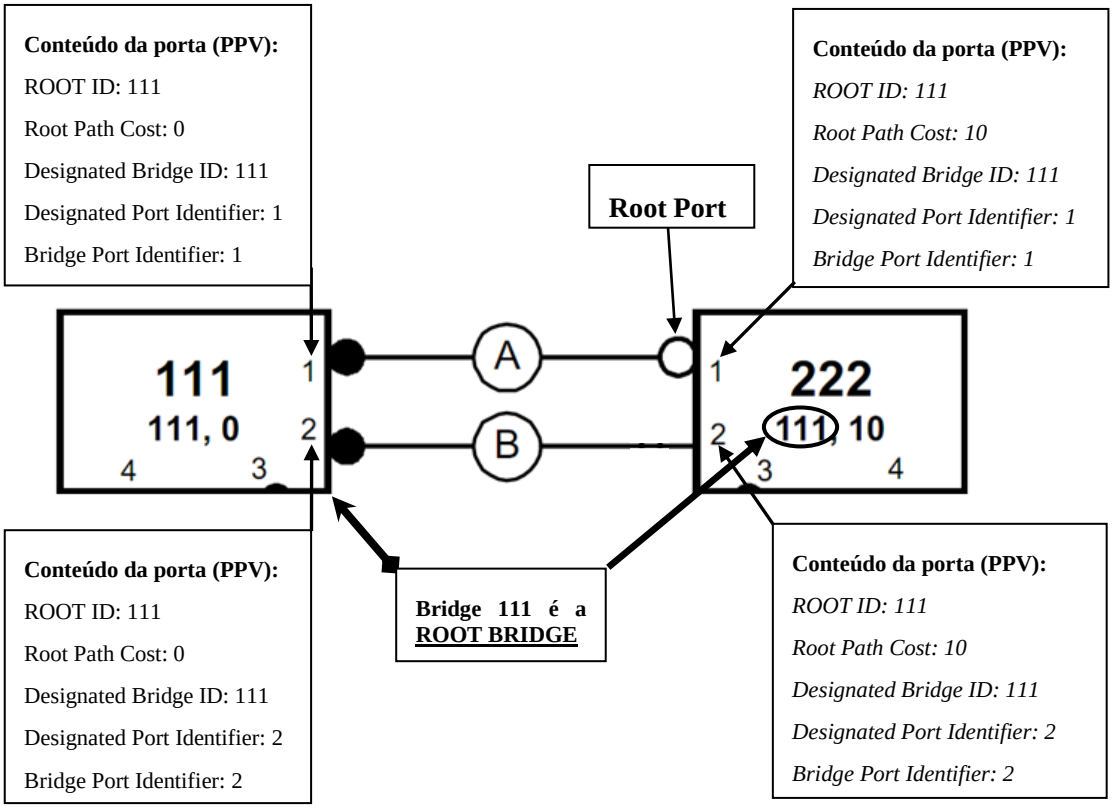


Figura 20 – *Bridge* 222 aceita informações superiores da *bridge* 111.

Tendo sido escolhida a ROOT PORT, deve-se eleger as DESIGNATED PORTS. Uma porta não pode ser *designated* e *root* ao mesmo tempo, portanto todas ROOT PORT estão fora da eleição. Neste momento, cada porta compara a informação do melhor BPDU recebido com as informações que a porta armazena. Se as informações que a porta armazena forem melhores do que os BPDUs que ela recebe, a porta torna-se *designated*. Todas as portas da ROOT BRIDGE conectadas em RSTP *bridges* são DESIGNATED.

A Figura 21 mostra uma topologia onde já ocorreu a eleição da ROOT PORT, e a Figura 22 mostra a eleição das portas DESIGNATED.

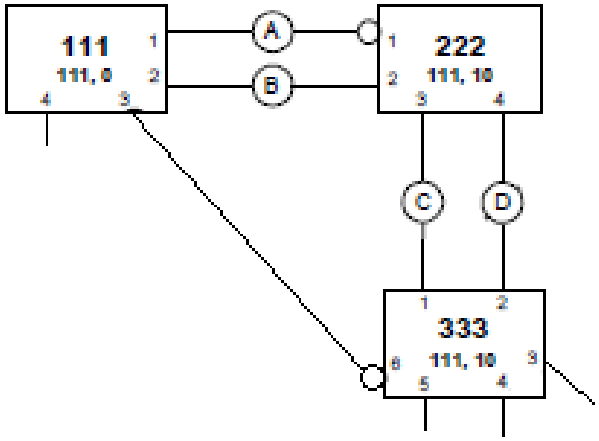


Figura 21 – Eleição realizada das *root ports* (*bridge* 222 porta 1, *bridge* 333 porta 6).

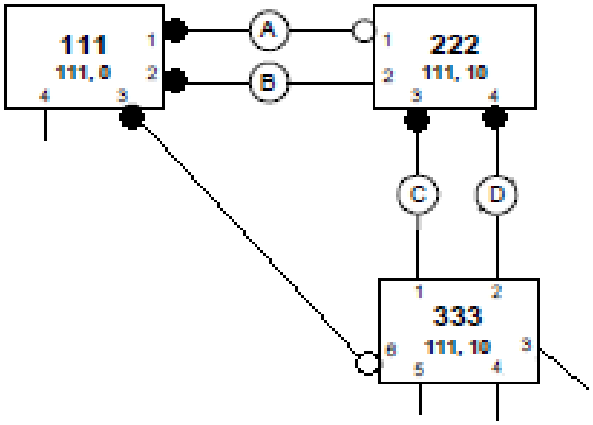


Figura 22 – Eleição realizada das *designated ports* (círculos pretos).

A Tabela 2 apresenta os valores utilizados para a eleição das portas *designated*.

Tabela 2 – Valores enviados e armazenados nas *bridges*.

	Bridge 1			Bridge 2				Bridge 3		
Melhor BPDU Recebido	**p1**	**p2**	**p3**	**p1**	**p2**	**p3**	**p4**	**p1**	**p2**	**p6**
Root Bridge Identifier	111	111	111	111	111	111	111	111	111	111
Root Path Cost	10	10	10	0	0	10	10	10	10	0
Designated Bridge Identifier	222	222	333	111	111	333	333	222	222	111
Designated Port Identifier	1	2	6	1	2	1	2	3	4	3
Bridge Port Identifier	1	2	3	1	2	3	4	1	2	6
BPDU que a porta envia										
Root Bridge Identifier	111	111	111	111	111	111	111	111	111	111
Root Path Cost	0	0	0	10	10	10	10	10	10	10
Bridge Identifier	111	111	111	222	222	222	222	333	333	333
Port Identifier	1	2	3	1	2	3	4	1	2	6

As portas que não são eleitas como *designated*, passam para o estado de *discarding*. Se uma *bridge* possuir duas portas que cumprem os requisitos para ser DESIGNATED de um segmento, a melhor dentre as duas é eleita, e a outra é eleita ALTERNATE.

A Figura 23 mostra a configuração final da topologia.

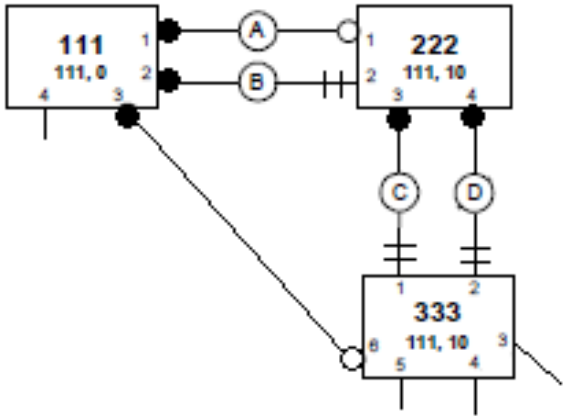


Figura 23 – Estado final da topologia. Portas em *alternate discarding* são marcadas com 2 traços no enlace.

2.4 Seqüencia *Proposal / Agreement*

O sistema *Proposal/Agreement* é utilizado para colocar as portas com os papéis de *root e designated* para o estado *forwarding* o mais rápido possível, através de trocas de BPDUs com suas respectivas *flags* ativadas (Figura 4). Este processo ocorre paralelamente com a convergência da topologia e a eleição da *root bridge* do segmento, resumindo-se em uma espécie de *handshake* entre as *bridges*.

Quando uma porta é selecionada pelo *Spanning Tree* para se tornar uma porta *designated*, o padrão 802.1D ainda espera dois <*forward delay*> segundos (2x15 segundos por padrão) antes de ocorrer a transição para o estado *forwarding*. No RSTP, esta condição corresponde a uma porta com um papel de *designated*, mas em estado *bloqueado*. A Figura 24 [CIS09] ilustra um diagrama passo a passo como a transição rápida é alcançada. Suponha que um novo enlace é criado entre a *root bridge* e a *bridge* A. Ambas as portas neste enlace são postas em um estado *designated blocking* até que elas recebam um BPDU de suas contrapartes.

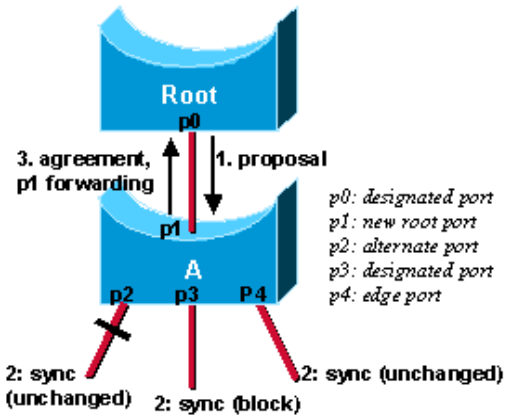


Figura 24 – Exemplificação de *Proposal/Agreement*.

Quando uma porta *designated* está no estado *discarding* ou *learning* (e somente nestes casos), ela seta o *proposal bit* nos BPDUs, que é enviado. Isto é o que ocorre para a porta P0 da *root bridge*, como é mostrado no passo 1 do diagrama da Figura 24. Como a *bridge* A recebe informação superior, ela imediatamente sabe que P1 é a sua nova porta *root*. A *bridge* A então começa uma sincronização para verificar que todas suas portas estão em sincronismo com esta nova informação.

Uma porta está em sincronismo se ela cumprir qualquer um destes critérios:

- A porta está em estado *blocking*, o que significa *discarding* em uma topologia estável.
- A porta é do tipo *edge*.

A fim de ilustrar o efeito do mecanismo de sincronismo em diferentes tipos de portas, supor que exista uma porta P2 *alternate*, uma porta P3 *designated forwarding*, e uma porta P4 *edge* na *bridge* A. Note que P2 e P4 já cumprem um dos critérios. A fim de estar em sincronismo (passo 2 do diagrama da Figura 24), a *bridge* A precisa somente bloquear a porta P3, e atribuir à porta o estado *discarding*. Agora que todas as suas portas estão em sincronismo, a *bridge* A pode desbloquear sua nova porta *root* P1 selecionada, e enviar uma mensagem *agreement* para responder à *root bridge* (passo 3 do diagrama da Figura 24). Esta mensagem é uma cópia do *proposal* BPDU, com o *bit agreement* setado ao invés do bit *proposal*. Isto assegura que a porta P0 sabe exatamente para qual *proposal* o *agreement* recebido corresponde.

Uma vez que P0 recebe o *agreement*, ela pode passar imediatamente para o estado *forwarding*. Este é o passo 4 da Figura 25 [CIS09].

Note que a porta P3 da *bridge* A é deixada no estado *designated discarding* depois do sincronismo, exemplificado na Figura 25, como um caminho bloqueado. No passo 4 (Figura 25), aquela porta (porta P3 da *bridge* A) está exatamente na mesma situação que a porta P0 da *root Bridge* estava no passo P1. Ela então começa a fazer proposta para seu vizinho, fazendo tentativas para passar rapidamente para o estado *forwarding*.

O mecanismo de *proposal/agreement* é muito rápido, uma vez que não depende de qualquer *timer*. Essa onda de *handshakes* propaga rapidamente em direção as extremidades da rede, e rapidamente restaura a conectividade depois de uma mudança na topologia.

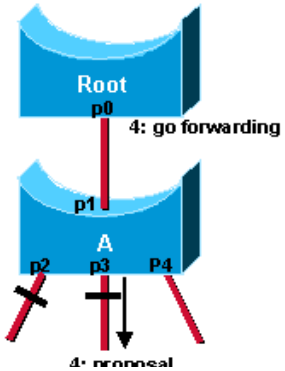


Figura 25 – Exemplificação de *Proposal/Agreement*, passo 4.

Se uma porta *designated discarding* não receber um *agreement* depois de enviar um *proposal*, ela lentamente transita para o estado *forwarding*, retornando para a tradicional seqüência *listening-learning* do modelo STP (ativação do *timer Forward Delay*). Isto pode ocorrer se uma *bridge* remota não entender BPDUs do RSTP, ou se a porta da *bridge* remota está em estado *blocking*.

2.5 *Cisco UplinkFast* anexado ao RSTP

Outra forma de transição imediata para o estado *forwarding* incluída no RSTP é o sistema Cisco *UplinkFast*, extensão proprietária do *spanning tree* proposta pela empresa CISCO. Basicamente, quando uma *bridge* perde sua porta *root*, a *bridge* é capaz de colocar sua melhor porta *alternate* diretamente para o modo *forwarding* (o aspecto de uma nova porta *root* é tratado também pelo RSTP), sem a necessidade de entrar na seqüência *Proposal/Agreement*. A seleção de uma porta *alternate* como nova porta *root* gera uma mudança na topologia. O mecanismo de troca de topologia do modelo RSTP limpa as entradas adequadas nas tabelas de *Content Addressable Memory* (CAM) do *upstream* da *bridge*. Isto remove a necessidade de um falso processo de geração de *multicast* do *UplinkFast*. Este sistema não precisa ser mais configurado por causa da inclusão nativa do mecanismo e ativação automática no RSTP.

2.6 Mecanismo de Mudança de Topologia em STP

Quando uma *bridge* do modelo STP detecta uma mudança de topologia, ela utiliza um mecanismo para notificar primeiramente a *root bridge*. Este sistema é mostrado no diagrama da Figura 26 [CIS09].

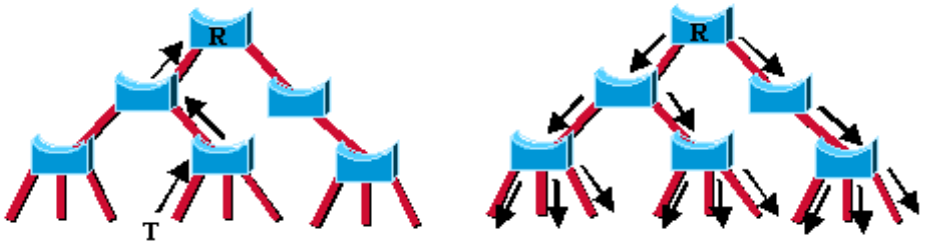


Figura 26 – Mecanismo de mudança de topologia segundo modelo STP.

Uma vez que a *root bridge* está ciente de uma mudança na topologia da rede, ela ativa a *flag TC* nos BPDUs que ela envia, os quais são retransmitidos para todas as *bridges* na rede. Quando uma *bridge* recebe um BPDU com o bit da *flag TC* setado, ela reduz seu *bridging-table aging time* para o tempo de <*forward delay*>, que corresponde a quinze segundos. Isto assegura relativamente um rápido *flush* das informações consideradas obsoletas. Este mecanismo de mudança de topologia é modificado no RSTP.

2.7 Detecção de Mudança de Topologia - RSTP

No RSTP, somente portas *non-edge* que transitam para o estado *forwarding* causam uma mudança na topologia. Isto significa que uma perda de conectividade não é mais considerada como uma mudança na topologia, ao contrário do modelo STP, onde uma porta que transitasse para o estado *blocking* não geraria mais um TC. Quando uma *bridge* RSTP detecta uma mudança na topologia ocorrem os seguintes passos:

- A *bridge* inicializa o *TC while timer* com um valor igual ao dobro do <*hello-time*> para todas suas portas *non-edge designated* e sua porta *root*, se necessário.
- O endereço *MAC* associado a todas as portas é apagado.

Enquanto que o *TC while timer* executar em uma porta, os BPDUs enviados para fora desta porta têm o *TC bit* ativado. BPDUs também são enviados na porta *root* enquanto que o *timer* estiver ativo.

2.8 Propagação de Mudança de Topologia - RSTP

Quando uma *bridge* recebe um BPDU com o *TC bit* ativado, vindo de uma *bridge* vizinha, ocorrem os seguintes passos:

- Ela limpa o endereço *MAC* armazenado em todas as suas portas, exceto aquela que recebe a mudança de topologia.
- Ela começa o *TC While timer* e envia BPDUs com a *flag TC* ativada em todas suas

portas *designated* e porta *root* (RSTP não utiliza mais o específico *TCN BPDU*, a menos que uma *bridge* antiga {*bridge* STP} precise ser notificada).

Nesse sentido, os BPDUs com a *flag* TC se espalham muito rápido através de toda rede. A propagação do TC se resume a um processo de um passo. Na verdade, a *bridge* que gera a mudança de topologia espalha essa informação por toda a rede (exemplificado na Figura 27 [CIS09]), contrário ao modelo STP onde só a *root bridge* faz. Esse mecanismo é muito mais rápido que o modelo STP equivalente. Não é necessário esperar a *root bridge* para ser notificada e então manter o estado de mudança de topologia para toda a rede por <*max age* + *forward delay*> segundos.

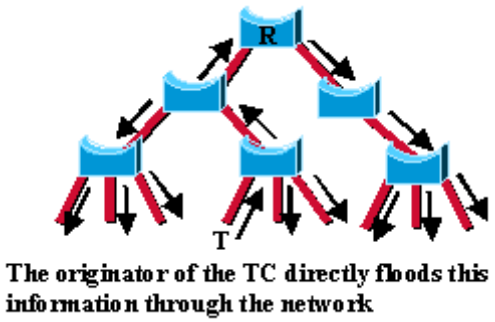


Figura 27 – Exemplificação do *flood* de BPDUs com *flag TC*.

Em apenas poucos segundos, ou em um tempo múltiplo de *hello-times*, a maioria das entradas nas tabelas CAM de toda a rede (VLAN) são apagadas. Essa abordagem resulta potencialmente em mais espalhamentos temporários, mas em contrapartida ela limpa informações potencialmente "antigas" que impedem rápida restituição da conexão.

2.9 Compatibilidade com STP

O RSTP é capaz de interoperar com antigos protocolos STP. Entretanto, é importante ressaltar que os benefícios inerentes de convergência rápida do modelo RSTP são perdidos quando se interage com *bridges* antigas.

Cada porta mantém uma variável que define o protocolo para executar em um dado segmento correspondente. Um *timer* de migração (*Migration Timer)* de três segundos também é ligado quando a porta é ativada. Quando esse *timer* é executado, o modo STP ou RSTP associado à porta é travado. Tão logo que o *delay* de migração expirar, a porta se adapta ao modo que corresponde ao próximo BPDU que ela receber. Se a porta mudar seu modo de operação devido a um BPDU recebido, o *delay timer* de migração recomeça. Isso limita a possibilidade freqüente de troca do modo de operação.

Por exemplo, suponha as *bridges* A e B rodando RSTP, com a *bridge* A em *designated* para o segmento. Uma antiga *bridge* C STP é introduzida neste enlace. Esta situação está exemplificada na Figura 28 [CIS09].

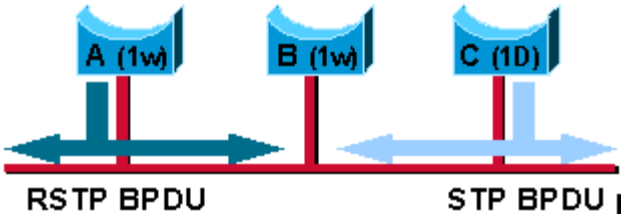


Figura 28 – Interação entre *Bridges* STP e RSTP.

Como as bridges do modelo STP ignoram e descartam os RSTP BPDUs, a *bridge* C acredita que não há outras *bridges* no segmento e começa a enviar seu formato de BPDUs STP inferior. A *bridge* A recebe esses BPDUs e, após dois <*hello-time*> no máximo, troca seu modo de operação para o modelo STP somente naquela porta. Como resultado, a *bridge* C agora entende os BPDUs da *bridge* A e aceita a *bridge* A como uma *bridge designated* para aquele segmento, exemplificado na Figura 29 [CIS09].

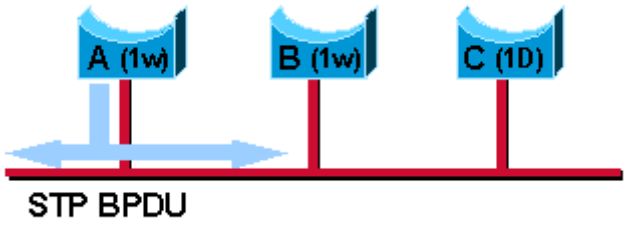


Figura 29 – *Bridge* RSTP operando em STP.

Observe neste caso em particular, se a *bridge* C é removida, a *bridge* A opera no modo STP naquela porta apesar dela estar capacitada a trabalhar mais eficientemente no modo RSTP com seu vizinho, a *bridge* B. Isto é porque a *bridge* A não sabe que a *bridge* C foi removida daquele segmento. Para este caso em particular, uma intervenção de usuário é requerida a fim de reiniciar manualmente a detecção do protocolo da porta.

Quando uma porta está no modo de compatibilidade 802.1D, ela também é capaz de lidar com BPDUs de notificações de mudança de topologia (*Topology Change Notification* – TCN), e BPDUs com as *flags* TC ou TCA ativados.

2.10 *Timers*

O protocolo RSTP se utiliza de alguns *timers* para o funcionamento correto do mesmo, e também para operar com equipamentos anteriores executando o protocolo STP.

Nesta implementação serão abordados os seguintes *timers*:

- *Hello Time*: Este *Timer* controla o envio de BPDUs a cada dois segundos por padrão, sendo administrável.
- *Forward Delay*: *Timer* utilizado para retardar a transição do estado da porta até que outras *bridges* tenham recebido informações *Spanning Tree*. Este *timer* é ativado quando a *bridge*

não recebe resposta, em BPDUs RSTP, referente às informações enviadas. A utilização deste *timer* significa retroceder o protocolo para o modo de compatibilidade com STP, para então passar o estado da porta para *Learning* e depois para *Forwarding*. Isto acontece devido à uma falha no processo do sistema *Proposal/Agreemet*, onde uma *brigde* não recebe resposta depois de enviar um BPDU *Proposal*. O *Timer Forward Delay* é igual à quinze segundos por padrão, sendo administrável.

- *Edge Time*: *Timer* ativado quando a porta é ativada, tendo a duração de três <*hello-time*>. Após este tempo, caso a porta não tenha recebido nenhum BPDU, a porta entra em modo *Edge*. Se durante este tempo, a porta receber algum BPDU, ocorre o *flush* no *timer*, desativando-o, e colocando a porta em modo operacional RSTP, com o estado *Discarding* e papel *Designated*.
- *TC While*: *Timer* ativado quando a *bridge* receber um BPDU com a *flag* TC ativada, ou quando é gerada uma mudança de topologia na própria *bridge*. Então, no intervalo de tempo do *timer*, que corresponde ao dobro do <*hello-time*>, a *bridge* envia BPDUs com a *flag TC* ativada para todas suas portas *non-edge designated* e sua porta r*oot*.
- *Hello When*: *Timer* utilizado em portas com papel *Root* e *Alternate*. O *Timer* compreende um tempo de três vezes o <*hello_time*>. Ele é ativado após o recebimento de um BPDU, e no decorrer deste tempo, caso não receba algum BPDU, a porta transita para o papel *Designated*, mantendo seu estado. Se for uma porta *Alternate*, após a porta transitar para o papel *Designated*, o *Timer Forward Delay* é ativado a fim de colocar a porta em estado *Forwarding*.
- *Migrate*: Este *Timer* é ativado quando a porta da *bridge* é ativada. Ele corresponde a um tempo de três segundos. Quando esse *timer* é executado, o modo STP ou RSTP associado à porta é travado. Tão logo que o *timer* expirar, a porta se adapta ao modo que corresponde ao próximo BPDU que ela receber. Se a porta mudar seu modo de operação devido a um BPDU recebido, o *timer* recomeça.
- *Message Age*: Campo do BPDU com a função de ter um controle dos pacotes que trafegam na rede. O BPDU que parte da *root bridge* carrega este campo com zero. Em cada *bridge* que receber o BPDU, o campo é incrementado, e então, se o campo *Message Age* chegar ao máximo, estipulado pelo *Max Age*, este BPDU é descartado pelo sistema.
- *Max Age:* Campo do BPDU com a função de limitar o número de *bridges* que um pacote BPDU pode alcançar, segundo seu iterador *Message Age*. Este campo tem seus valores administráveis.
- *Bridge Times*: Este *Timer* compreende um conjunto de *Timers*, sendo o *Forward Delay, Hello Time, Max Age* e *Message Age* correspondendo a zero. Estes *Timers* são utilizados onde a *bridge* em questão é a *root Bridge* do segmento, utilizando-se dos próprios tempos configurados.

- *Root Times*: Este *Timer* compreende um conjunto de *Timers*, sendo o *Forward Delay, Hello Time, Max Age* e *Message Age*, derivados das informações (BPDUs) recebidas pela porta *Root*, caso a *bridge* em questão corresponda a uma *Designated Bridge,* ou seja, uma *bridge* diferente de *root bridge*. Assim, pela porta *Root*, ela recebe as configurações estipuladas pela *root bridge*, e se utiliza destas configurações para suas operações, as quais devem ser atendidas por todas as demais *bridges* do segmento.

3. IMPLEMENTAÇÃO DO ALGORITMO RSTP

A implementação do *kernel* do algoritmo RSTP é apresentada neste Capítulo, abordando seus principais passos de execução e características. O *kernel* do algoritmo é utilizado tanto no *Simulador de Conexões*, quanto na plataforma de desenvolvimento. Esta abstração desta parte do projeto é apresentada em destaque na Figura 30.

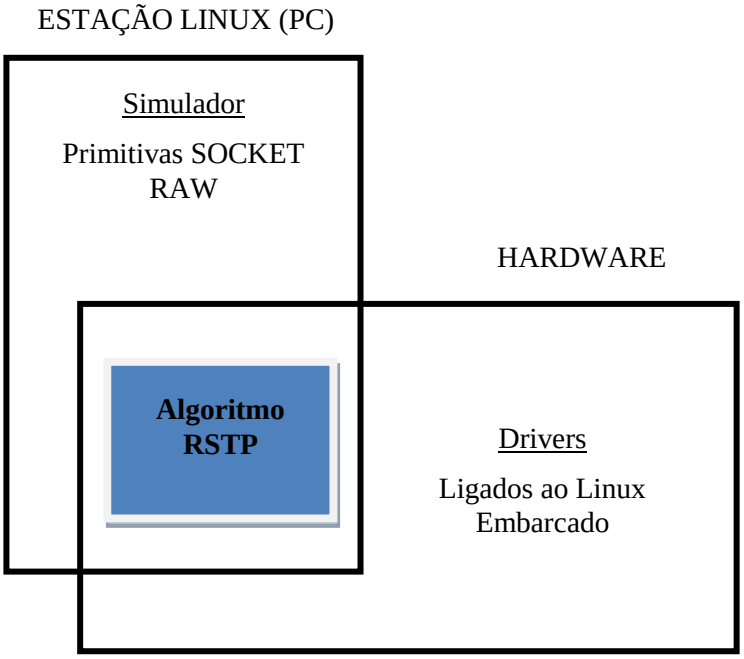


Figura 30 – Representação da implementação do algoritmo RSTP.

O algoritmo RSTP compreende vários passos, onde a maioria deles são executados em paralelo através de *threads*. Primeiramente, ao executar o algoritmo, o sistema sofre uma configuração inicial para executar seus passos RSTP propriamente ditos, chamando a função INICIALIZABRIDGE. Após esta configuração, diversas *threads* de controle do RSTP são executadas em paralelo.

As *threads* executadas pelo algoritmo totalizam cinco. A *thread* SEND, responsável por enviar pacotes BPDUs pelas portas da *bridge*; a *thread* RECEIVE, responsável por receber os pacotes BPDUs, avaliá-los e com as informações obtidas, configurar as portas da *bridge* e seu sistema; a *thread* TIME_LINKSTATE_POLLING, que atua no monitoramento dos enlaces das portas da *bridge*; a *thread* TIME, responsável por coordenar os *timers* da *bridge*; a *thread* ADMIN, que possibilita ao administrador de rede alterar configurações da *bridge* em tempo de execução. Este processo é exemplificado na Figura 31, onde os processos internos das *threads* serão explicadas no decorrer do Capítulo.

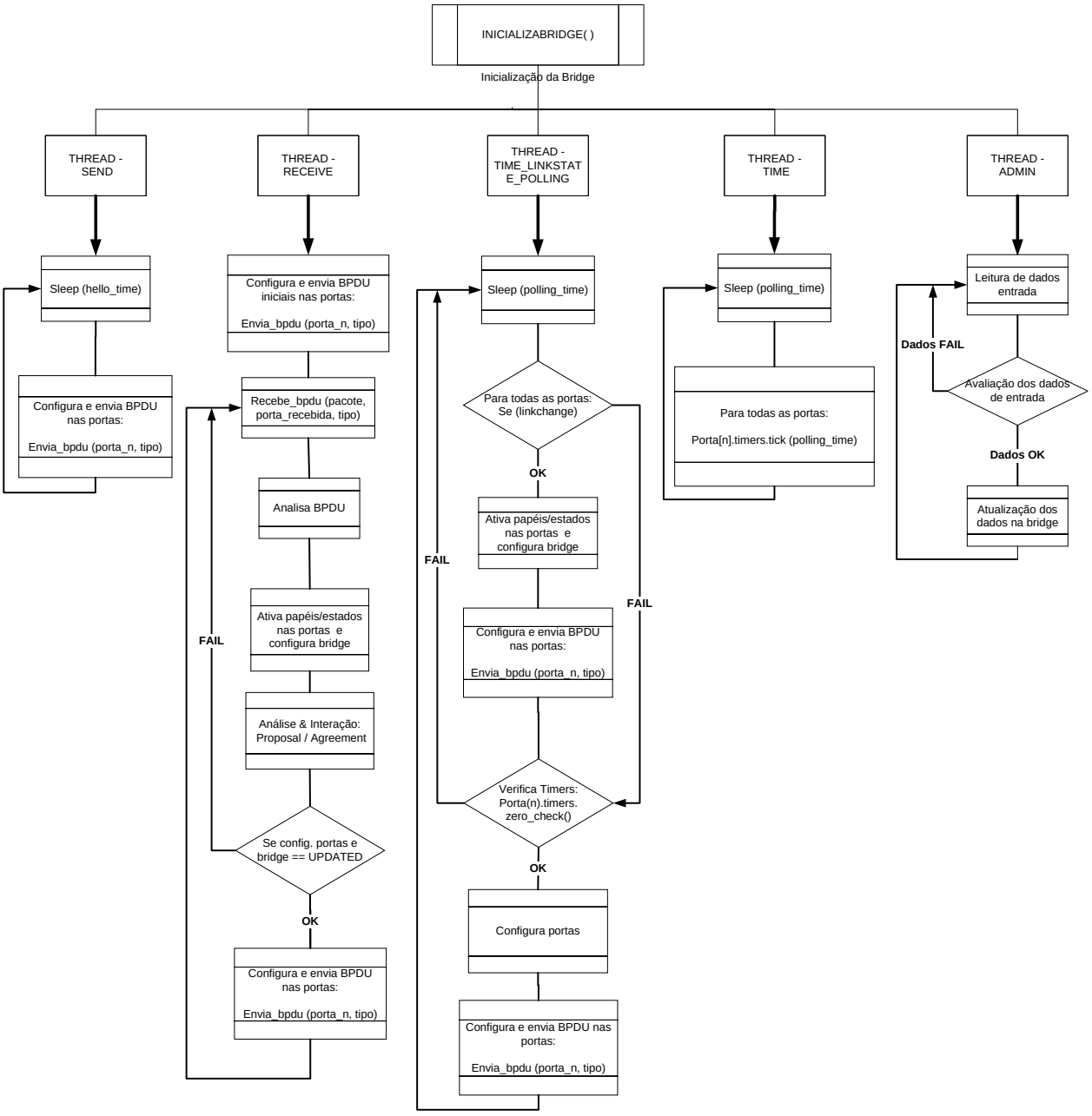


Figura 31 – Diagrama geral do algoritmo.

3.1 Inicialização da *Bridge*

Na inicialização da *bridge*, vários procedimentos são realizados para o seu correto funcionamento. Primeiramente, são criadas as variáveis de envio e recebimento de pacotes através de *socket raws*, tanto para o simulador quanto para o *hardware*.

Na seqüência, a função INICIALIZABRIDGE é executada para realizar a leitura das configurações iniciais da *bridge*. Dentro desta função são chamadas outras funções auxiliares que configuram parâmetros específicos da *bridge*. Elas são chamadas nas respectivas ordens:

1) SWITCHINIT é chamada para inicializar o *switch* do *hardware*, configurando-o.

2) A função MACHWRECOVERY é chamada para a configuração do *mac* da *bridge*, através de biblioteca que faz a interação com o *hardware.* Para a versão de simulação, a criação de uma classe chamada PARSER, realiza a operação de leitura do arquivo de entrada para o simulador, configurando o *mac* da *bridge* e criando a matriz de conexões.

3) É configurado o ID da *bridge* através da função SETBRIDGEID.

4) A função CONFIGURAPORTAS se encarrega de configurar os estados e papéis inicias das portas (*discarding – designated)*, os custos de seus enlaces e realiza a ativação dos *timers* das mesmas. A configuração do algoritmo para o *hardware* é feito através da biblioteca de integração do mesmo. Para a configuração do simulador é feito a leitura do arquivo de entrada através da classe PARSER citada anteriormente.

Após estas funções é feita a configuração dos parâmetros da própria *bridge*. Inicialmente, a *bridge* é configurada como sendo a *root bridge* do segmento. É configurada a identificação da *root bridge*, sendo a própria *bridge; root path cost* é configurado como zero; *message age* é configurado como zero; *max age*, *hello time* e *forward delay* são configurados inicialmente com valores padrões; os *timers* que a *bridge* inicialmente configura derivam do conjunto *bridge_times*; é ativado o modo *root bridge*.

Em seguida, as *threads* TIME_LINKSTATE_POLLING, RECEIVE, SEND, TIME e ADMIN são iniciadas para o funcionamento do algoritmo propriamente dito.

3.2 Transmissão de Pacotes

O envio de pacotes BPDUs é feito em vários momentos dentro do algoritmo. Para o sistema de *hellos* periódicos que o RSTP utiliza, é criado a *thread* SEND (Transmissão de Pacotes). Esta *thread* SEND tem a finalidade de enviar continuamente BPDUs (do tipo *is_hello,* explicado em seguida) com um intervalo estipulado pelo parâmetro administrável <*hello-time*>.

O processo de envio ocorre executando a função ENVIA_BPDU. Ela tem como parâmetros o número da porta e o tipo de envio, que pode variar de *is_hello*, *not_hello* e *tcn_bpdu*. O tipo de envio *is_hello* utiliza as configurações atuais que estão armazenadas nos PPVs das portas, enviando BPDUs por todas as portas com papel *designated* ativado. O tipo *not_hello* identifica BPDU contendo informações recentes, que foram configuradas por algum processo e necessitam do envio destas informações imediatamente, como por exemplo, uma alteração na topologia. O tipo *tcn_bpdu* tem a finalidade de avisar *bridges* STP da mudança de topologia até que a mensagem atinja a *root bridge* STP, para que esta avise outras *bridges* da mudança, por todas suas portas. Caso ocorra uma mudança de topologia (alguma porta da *bridge* recebeu BPDU com TC habilitado), e a *bridge* possuir uma porta *root* operando em modo STP, é enviado nesta porta, após o BPDU de *hello*, um BPDU TCN (Topology Change Notification – Notificação de mudança de topologia). O BPDU TCN tem uma configuração própria, apresentando apenas os campos *Protocol Identifier* (recebe valor zero – campo de 8 *bits*), *Protocol Version Identifier* (recebe valor zero – campo de 8 *bits*) e *BPDU Type*

(recebe valor binário 1000 0000 – campo de 8 *bits*).

A função ENVIA_BPDU configura o pacote BPDU para ser enviado pelas portas. Dentro desta função existem duas funções auxiliares (CONFIGBPDUV2 e CONFIGBPDUV0) que configuram o BPDU de acordo com a versão *spanning tree* em operação na porta. Existem duas versões, a versão zero (STP) e a versão dois (RSTP). A versão zero existe para compatibilidade com *bridges* STP, e possui quatro campos com dados diferentes da versão dois - RSTP. Os campos *Protocol Version Identifier* e *BPDU Type* recebem o valor zero, e o campo *Version 1 Length* deixa de existir. No campo de *flags* de oito *bits,* a versão zero leva em consideração apenas o primeiro *bit* (TC – Topology Change) e o último *bit* (TCA – Topology Change Acknowledgment) do octeto, ignorando os outros seis *bits* intermediários. Os demais campos são configurados da mesma maneira para as duas versões, obtendo informações dos PPVs das portas e das configurações da *bridge*.

O campo *flags* é configurado de acordo com as *flags* armazenadas em cada porta referente à sua versão *spanning tree*. Na versão dois, o *TCWhile timer* é verificado. Se estiver ativo ou vai ser ativado, é inserido a *flag TC* no campo de *flags* do BPDU.

Caso uma porta tenha recebido um BPDU TCN, e a *bridge* em questão for *root bridge* do segmento ou a porta *root* estiver operando em RSTP, é ativado a *flag TC* (execução da função SET_TC_FLAG_RSTP na porta) no BPDU para envio pela porta. Se a porta *root* estiver operando em modo STP, é enviado um BPDU TCN pela porta *root*, independente do BPDU que vai ser enviado na seqüência requisitado primeiramente pela função ENVIA_BPDU.

A função SET_TC_FLAG_RSTP é executada em portas com papel *root* ou *designated*. Se a porta que está executando a função não estiver com o *TcWhile timer* ativado, é realizado em todas as portas o *flush* da tabela de MAC e do tempo transcorrido anteriormente pelo *TcWhile timer*, ativando o mesmo. Então os BPDUs serão enviados em todas as portas *designated* e *root*, configurados com a *flag TC* ativada de acordo com o *TcWhile timer*.

Após a configuração, é feito a montagem do pacote para ser enviado. Existem tamanhos de pacotes diferentes para os três tipos de BPDUs, devido à estrutura de cada um.

Para o *hardware*, é anexado ao pacote o BPDU que será enviado, e no início do pacote, incluído o *header ethernet* juntamente com a *SpecialTag*, necessária para a plataforma de desenvolvimento identificar as portas do *switch*. Com o pacote completo, o mesmo é enviado para a interface de rede através de *socket raw*.

Para o simulador, é anexado ao pacote o BPDU que será enviado, e no início do pacote, incluído o sistema de controle para o simulador. Esta configuração do controle consiste em incluir o identificador de BPDU (666), o "canal" atribuído a porta e a direção do pacote, neste caso *bridge_out*. Com o pacote completo, é feito o envio do mesmo através de *socket raw* para a interface de rede *l0*, aonde o simulador recebera este pacote e encaminhará ao seu destino.

A função ENVIA_BPDU realiza um controle administrável do número de pacotes enviados

por porta. O número de BPDUs de configuração, diferentes de BPDUs *is_hello*, é limitado a seis (padrão) envios por segundo em cada porta. Caso a porta tenha excedido esse número, a operação de envio é interrompida por um segundo e os BPDUs que seriam enviados nesta porta são armazenados em fila. Após o termino do tempo, os pacotes armazenados são enviados na porta conforme a fila.

3.3 Recepção de Pacotes

A maior complexidade do algoritmo se encontra na *thread* RECEIVE. Esta *thread* realiza, através da recepção de pacotes BPDUs, a maioria das configurações que a *bridge* recebe, além de ativar seus sistemas auxiliares para a convergência da topologia, como os *timers* e a seqüência *Proposal* / *Agreement*.

Ao iniciar esta *thread,* é feito o envio primário das configurações da *bridge* e de suas portas (neste momento a *bridge* está configurada como *root bridge*), através da função ENVIA_BPDU. É realizado o envio de BPDUs *not_hello* em todas as portas ativas, as quais inicialmente são todas configuradas com o papel *designated* e em estado *discarding*. A *flag Proposal* é ativada no BPDU a fim de estabelecer o inicio da seqüência de *Proposal* / *Agreement*.

Na seqüência, a *thread* entra em um laço infinito para receber pacotes e configurar a *bridge*. A função RECEBE_BPDU é executada a fim de receber o BPDU enviado à *bridge*. Esta função utiliza SOCKET RAW para efetuar a recepção do BPDU, a qual fica em laço infinito até que receba um pacote BPDU. Após o recebimento do BPDU, é feito a verificação do mesmo, analisando o tamanho do pacote para confirmar sua integridade. Este BPDU recebido é armazenado em uma variável temporária para realizar a análise de seus campos.

Para o simulador, o "canal" do BPDU (caracteres de controle contidos no *header* adicional do BPDU) é verificado com os "canais" associados às portas da *bridge*. Se o canal do BPDU for igual ao canal da porta da *bridge*, então o pacote é para a *bridge*, e ao mesmo tempo, se obtém a identificação da porta que recebeu o pacote.

Em seguida, é verificado o *status* (ativada ou desativada) da porta que recebeu o BPDU, e se o pacote contém a *flag proposal* ativada. Este procedimento é realizado pois o BPDU contendo a *flag proposal* é enviado no momento em que se ativa o enlace, ou seja, se conecta o cabo entre as *bridges*. Então, caso uma das *bridge* tenha um sistema de varredura de portas mais rápido que o algoritmo implementado, ela irá enviar o BPDU com a *flag proposal*, e o algoritmo implementado irá perder este pacote devido a sua porta estar ainda desativada. Assim, caso a porta ainda desativada receba o BPDU com a *flag proposal*, ela é ativada imediatamente através da função ACTIVE_PORT. Se o BPDU recebido não estiver com a *flag proposal* ativado, a porta descarta o pacote e a sua ativação será de acordo com a *thread* TIME_LINKSTATE_POLLING.

O BPDU é verificado segundo seu valor de *MaxAge*. Caso tenha passado o limite estipulado pela *bridge* (*bridge times)*, o pacote é descartado e a porta recebe configuração com papel *designated*.

Se o BPDU recebido for do tipo BPDU TCN, a porta receptora recebe configurações para enviar BPDUs com a *flag TC Ack* ativada (*flag* correspondente ao modo STP).

Posteriormente, é feito o controle de portas *edge*. Caso a porta esteja configurada como *edge* e receba um BPDU, é realizada a desativação do *timer Edge Time* para a porta, e recebe uma configuração com papel *designated*, estado *discarding* além da desativação do modo *edge*. Caso a porta esteja configurada em modo RSTP, ao receber um BPDU, apenas o *timer Edge Time* é desativado.

O *root path cost* (RPC) recebido pelo BPDU é acrescido pelo valor do custo da porta, para comparação com o RPC armazenado nos PPVs das portas. Caso o BPDU tenha sido enviado da própria *bridge*, caso característico de uma conexão com um *hub* ou *self loop*, é somado ao RPC o custo da porta que enviou o BPDU.

Se o BPDU recebido for da versão zero e a porta estiver operando em RSTP, a porta é configurada para STP, realizado a ativação do *timer Migrate*, além da configuração do *timer TC While*. Se o BPDU versão zero tiver a *flag TC Ack* e a porta receptora estiver em modo STP, a *flag TC* é removida dos BPDUs enviados. Caso o BPDU recebido for da versão dois e a porta estiver operando em STP, a porta é configurada para RSTP, realizado a ativação do *timer Migrate* e configurado o *timer TC While*.

Após verificação do *timer Migrate*, os BPDUs recebidos contendo versões diferentes das da porta são descartados.

Neste ponto, é comparado o BPDU recebido com o PPV da porta receptora. Esta análise abre três configurações possíveis proveniente da diferença de dados:

- Caso o *root path cost* tenha mudado, realizada a atualização dos dados do PPV da porta receptora pelos novos dados do BPDU. O *timer Forward Delay* é reiniciado. Se a porta tiver papel *root*, o BPV recebe os valores atuais do PPV da porta receptora do BPDU. Todas as portas fazem a reconfiguração do *root path cost* a partir dos valores armazenados no PPV da porta receptora, somando o custo do enlace de cada porta (inclusive portas com papel *alternate*). Se a porta estiver operando em modo STP, é enviado um BPDU TCN.
- Caso a *root bridge ID* tenha mudado, e a *bridge* não possua uma porta *alternate* com informações melhores do que as contidas no BPDU recebido:
    - Se a porta receptora do BPDU estiver configurada com papel *root*, é realizada a comparação do PPV com o BPDU recebido, atualizando as informações do PPV, caso o BPDU contenha informações melhores. O *timer Forward Delay* é reiniciado. O BPV é configurado com os valores do PPV da porta receptora (ocorre envio de BPDU TCN caso a porta opere em STP); caso a *bridge* se torne *root bridge*, o conjunto de *timers Bridge Times* é ativado e o BPV é atualizado com o modo inicial de *root bridge*. Ocorre a configuração dos PPVs das demais

portas da *bridge* a partir das novas informações adquiridas.

- Se a porta receptora do BPDU estiver configurada com papel *alternate*, é realizada a atualização do PPV das portas conectadas a mesma *bridge*, e configuradas com papel *designated* e estado *discarding*.

- Caso a *root bridge ID* tenha mudado, e a *bridge* possua uma porta *alternate* com informações melhores do que as contidas no BPDU recebido:
  - Se a porta receptora do BPDU estiver com papel *root*, *ou* estiver com papel *alternate* e o BPV tiver a mesma *designated bridge ID* do PPV da porta receptora, a melhor porta *alternate* é eleita como porta *root* e o BPV é atualizado com as informações do seu PPV. A nova porta *root* realiza a transição para o estado *forwarding*, ativando a *flag TC* nos BPDUs enviados. Caso a porta esteja operando em STP, BPDUs TCN são enviados na porta. As demais portas são reconfiguradas com estado *discarding* e papel *designated* e com os novos dados provenientes da nova porta *root*,
  - Se a porta receptora do BPDU estiver com papel *alternate*, as portas conectadas a esta mesma *bridge* são reconfiguradas com os dados do BPV, estado *discarding* e papel *designated*.

Após a configuração do PPV das portas, o melhor PPV é armazenado no BPV. A partir destas informações, se define a *root bridge*. É comparada a identificação da *bridge* com a identificação da *root bridge* (proveniente das informações gravadas no BPV). Caso a *bridge* seja *root bridge*, o *root path cost* é configurado com zero, seu BPV é reconfigurado com informações iniciais de estado de *root bridge* e o conjunto de *timers Bridge Times* é ativado. Caso a *bridge* seja *designated bridge*, os PPVs das portas (menos a porta receptora do BPDU) são reconfigurados com os dados do BPV (*root bridge ID* e RPC mais enlace da porta) e atualizadas com o papel *designated*.

Depois das configurações de todos os vetores de dados anexados ao sistema, é realizada a configuração dos papeis das portas. Os papéis são analisados e atribuídos na ordem abaixo, sendo crucial para a configuração das portas:

- *Backup*: Se o campo *designated bridge ID* do PPV da porta for igual à *bridge identifier*, se o campo *root bridge ID* do PPV for igual à *root bridge* (BPV) e o campo *designated port ID* for diferente do campo *bridge port ID* do PPV, a porta é configurada com papel *backup* e estado *discarding*.
- *Root*: Se a identificação da porta for a mesma contida no campo *bridge Port ID* do BPV, a porta é configurada com papel *root* e estado *discarding*. Um BPDU recebido em uma porta, sendo melhor do que todos os outros faz com que o PPV da porta seja eleito como BPV. Assim a porta que recebe o melhor BPDU de todas as portas é eleita a porta *root* da *bridge*.
- *Designated*: Caso a *bridge* seja do tipo *root bridge,* suas portas, exceto as portas *backup* e

portas em modo *edge*, são configuradas com papel *designated* e estado *discarding*. Caso a *bridge* seja *designated bridge*, se o campo *designated bridge ID* do PPV da porta for igual à *bridge identifier*, a porta é configurada com o papel *designated* e com estado *discarding.*

- *Alternate*: Se a comparação de igualdade anterior (*Designated*) entre o campo *designated bridge ID* do PPV da porta e o *bridge identifier* não se confirmar, a porta será configurada com papel *alternate* e estado *discarding*.

Após a configuração das portas, se tiver sido configurado uma porta com estado *root* (significa que a *bridge* é *designated bridge*), o conjunto de *timers Root Times* é configurado e ativado, conforme BPDU recebido.

O processo de configuração do estado das portas tem inicio neste ponto, onde a seqüência *Proposal / Agreement* é executada. Caso a porta receptora do BPDU não esteja em estado *forwarding* e estiver configurada com papel *root*, é executada a função PROPOSED. Esta função analisa as *flags* do BPDU recebido. Se o BPDU estiver com as *flags designated* e *proposal*, ou *designated, forwarding* e *learning*, a função retorna verdadeiro. Com o retorno positivo da função, é executada a função SET_SYNC_TREE. Esta função realiza o sincronismo em todas as portas (exceto portas em modo *edge* e desativadas), configurando seus estados para *discarding*. Então, se todas portas estiverem sincronizadas e a porta receptora estiver configurada com papel *root* e ainda não está em *forwarding*, é realizado o envio de BPDU *not_hello* (ENVIA_BPDU) na porta receptora. Após o envio, a porta é configurada para o estado *forwarding*, e ativado a *flag TC* nos BPDUs que serão enviados.

Se a porta receptora estiver configurada com papel *backup* ou *alternate*, é realizado o envio de BPDUs *not_hello* (ENVIA_BPDU) na porta, com a *flag agreement* ativada. A porta receptora não irá entrar em *forwarding* (devido aos papéis configurados), mas a porta da outra *bridge* que está conectada a porta receptora será configurada com estado *forwarding*, pois seu papel é *designated*.

Se a porta receptora estiver conectada a um *hub*, configurada com papel *designated* e ainda não está em *forwarding*, é executada a função AGREED. Esta função analisa as *flags* contidas no BPDU recebido. Se o BPDU estiver com as *flags agreement* e *root* ou *forwarding*, *learning* e *root*, é realizada a configuração da porta para o estado *forwarding*, e ativado a *flag TC* nos BPDUs enviados.

Caso a informação recebida pelo BPDU seja pior que as informações armazenadas no PPV da porta, é enviado um BPDU com as informações melhores da porta, ignorando as recebidas. Se o BPDU contiver a *flag TC* ativa, é ativada na porta a mesma *flag* para que os BPDUs sejam enviados contendo esta *flag*.

Ocorrendo alterações nas configurações da *bridge,* é realizado o envio em todas as portas com papel *designated*, de BPDUs is_hello com as novas informações configuradas na *bridge*. As telas de interface sobre as alterações ocorridas são geradas neste ponto (*Interface de Funcionamento*)

3.4 Avaliação do Estado Físico das Portas

O controle do algoritmo se baseia totalmente na configuração das portas e também em seu estado físico. Com essa necessidade, é implementado uma *thread* que controla o estado físico das portas, configurando as mesmas caso elas sejam desativadas ou ativadas.

As portas do *switch* do *hardware*, como as portas virtuais no caso do simulador, são verificadas continuamente de acordo com o *Polling_time* configurado para um segundo.

Inicialmente, a *thread* verifica alguma alteração do *status* das portas (ativada ou desativada). Caso haja alteração, a *flag link_change* é ativada (*flag* de controle do sistema da *thread*). Para o simulador, as configurações das portas são relidas do arquivo de entrada, *par.txt*.

A *flag link_change* ativa a parte central do algoritmo de mudança de estados físicos das portas. Para as portas que tiveram o *status* alterado, abre-se quatro possíveis configurações referente ao papel que estas portas estão ou estavam exercendo:

- *Is Designated Port*: Caso a queda do enlace (ou a porta seja desativada) ocorra em uma porta com o papel *designated*, é verificado se a *bridge* possui alguma porta com papel *backup* relacionado a esta porta *designated*. Caso exista, a porta com papel *backup* assume o papel de *designated* e transita para o estado *forwarding*, executa a função SET_TC_FLAG_RTSP (explicado na seção 3.2 deste Capítulo) e ainda atualizando as informações antigas armazenadas no seu PPV. A porta *designated* na qual ocorreu a queda do enlace é configurada com estado *discarding*. Em uma situação onde existam várias portas com papel *backup*, simbolizando uma conexão com um *hub*, a porta com melhor configuração (melhores valores armazenados) assumirá o papel de *designated*.
- *Is Root Port*: Caso a queda do enlace (ou a porta seja desativada) ocorra em uma porta com o papel *root*, é verificado se a *bridge* possui alguma porta com papel *alternate*:
    - Caso existam várias, é selecionada a melhor porta *alternate* (com melhores valores em seu PPV). Esta porta *alternate* selecionada recebe o papel de porta *root*. Caso esta porta esteja operando em STP, é enviado no mesmo instante um BPDU TCN na porta. É realizada uma reconfiguração do root path *cost* do PPV das portas. É armazenando o *root path cost* da nova porta *root* mais o custo do próprio enlace de cada porta em todas as portas diferentes de portas *alternate*. Todas as portas *non-edge designated* são configuradas para o estado *discarding*. A porta *root* na qual ocorreu a queda do enlace é configurada com o estado *discarding*, e a nova porta *root* transita para o estado *forwarding*. É executada a função SET_TC_FLAG_RSTP. O BPV (Brdge Priority Vector) é atualizado com os novos dados do PPV da porta *root* atual. Em seguida, é enviado BPDUs *not_hello* em todas as portas *designated* com a *flag Proposal* ativada.
    - Caso não existam portas *alternate*, é realizado um *flush* em todas configurações da *bridge*, através da função REROOT. Esta função faz com que os PPVs das

portas e o BPV da *bridge* retornem ao seu estado inicial, ou seja, com as configurações padrão do momento da inicialização da *bridge*. Assim a *bridge* volta a estar configurada como *root bridge,* possuindo todas suas portas em papel *designated* e estado *discarding*, além de seus *timers* zerados. Na seqüência são enviados BPDUs *not_hello* com configurações de *root bridge* em todas as portas ativas. O BPDU enviado possui a *flag Proposal* ativada.

- *Is Unknown*: Se a porta está sendo ativada, é executada a função ACTIVEPORT. Esta função realiza a configuração primária do PPV da porta, obtendo a informação do ID da *root bridge* atual e do *root path cost* do BPV da *bridge*. O custo do enlace da porta é somado ao *root path cost* obtido e armazenado em seu PPV. A porta é configurada com o papel *designated* e estado *discarding*. Os *timers* da porta, *Edge Timer* e *Migrate Timer*, são zerados e ativados. Na seqüência são enviados BPDUs *not_hello* em todas as portas ativas.
- Se a porta não estiver mais ativa (desligada), a porta é configurada como *Unknown* no campo papel e com estado *discarding*. Seus *timers* são desativados e suas informações armazenadas no PPV são apagadas, retornando aos valores padrão iniciais.

Após verificação dos enlaces em cada porta, é feita a verificação dos *timers* das portas. Esta verificação é feita através da função ZERO_CHECK, executada em todas as portas. Esta função analisa todo o conjunto de *timers* que pertence à porta, retornando verdadeiro caso algum *timer* tenha chegado ao seu tempo de ativação (*timer* chegou à zero). Caso algum *timer* tenha chegado ao seu tempo de ativação, é feito os seguintes procedimentos:

- *Timer - Edge Time*: A porta é configurada como e*dge*, recebe o papel *designated*, transita para o estado *forwarding* e desativa o *timer Edge Time* para a porta.
- *Timer - Forward Delay*: Caso a porta tenha o papel *root* ou *designated*:
    - Se a porta estiver com o estado *discarding*, é reconfigurada com o estado *learning* e o *timer Forward Delay* é recomeçado.
    - Se a porta estiver com o estado *learning*, é reconfigurada com o estado *forwarding* e o *timer Forward Delay* é desativado. Caso a porta esteja operando em modo RSTP, é executada a função SET_TC_FLAG_RSTP na porta.

  Caso a porta tenha o papel *alternate* ou *backup*, o *timer Forward Delay* é desativado.
- *Timer - Hello When*: Caso a porta tenha o papel *alternate* ou *backup*, o seu PPV é atualizado com informações atuais armazenadas na *bridge* (BPV), *root path cost* é atualizado, e a porta é configurada com papel *designated*. Caso a porta tenha o papel *root,* é realizada a configuração do BPV para o estado padrão inicial e executada a função REROOT. Após a configuração é enviado BPDUs *not_hello* em todas as portas ativas operando com papeis *spanning tree* (ignorando portas *edge*). Caso a porta tenha

o papel *designated*, o *timer Hello When* é desativado nesta porta.

- *Timer - Tc While*: O *timer* é desativado, restaurando seu contador ao ponto inicial.
- *Timer – Migrate:* O *timer* é desativado.

Após as operações com *timers*, são geradas as telas de interface para o usuário das modificações ocorridas no sistema pelo algoritmo (referente às modificações geradas pelos *timers)*, além de exibir o estado atual em que o sistema se encontra (*Interface de Funcionamento*).

3.5 Temporizadores

O algoritmo RSTP opera com base em alguns *timers* explicados no Capítulo 2.10. Para isso, é necessário que se tenha um controle do tempo para que estes *timers* funcionem de forma paralela e contínua. Sendo assim, a *thread Time* é criada para controlar os contadores dos *timers* de todas as portas.

A cada um segundo, tempo especificado para executar o *polling* de leitura (*Polling_Time),* esta *thread* acessa todas as portas da *bridge* e decrementa os seus contadores com base no tempo de *polling*.

A verificação de que algum dos *timers* de uma porta da *bridge* chegou ao seu tempo é feita na *thread Time LinkState Polling*, mencionado anteriormente.

3.6 Administração

O algoritmo RSTP tem alguns parâmetros administráveis pelo usuário, com a possibilidade de alteração em tempo de execução. Devido a esta necessidade é implementado uma *thread* específica para administração.

O funcionamento desta *thread* de administração se resume em receber continuamente comandos contendo o nome do parâmetro a ser modificado e o seu valor. Antes da informação ser alterada na *bridge*, a mesma passa por um sistema de confirmação de dados, que compõem a biblioteca CHECKVALUES utilizada.

A informação requisitada é alterada instantaneamente na configuração da *bridge*, sem interrupção de funcionamento.

Os parâmetros administráveis são os seguintes:

- *Hello Time*, *Max Age*, *Forward Delay* → Tempos administráveis que seguem uma fórmula [INS04] de coerência de dados. Estes dados têm de ser válidos para que a *bridge* receba os novos valores. A fórmula de avaliação é a seguinte:

$$(\ 2 \times (Bridge\_Forward\_Delay - 1.0\ seconds)\ >= Bridge\_Max\_Age\ )$$
$$\&\&$$
$$(\ Bridge\_Max\_Age\ >= 2 \times (\ Bridge\_Hello\_Time + 1.0\ seconds\ )$$

- *Bridge Priority* → Valor da prioridade que compõem a identificação da *bridge*. Aceita valores de 0 a 61440.
- *Port Priority* → Valor da prioridade da porta que compõe sua identificação. Aceita valores entre 0 a 240.
- *Tx Count* → Número máximo de BPDUs que a *bridge* pode enviar por segundo. Aceita valores entre 1 a 10.
- *Port Cost* → Custo do enlace anexado a porta. Aceita valores entre 1 a 200000000.
- *Migrate* → Ativa o *timer Migrate* em porta desejada.
- *Force* STP → Habilita porta desejada a operar em modo STP.
- *Back* RSTP → Faz com que a porta retorne ao modo de operação RSTP.
- *Disable Port* → Desativa porta desejada.
- *Enable Port* → Ativa porta desejada.
- *Admin Edge* → Habilita porta desejada a trabalhar em modo *edge*.
- *Path Cost Method* → Modifica o sistema de custos da porta. As portas podem utilizar sistema de custo LONG ou SHORT.

3.7 Interface de Funcionamento do Algoritmo

O protocolo RSTP geralmente opera acoplado a um sistema de controle maior e mais complexo de *bridges*, onde realiza a sua parte de tratamento sobre laços lógicos entre *bridges* na rede. Assim, o protocolo não contém uma interface nativa do algoritmo, sendo difícil de analisar e verificar o funcionamento do seu processo. A partir deste detalhe, foi criando um sistema para monitoramento do algoritmo como um todo, para ser executado tanto no *Simulador de Conexões*, quanto na parte de *hardware,* na plataforma de desenvolvimento.

Essa *Interface de Funcionamento* exibe todas as informações referentes ao funcionamento do algoritmo. Na inicialização do algoritmo, a interface gera uma tela de informações referente à qual *bridge* está sendo executada, configurações da *bridge* (Identificador da *bridge* e *root bridge* atual), e as informações atuais referentes ao vetor de prioridades da *bridge* (*Bridge Priority Vector*). A Figura 32 ilustra tais informações.

Detalhes da *bridge* são exibidos após sua inicialização para fins de controle, como número de portas da *bridge*, identificador das portas (*PortId*), versão do STP executando em cada porta (*Version*), modo operacional (porta ativa ou desligada), custo de cada enlace referente a porta (*Cost*), canal referente a cada porta da *bridge* (apenas para o simulador – *Channel*, explicado no Capítulo 4.1), papéis das portas (*PortRole*), modo *edge* (*Edge Mode*), e estado das portas (*Port State*). Para as informações sobre papéis das portas (*Root port – R, Designated port – D, Alternate port – A, Backup port – B)* e estado das portas (*Discarding – D, Learning – L, Forwarding – F),* foram utilizados as

iniciais dos seus respectivos estados e papéis para serem exibidos na interface. Para o campo de modo *edge* é exibido seu estado em sim (Y) ou não (N). Estes detalhes podem ser vistos na Figura 33.

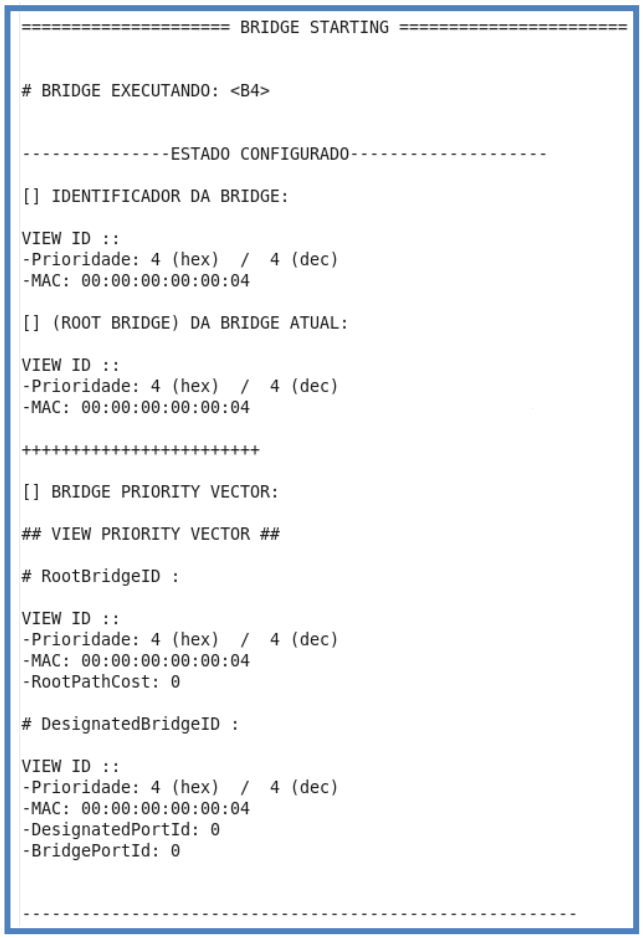


Figura 32 – Tela inicial de execução do algoritmo.

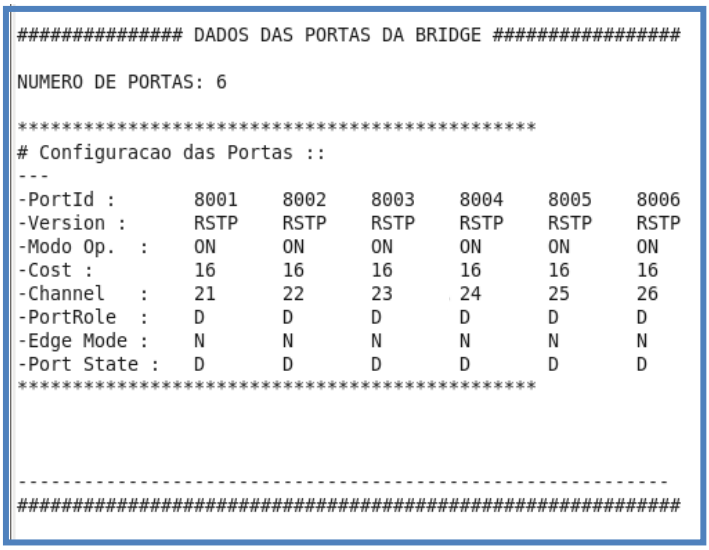


Figura 33 – Tela de dados das portas da *bridge*.

A interface exibe avisos de alterações no algoritmo, como mudança de topologia, que ocorre devido a troca de papéis das portas, a atualização da *root bridge* do segmento, e a atualização do *Root Path Cost*. Detalhes descritos na Figura 34.

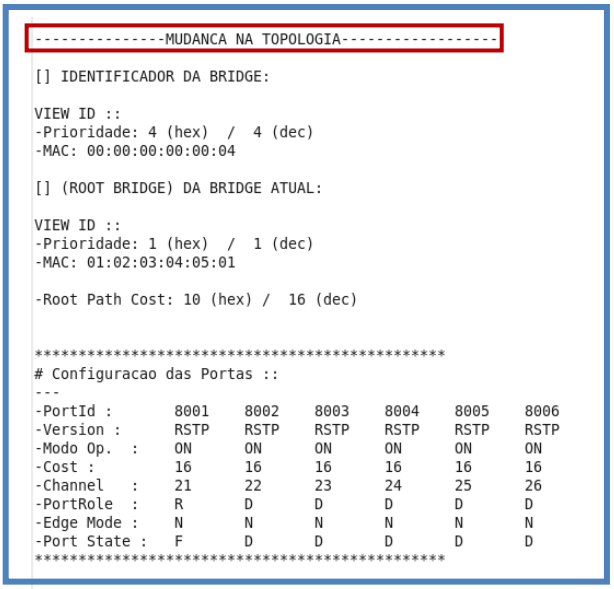


Figura 34 – Tela de mudança de topologia.

Atualizações de PPV (*Port Priority Vector*) de cada porta e atualizações do BPV (*Bridge Priority Vector*) indicando *Root Path Cost*, também geram avisos na interface, como mostrado na Figura 35.

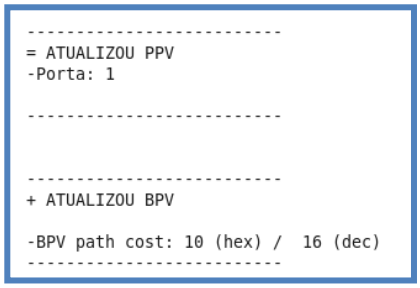


Figura 35 – Tela de atualização de PPV e BPV.

As mudanças de estado de cada porta da *bridge* geram avisos contendo suas modificações e demais informações pertinentes a *bridge*, como apresentado na Figura 36.

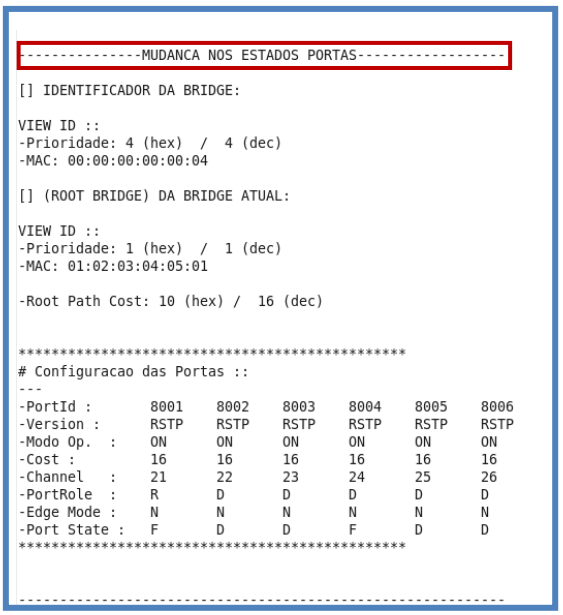


Figura 36 – Tela de mudança dos estados das portas.

Caso o *timer Forward Delay* tenha transcorrido para determinadas portas com papel *Designated*, a interface gera um aviso específico, exibindo tais portas transitando do estado *Discarding* para *Learning*, e logo após, *Forwarding*. Esta ilustração está presente nas Figuras abaixo (Figura 37, Figura 38, Figura 39), onde as portas três, quatro, cinco e seis destacadas, trocam seus estados mediante termino do *timer Forward Delay*.

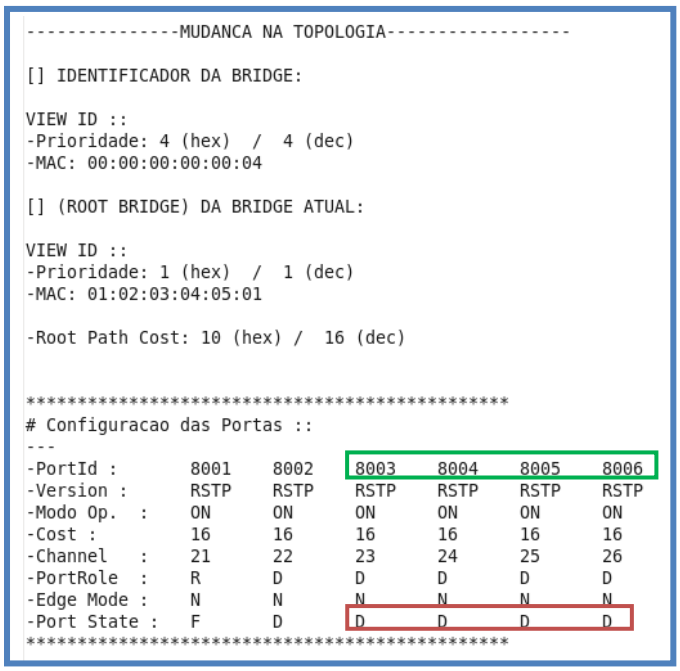


Figura 37 – Tela com portas em estado *Discarding* durante o *timer Forward Delay*.

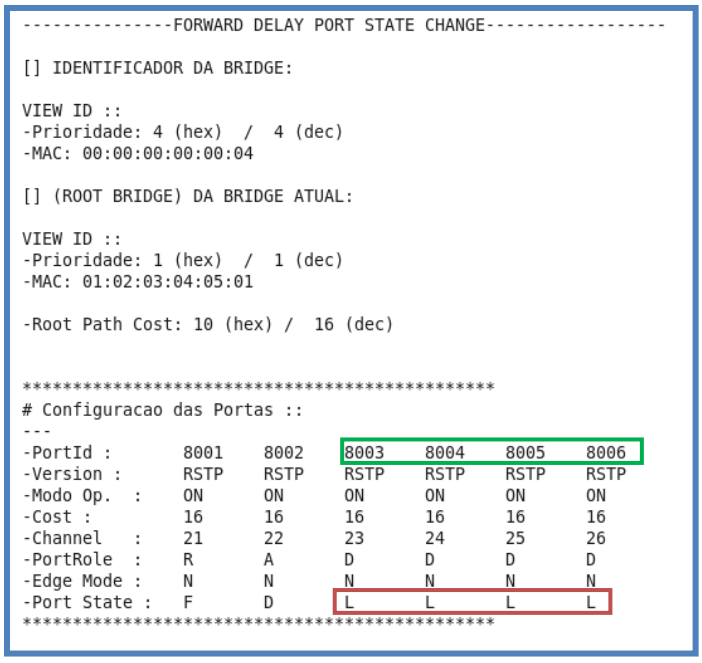


Figura 38 – Tela com portas em estado *Learning* após fim do *timer*.

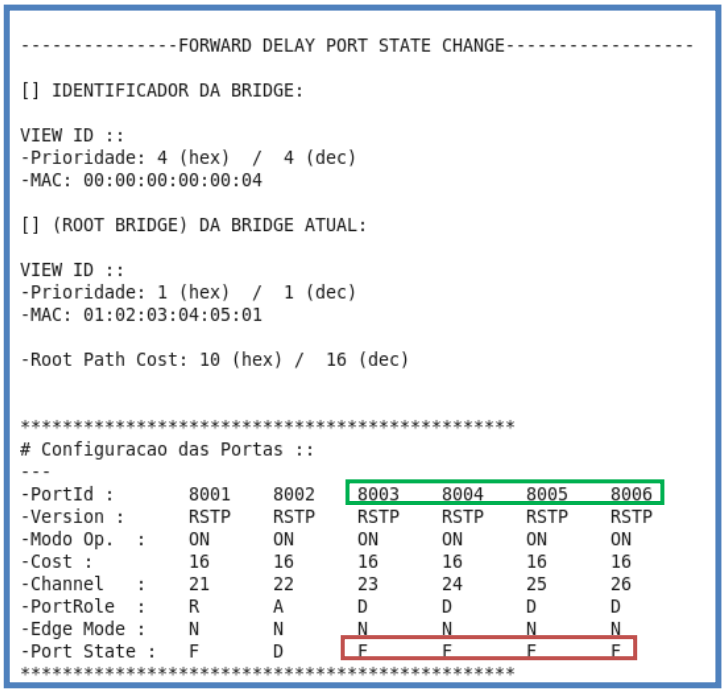


Figura 39 – Tela com portas em estado *Forwarding* após *timer*.

4. SIMULAÇÃO DO ALGORITMO RSTP

O principal propósito do simulador é permitir experimentar o algoritmo RSTP sem a necessidade da montagem de topologias de redes físicas, pois ele realiza a virtualização dos fios das diversas *bridges* em teste. O simulador permite a realização de testes mais complexos (com um maior número de *bridges* e portas), somado a praticidade e velocidade, dado o grande número de cabos de rede e plataformas de desenvolvimento necessárias para elaborar topologias complexas fisicamente. A parte da implementação que compreende o simulador está apresentada em destaque na Figura 40.

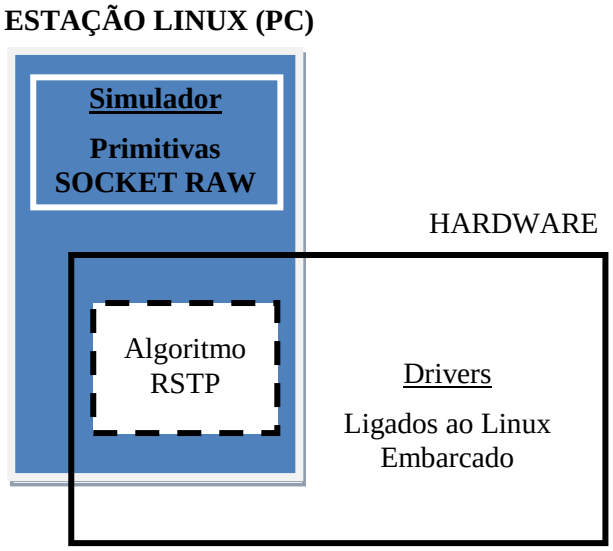


Figura 40 – Representação da implementação no simulador.

O *Simulador de Conexões* interage com as várias *bridges* (ou seja, vários terminais rodando o protocolo RSTP), através de SOCKET RAW (*socket* que permite enviar e receber diretamente pacotes de rede por aplicações, evitando todos encapsulamentos no software de rede do sistema operacional) pela interface de rede *l0* (interface de rede interna do sistema Linux - *Loopback*). O sistema do simulador é executado em um terminal Linux separado, totalmente independente das *bridges*, garantindo uma simulação com uma topologia similar a topologia física.

O simulador compreende um *Simulador de Conexões* e "*n*" instâncias de *bridges*, executando em terminais, contendo o algoritmo RSTP propriamente dito, descrito no Capítulo 3. A Figura 41 ilustra esta estrutura. A Seção 4.1 detalha o funcionamento do simulador, com suas características e especificações.

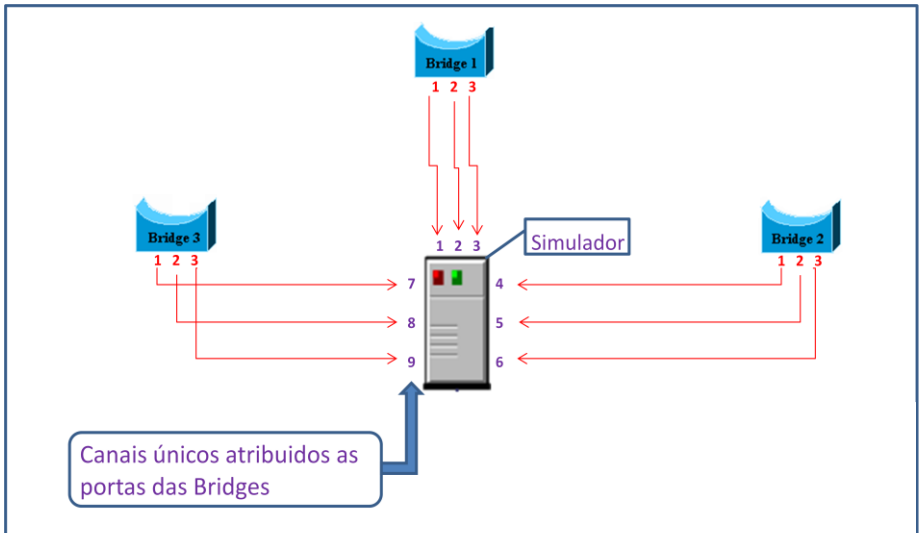


Figura 41 – Ilustração do sistema do simulador de conexões. Canais são números atribuídos as portas das *bridges* para que as mesmas tenham uma identificação única, com a finalidade de informar ao simulador a origem do pacote, devido à utilização da interface l0 do Linux. Este problema é melhor explicado no Capítulo 4.1.

4.1 Simulador de Conexões

O simulador inicia fazendo uma leitura de um arquivo de configuração, previamente criado, contendo informações sobre as ligações entre as *bridges*. Assim, com estas informações sobre a topologia, o simulador monta em seu sistema uma matriz de conexões. O arquivo é lido tanto pelo simulador quanto pelo protocolo das *bridges*. No caso das *bridges*, elas lêem informações do arquivo referente ao número de portas que a mesma terá, se estão ativas, o custo do enlace de cada conexão (referente a cada porta), e seu ID (composto pelo campo de Prioridade e pelo campo de MAC). A Figura 42 apresenta um exemplo de arquivo de topologia.

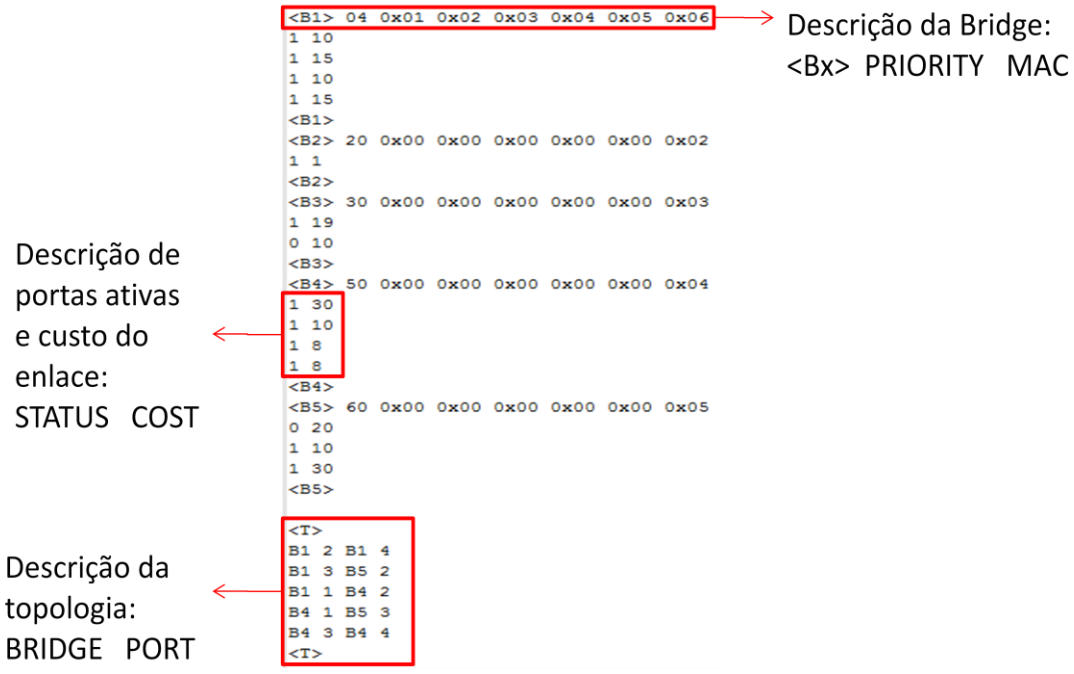


Figura 42 – Exemplo de arquivo utilizado para definir a topologia no simulador.

Cada símbolo <Bx>, referencia uma *bridge* "x", com seus dados das portas abaixo, sendo lido somente pelo protocolo das *bridges*. O símbolo <T>, referencia a descrição da topologia, sendo lido pelo simulador. Cada linha da descrição da topologia tem um formato do tipo *Bx n By m*, onde B significa que uma *bridge* "x" com porta "n" está conectada a uma *bridge* "y" na porta "m" (Ex: Bx n By m).

A interface *l0* do Linux é como se fosse uma porta de rede interna do próprio sistema operacional, sendo assim, enviando-se um pacote nesta interface, o mesmo pode ser lido pela mesma aplicação que o enviou. Isso geraria problema, pois todas as *bridges* executando o protocolo enviam pacotes para a *l0*, e este pacote BPDU não tem destino na sua formatação, pois é enviado como se fosse *broadcast*. Devido a esse fato, cria-se um sistema de identificação dos pacotes enviados pelas *bridges*, a fim de poder realizar a virtualização de fios. Sem esse controle, todas as *bridges* estariam ligadas entre todas, não existindo conexões distintas entre elas, se assemelhando a um barramento.

Cada linha do arquivo, até o fim das declarações das *bridges,* gera um número único de acordo com sua linha, atribuído para cada porta de cada *bridge.* Esse número é gravado na matriz de controle do simulador. Esta identificação é denominada "canal" (Figura 41), com o propósito de que cada porta de cada *bridge* na topologia, tenha um identificador único para assim ter um controle dos pacotes que as mesmas enviam para as demais *bridges*.

O pacote BPDU que é enviado pelas *bridges* sofre uma modificação, adicionando-se seis caracteres de controle no inicio do pacote. Os três primeiros caracteres servem para identificar o pacote, que pertence ao sistema do *Simulador de Conexões*, preenchidos com "666", número adotado para a identificação. Os próximos dois caracteres referem-se ao "canal" atribuído a cada porta, na leitura do arquivo. O próximo caractere refere-se à direção do pacote, se ele está sendo enviado para o simulador, ou para uma *bridge*.

Uma *bridge* envia um pacote BPDU na interface *l0*, com a direção de controle referente ao simulador. O simulador, por sua vez, recebe este pacote, analisa os caracteres de identificação (os três caracteres de controle de identificação), e verifica se o caractere de direção está marcado para o simulador. Na seqüência, são armazenados no simulador, os dois caracteres subseqüentes de controle adicional do BPDU, que se referem ao "canal". Com este canal recebido, o simulador faz uma procura em sua matriz de conexões, procurando qual *bridge* com determinada porta possui este canal. Obtendo esta informação, o simulador recebe a linha da matriz em que a *bridge* em questão se encontra. Esta linha representa o enlace em que esta *bridge* está ligada. Assim, o sistema conhece todas as *bridges* com suas determinadas portas, que estão se conectando através desse enlace. O simulador então repassa o BPDU recebido pela *bridge* em questão, às demais *bridges* descritas naquela linha da matriz de conexões, ou seja, que estão ligadas pelo mesmo enlace. Ao repassar o pacote às *bridges*, o simulador troca o caractere de direção, na parte de controle adicional no BPDU, para *bridges*, possibilitando a recepção de pacotes pelas mesmas.

Além de resolver o problema de envio e recebimento de pacotes pela mesma aplicação,

utilizando a *l0*, o canal serve também para que o simulador não precise analisar o BPDU enviado pelas *bridges*. Ele somente analisa os *caracteres* de controle, para saber para quem precisa repassar o pacote, evitando maior complexidade de leitura do pacote, agilizando o processo.

Sintetizando, o S*imulador de Conexões* simplesmente repassa os pacotes BPDUs enviados entre as *bridges* interligadas, virtualizando assim os fios de uma topologia real. A Figura 41 mencionada anteriormente ilustra o S*imulador de Conexões*.

O *Simulador de Conexões* possui em paralelo, uma *thread* de configuração em tempo de execução. Esta *thread* possibilita ao usuário, configurar o sistema de *bridges* que estão interligadas no momento. É possível fazer a conexão de *bridges* novas, desligar suas portas, incluir portas novas e remover *bridges*. Para este feito, o arquivo de entrada para o simulador, onde são declaradas as conexões e as *bridges*, necessita primeiramente ser alterado a fim de atender a modificação desejada. Após esse passo, se utiliza da *thread* de configuração, para a releitura deste mesmo arquivo. Esta operação é realizada inserindo o comando ATUALIZAR na *thread* de configuração, como exemplificada na Figura 43.

Este procedimento faz com que o *Simulador de Conexões* reinicialize a matriz de conexões, redimensionando, se necessário, para a nova topologia inserida no arquivo. Paralelamente a isso, o *kernel do* algoritmo RSTP, possui implementado, uma *thread* de controle de estados físicos das portas, chamado TIME LINKSTATE POLLING. Esta *thread* executa um *polling* sobre todas as portas, a fim de obter o estado físico das mesmas, ligado ou desligado. Para o *Simulador de Conexões*, estas informações são obtidas através do arquivo de configuração, que ao ser alterado, o simulador obtém as novas configurações e automaticamente repassa as *bridges* e suas portas.

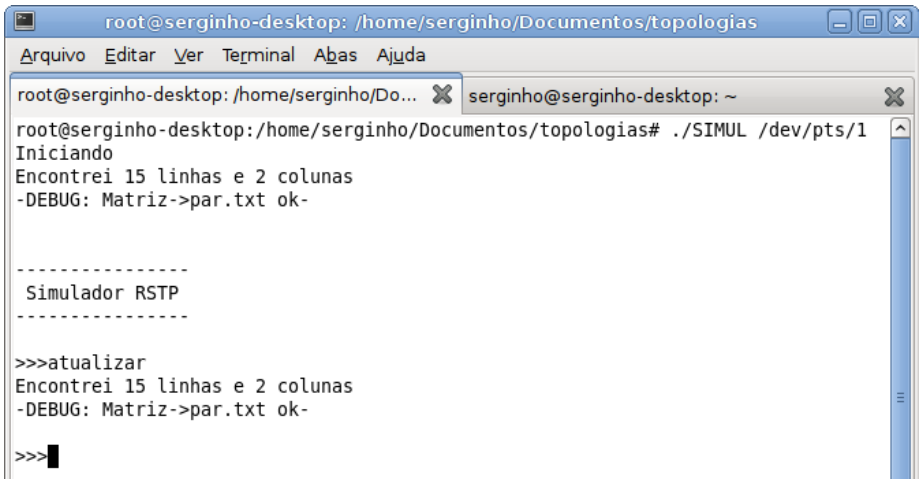


Figura 43 – Interface de configuração do simulador.

Para um controle do sistema como um todo, o *Simulador de Conexões* possui uma interface de controle de BPDUs, onde é exibido em um terminal os pacotes recebidos pelas *bridges* com uma breve descrição da parte inicial do BPDU recebido. Além dos BPDUs recebidos, a interface mostra o mesmo pacote recebido sendo repassado ao seu destino. Assim se tem um controle de que o pacote recebido foi encaminhado ao seu destino, como se fosse um *checkout* do simulador, validando a correta passagem por ele. Esta explanação pode ser verificada na Figura 44.

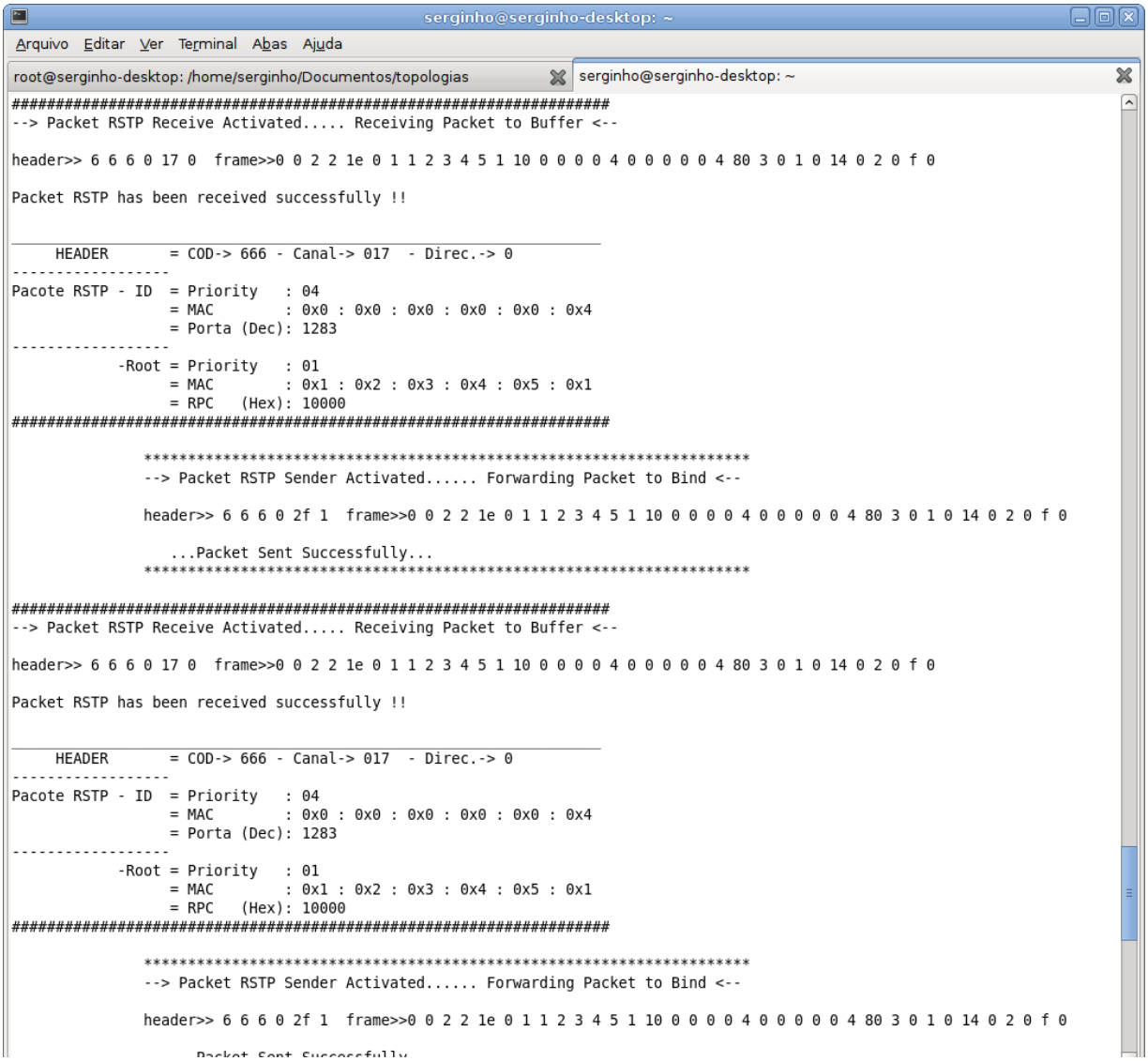


Figura 44 – Interface de controle de pacotes BPDUs do simulador.

A *Interface de Funcionamento* do algoritmo RSTP pelo simulador é a mesma utilizada para a plataforma de *hardware*, ou seja, está implementada no *kernel* do algoritmo para *debug* do sistema como um todo. Esta interface é explanada no Capítulo 3.7.

4.2 Avaliação do Algoritmo RSTP através do Simulador

A fim de validar o algoritmo RSTP com o funcionamento do *Simulador de Conexões*, é simulada uma topologia abrangendo todos estados RSTP possíveis das portas das *bridges* e a reconvergência da topologia, após alteração da mesma.

A topologia de teste contém sete *bridges* interligadas. Tanto a *bridge* 1 quanto a *bridge* 4 contêm uma conexão com um computador (porta *edge*). A topologia está disposta segundo a Figura 45 abaixo. O arquivo de entrada do simulador, o *par.txt*, está disposto conforme apresentado na Tabela 3.

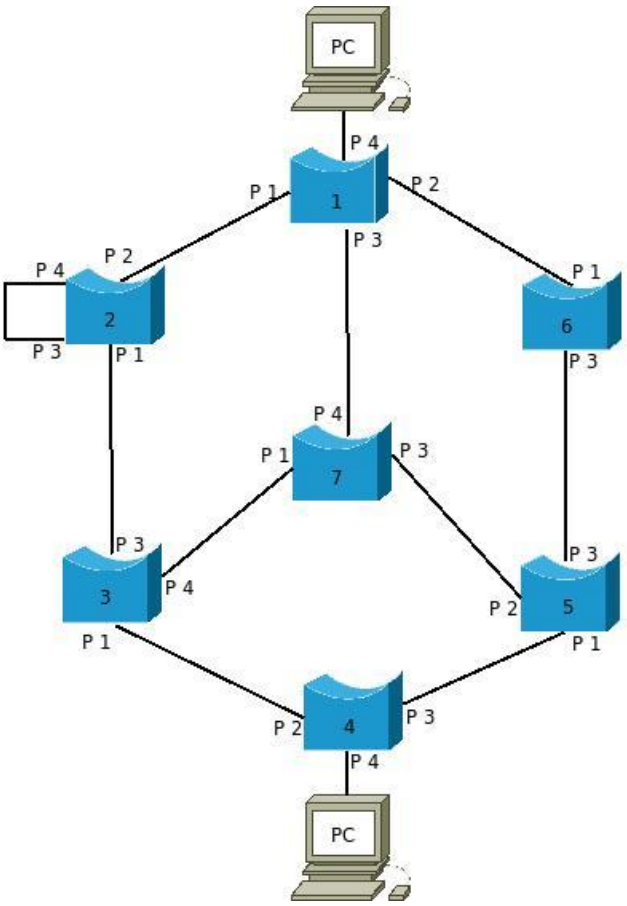
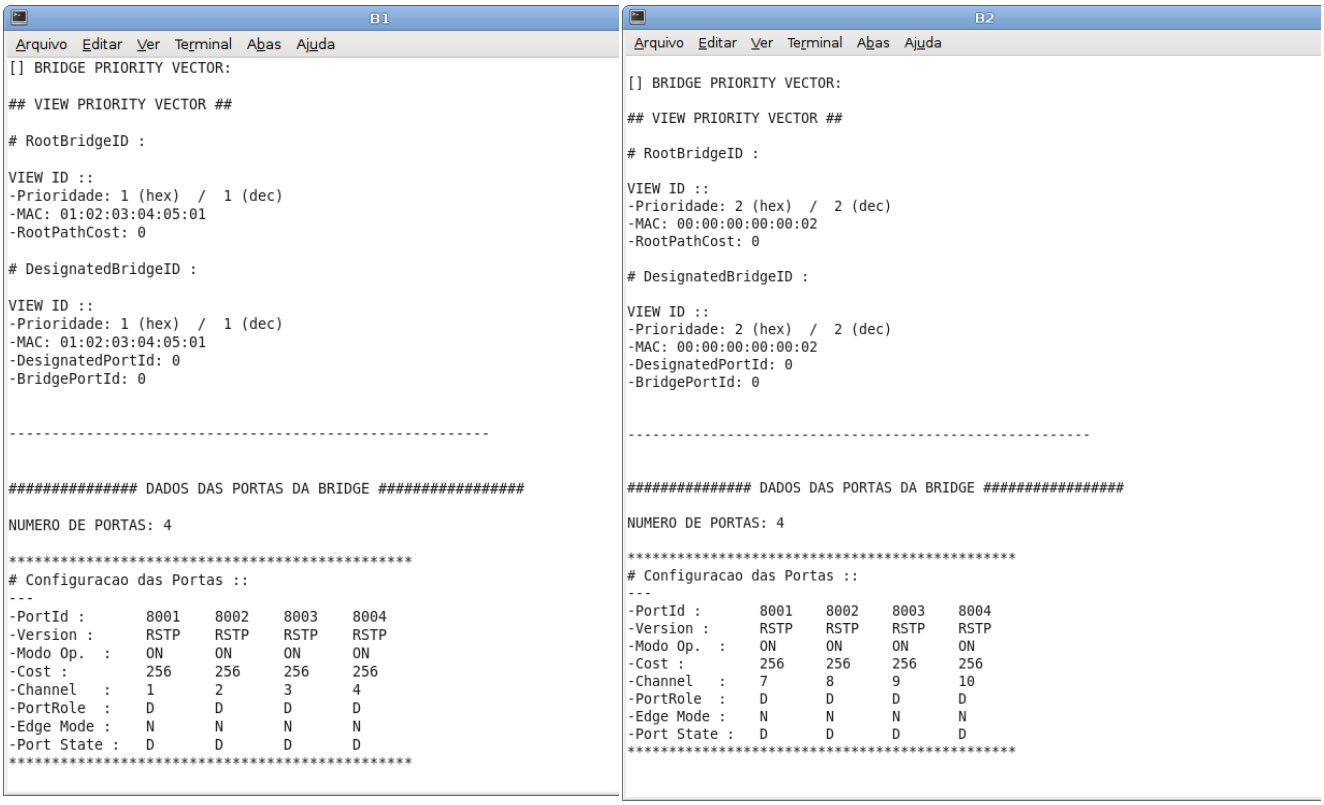


Figura 45 – Topologia de teste do simulador.

Tabela 3 – Arquivo de descrição da topologia.

<B01> 1 0x01 0x02 0x03 0x04 0x05 0x01 1 100 1 100 1 100 1 100 <B01> <B02> 2 0x00 0x00 0x00 0x00 0x00 0x02 1 100 1 100 1 100 1 100 <B02> <B03> 3 0x00 0x00 0x00 0x00 0x00 0x03 1 100 0 100 1 100 1 100 <B03> <B04> 4 0x00 0x00 0x00 0x00 0x00 0x04 0 100 1 100 1 100 1 100 <B04> <B05> 5 0x00 0x00 0x00 0x00 0x00 0x05 1 100 1 100 1 100 0 100 <B05>	<B06> 6 0x00 0x00 0x00 0x00 0x00 0x06 1 100 0 100 1 100 0 100 <B06> <B07> 7 0x00 0x00 0x00 0x00 0x00 0x07 1 100 0 100 1 100 1 100 <B07> <T> B01 01 B02 02 B02 04 B02 03 B02 01 B03 03 B03 01 B04 02 B04 03 B05 01 B05 03 B06 03 B06 01 B01 02 B07 04 B01 03 B07 01 B03 04 B07 03 B05 02 <T>

Inicializando o simulador com as sete *bridges* configuradas, seus estados iniciais são apresentados na Figura 46 e Figura 47 abaixo.



B1

Arquivo Editar Ver Terminal Abas Ajuda

```
[] BRIDGE PRIORITY VECTOR:

## VIEW PRIORITY VECTOR ##

# RootBridgeID :

VIEW ID ::
-Prioridade: 1 (hex)  /  1 (dec)
-MAC: 01:02:03:04:05:01
-RootPathCost: 0

# DesignatedBridgeID :

VIEW ID ::
-Prioridade: 1 (hex)  /  1 (dec)
-MAC: 01:02:03:04:05:01
-DesignatedPortId: 0
-BridgePortId: 0


-----------------------------------------------------


############### DADOS DAS PORTAS DA BRIDGE #################

NUMERO DE PORTAS: 4

**********************************************
# Configuracao das Portas ::
---
-PortId :       8001    8002    8003    8004
-Version :      RSTP    RSTP    RSTP    RSTP
-Modo Op.  :    ON      ON      ON      ON
-Cost :         256     256     256     256
-Channel   :    1       2       3       4
-PortRole  :    D       D       D       D
-Edge Mode :    N       N       N       N
-Port State :   D       D       D       D
**********************************************
```

B2

Arquivo Editar Ver Terminal Abas Ajuda

```
[] BRIDGE PRIORITY VECTOR:

## VIEW PRIORITY VECTOR ##

# RootBridgeID :

VIEW ID ::
-Prioridade: 2 (hex)  /  2 (dec)
-MAC: 00:00:00:00:00:02
-RootPathCost: 0

# DesignatedBridgeID :

VIEW ID ::
-Prioridade: 2 (hex)  /  2 (dec)
-MAC: 00:00:00:00:00:02
-DesignatedPortId: 0
-BridgePortId: 0


-----------------------------------------------------


############### DADOS DAS PORTAS DA BRIDGE #################

NUMERO DE PORTAS: 4

**********************************************
# Configuracao das Portas ::
---
-PortId :       8001    8002    8003    8004
-Version :      RSTP    RSTP    RSTP    RSTP
-Modo Op.  :    ON      ON      ON      ON
-Cost :         256     256     256     256
-Channel   :    7       8       9       10
-PortRole  :    D       D       D       D
-Edge Mode :    N       N       N       N
-Port State :   D       D       D       D
**********************************************
```

Bridge 1 **Bridge 2**

B3

Arquivo Editar Ver Terminal Abas Ajuda

```
[] BRIDGE PRIORITY VECTOR:

## VIEW PRIORITY VECTOR ##

# RootBridgeID :

VIEW ID ::
-Prioridade: 3 (hex)  /  3 (dec)
-MAC: 00:00:00:00:00:03
-RootPathCost: 0

# DesignatedBridgeID :

VIEW ID ::
-Prioridade: 3 (hex)  /  3 (dec)
-MAC: 00:00:00:00:00:03
-DesignatedPortId: 0
-BridgePortId: 0


-----------------------------------------------------


############### DADOS DAS PORTAS DA BRIDGE #################

NUMERO DE PORTAS: 4

**********************************************
# Configuracao das Portas ::
---
-PortId :       8001    8002    8003    8004
-Version :      RSTP    RSTP    RSTP    RSTP
-Modo Op.  :    ON      OFF     ON      ON
-Cost :         256     256     256     256
-Channel   :    13      14      15      16
-PortRole  :    D       U       D       D
-Edge Mode :    N       N       N       N
-Port State :   D       D       D       D
**********************************************
```

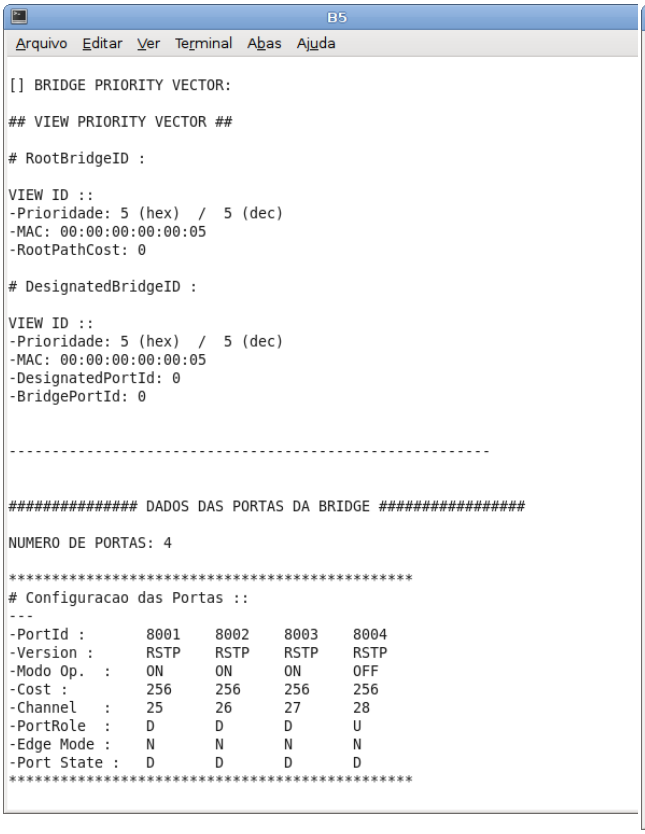
B4

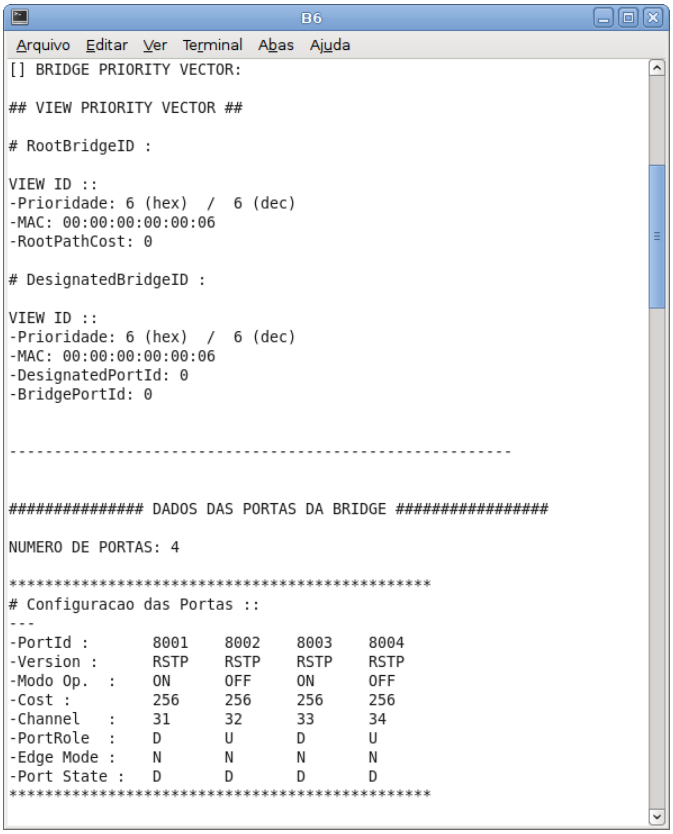
Arquivo Editar Ver Terminal Abas Ajuda

```
[] BRIDGE PRIORITY VECTOR:

## VIEW PRIORITY VECTOR ##

# RootBridgeID :

VIEW ID ::
-Prioridade: 4 (hex)  /  4 (dec)
-MAC: 00:00:00:00:00:04
-RootPathCost: 0

# DesignatedBridgeID :

VIEW ID ::
-Prioridade: 4 (hex)  /  4 (dec)
-MAC: 00:00:00:00:00:04
-DesignatedPortId: 0
-BridgePortId: 0


-----------------------------------------------------


############### DADOS DAS PORTAS DA BRIDGE #################

NUMERO DE PORTAS: 4

**********************************************
# Configuracao das Portas ::
---
-PortId :       8001    8002    8003    8004
-Version :      RSTP    RSTP    RSTP    RSTP
-Modo Op.  :    OFF     ON      ON      ON
-Cost :         256     256     256     256
-Channel   :    19      20      21      22
-PortRole  :    U       D       D       D
-Edge Mode :    N       N       N       N
-Port State :   D       D       D       D
**********************************************
```

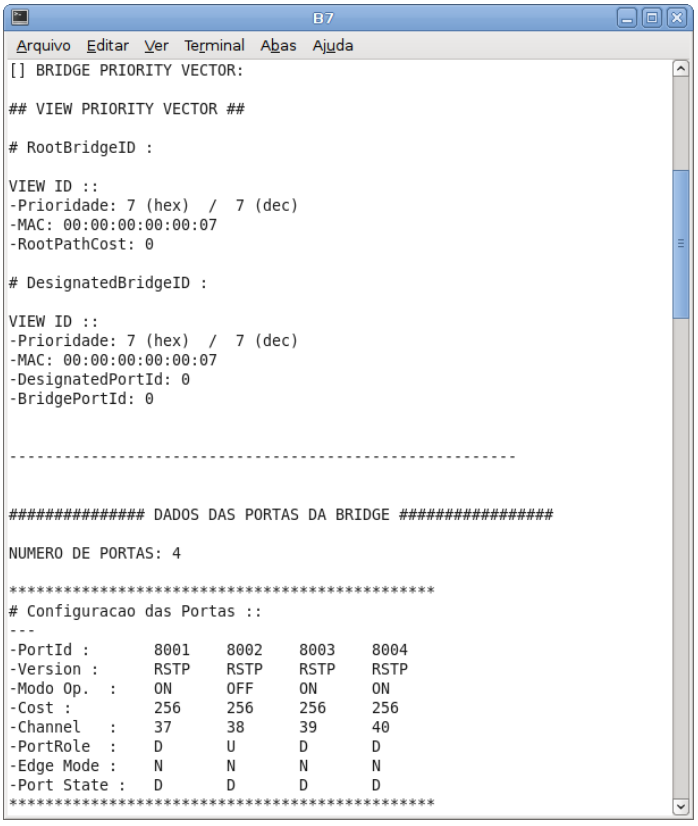
Bridge 3 **Bridge 4**

Figura 46 – Configurações iniciais das *Bridges 1 a 4* na topologia teste do simulador.





Bridge 5 **Bridge 6**



Bridge 7

Figura 47 – Configurações iniciais das *Bridges* 5 a 7 na topologia teste do simulador.

Após a inicialização das sete *bridges*, se inicia a troca de mensagens entre as mesmas. Com essa troca de BPDUs, a topologia realiza o processo de convergência, configurando as *bridges* e suas portas de forma a restar apenas um único caminho válido entre as *bridges*. Após esse procedimento, resultante do algoritmo RSTP, se obtém o resultado apresentado na Figura 49. A Figura 48 exibe a legenda para melhor compreensão do diagrama.

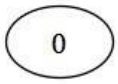
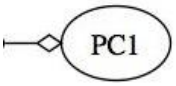
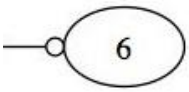
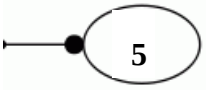
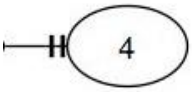
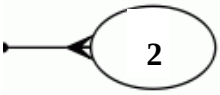
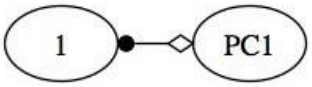
	**Exemplo de *Bridge***
	**Exemplo de PC**
	***Bridge* com porta *Root***
	***Bridge* com porta *Designated***
	***Bridge* com porta *Alternate***
	***Bridge* com porta *Backup***
	***Bridge* com porta em *Edge Mode***

Figura 48 – Legenda para compreensão do diagrama resultante da topologia convergida.

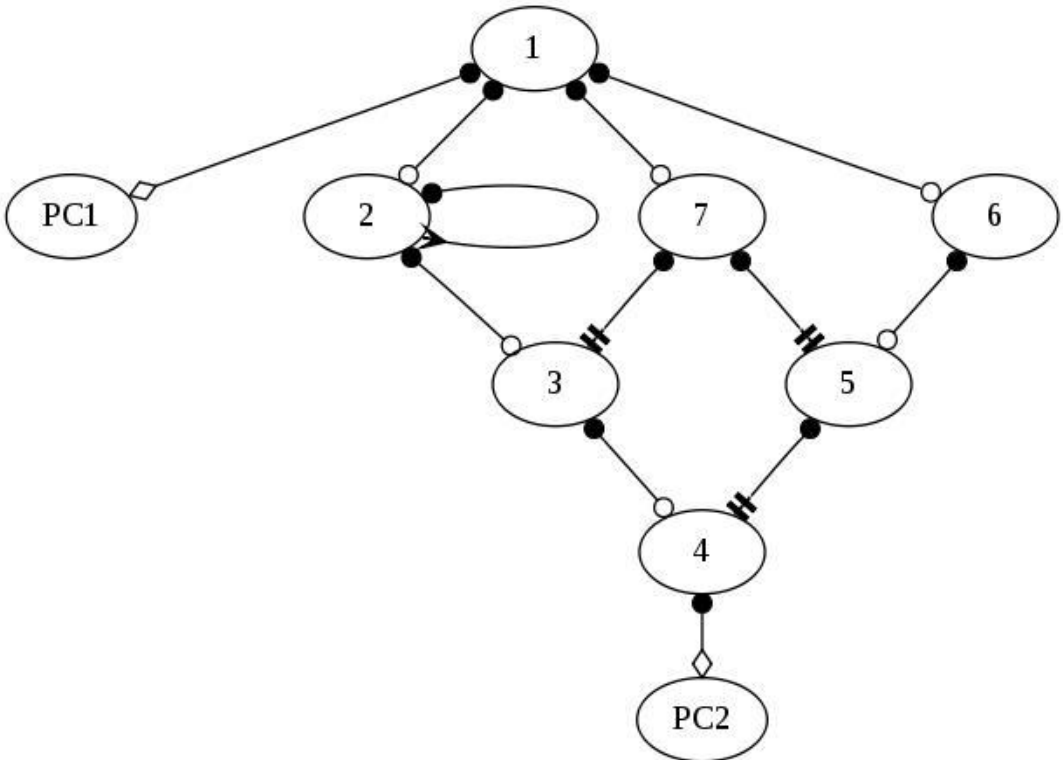
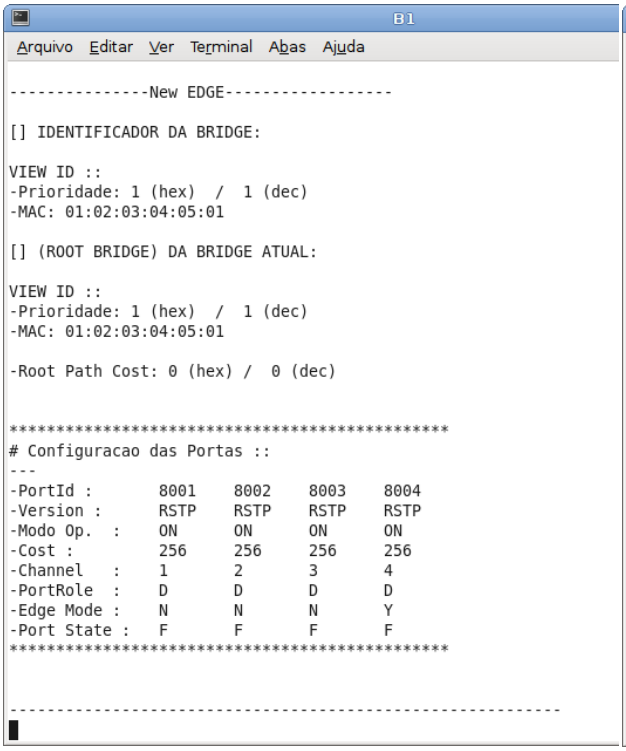
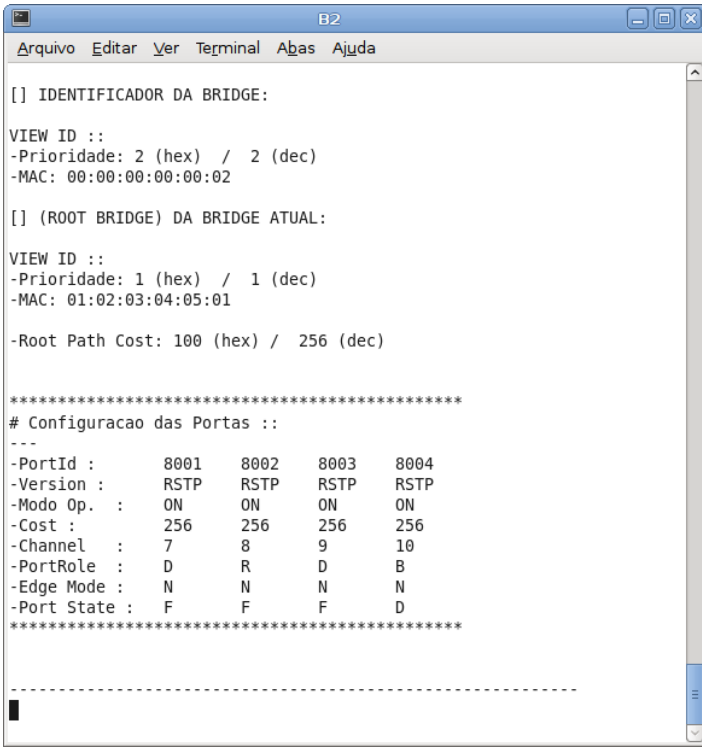


Figura 49 – Diagrama da topologia teste resultante após algoritmo RSTP.

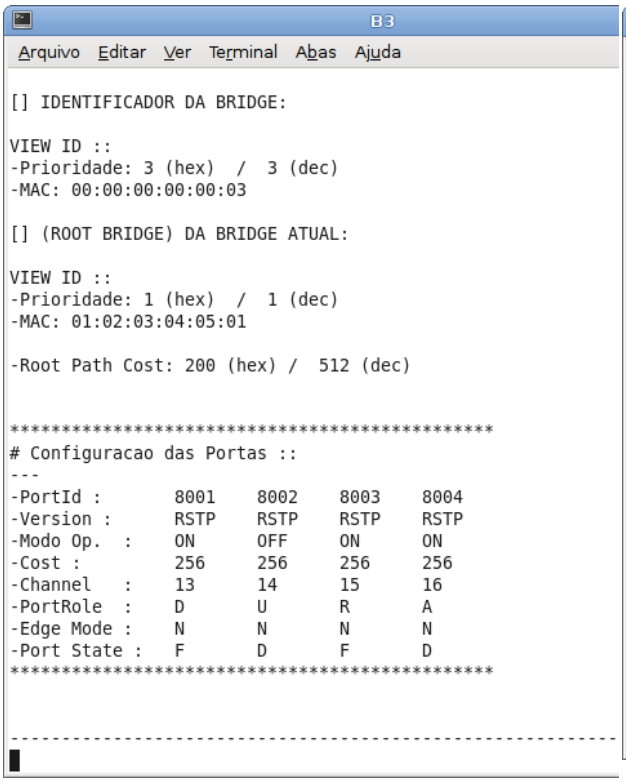
Este resultado pode ser conferido analisando as telas de interface do algoritmo, exibindo as sete *bridges* com suas portas configuradas. Esta ilustração é apresentada na Figura 50 e Figura 51.



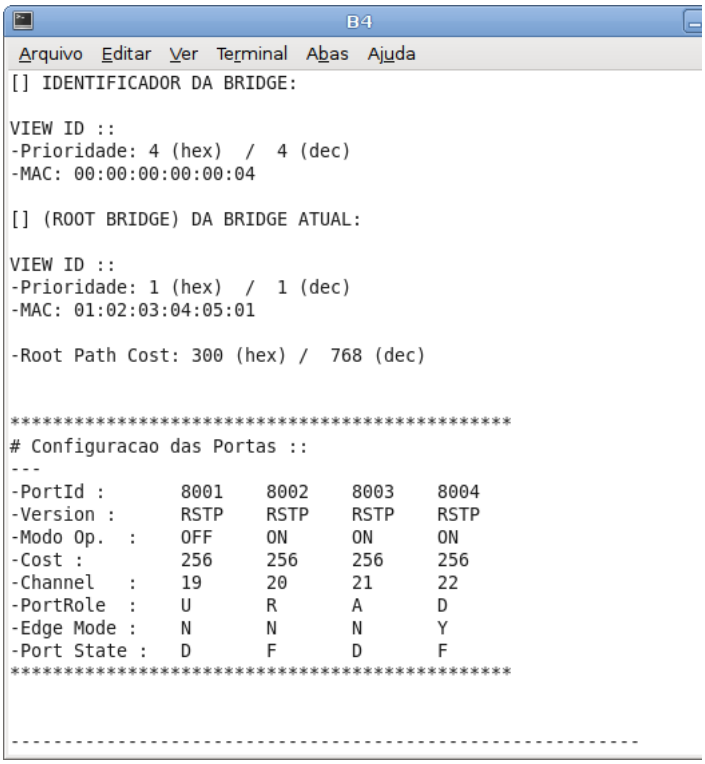
Bridge 1



Bridge 2

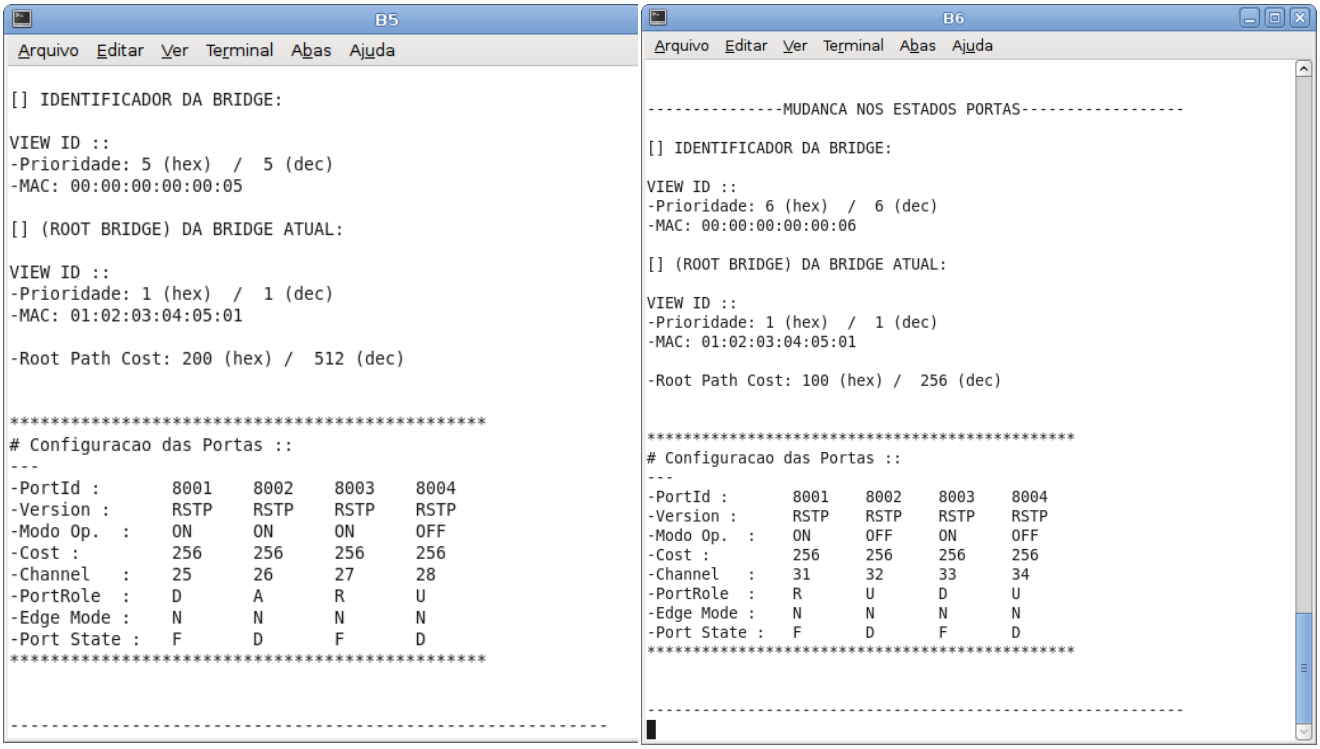


Bridge 3

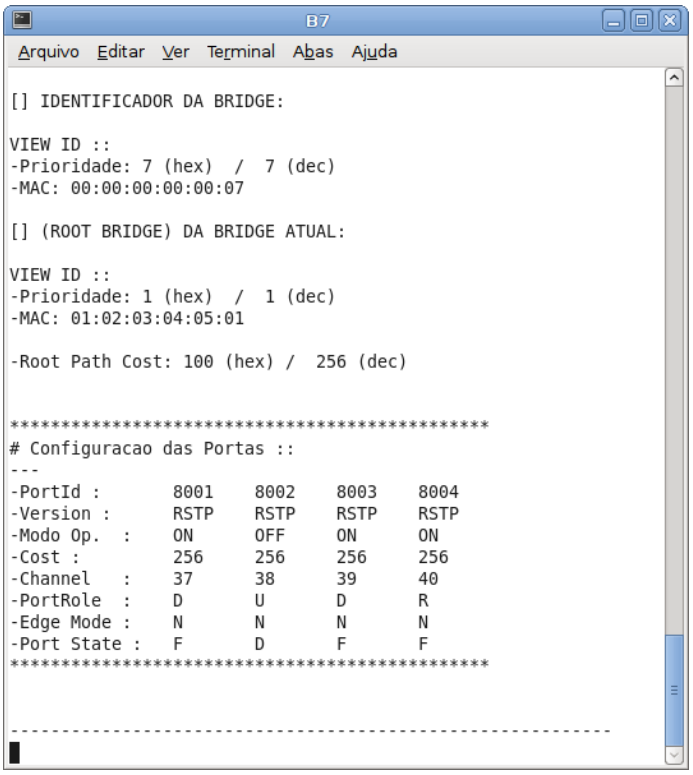


Bridge 4

Figura 50 – Configurações das *bridges* 1 a 4 após execução do algoritmo.



Bridge 5 **Bridge 6**



Bridge 7

Figura 51 – Configurações das *bridges* 5 a 7 após execução do algoritmo.

Após a convergência das *bridges*, é proposta a remoção de um enlace principal da topologia, pelo qual o caminho até a *root bridge* é realizado. A Figura 52 ilustra esta alteração.

A modificação tem o propósito de causar a maior alteração possível na topologia resultante,

obrigando o algoritmo a refazer os caminhos lógicos até a *root bridge*. Com isso, mostra-se um funcionamento completo do algoritmo RSTP implementado juntamente com o *Simulador de Conexões* para validá-lo.

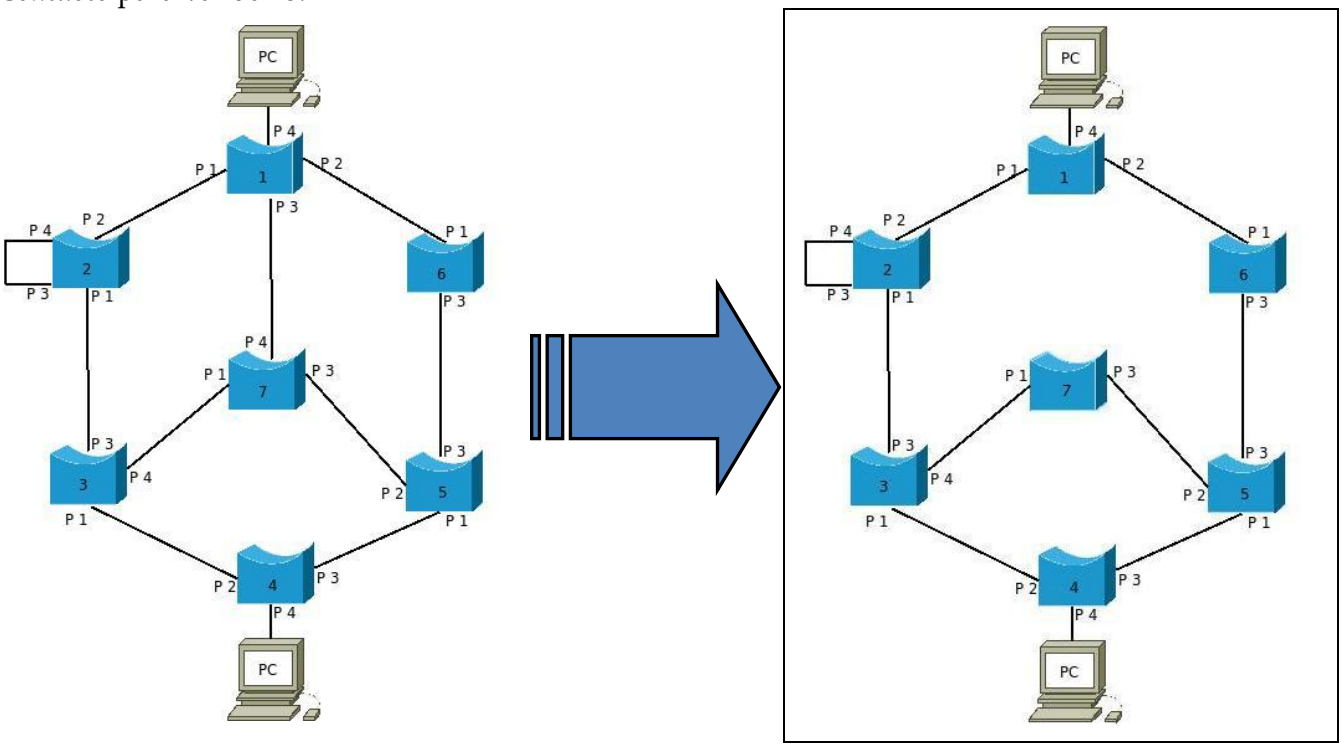


Figura 52 – Alteração no enlace da porta 4 da *bridge* 7.

Removendo o enlace da *bridge* 7 em tempo de execução no simulador, o algoritmo RSTP realiza a convergência com esta nova situação, devido a uma alteração crítica no caminho até a *root bridge* feito anteriormente. Assim, a Figura 53 mostra o diagrama da topologia convergida após a alteração.

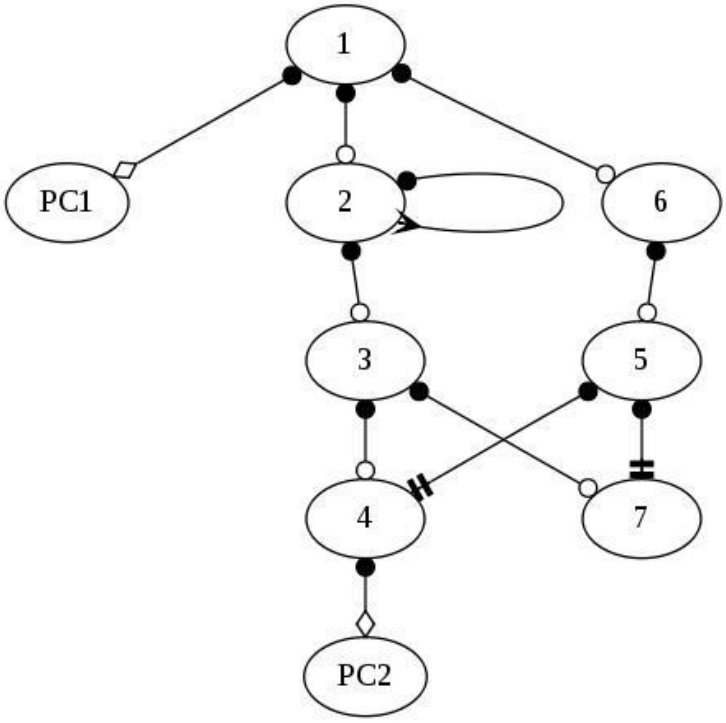
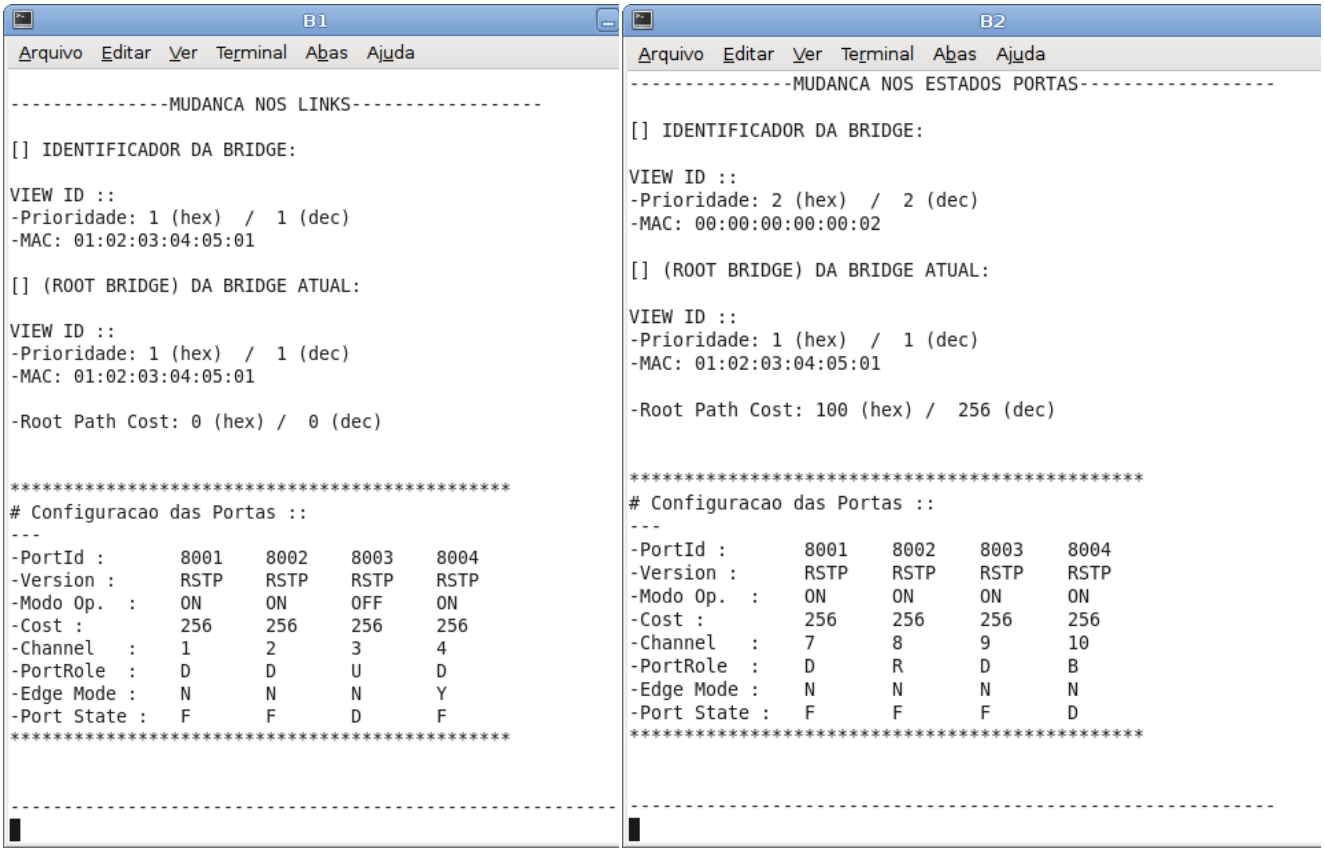
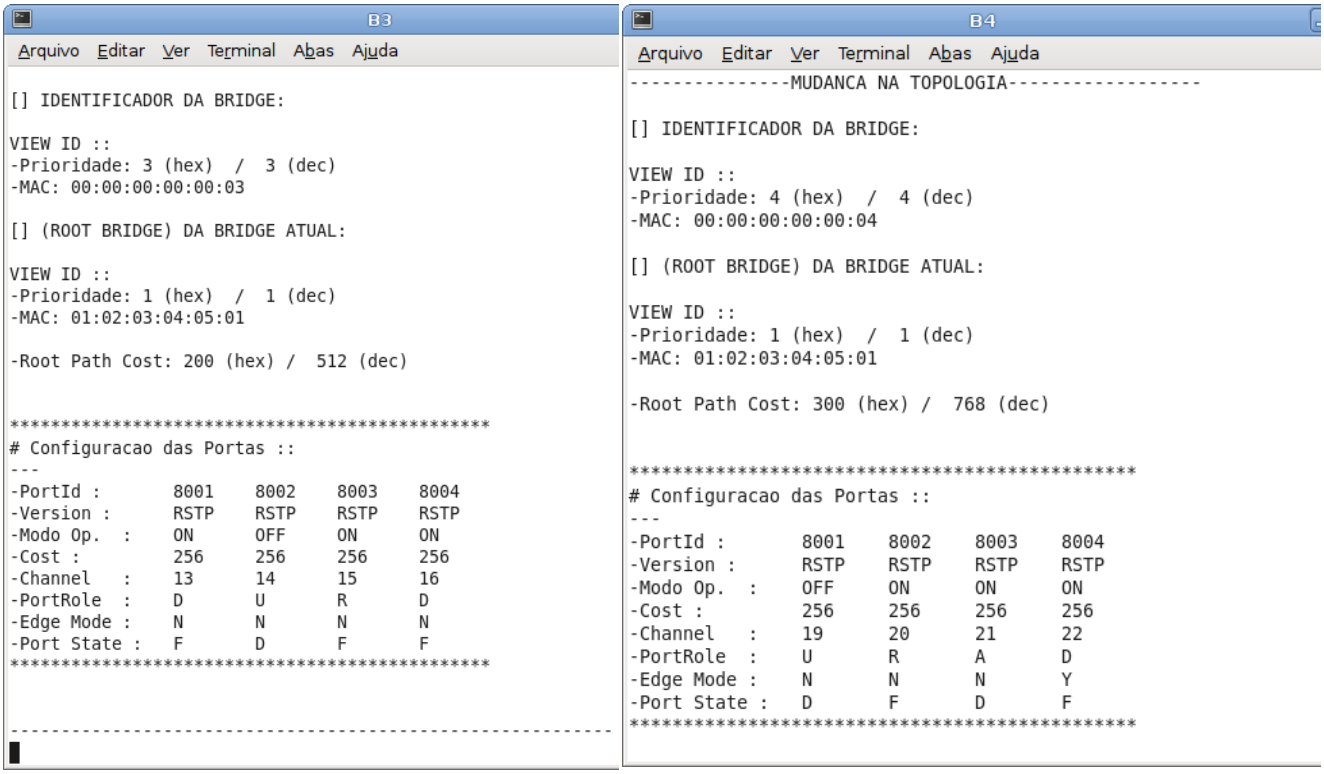


Figura 53 – Diagrama da topologia resultante após alteração do caminho até a *root bridge*.

Esta modificação é exibida nas telas de interface do algoritmo, mostrando as *bridges* com suas respectivas portas configuradas como exemplificado no diagrama da Figura 53. As telas de interface estão expostas na Figura 54 e Figura 55.

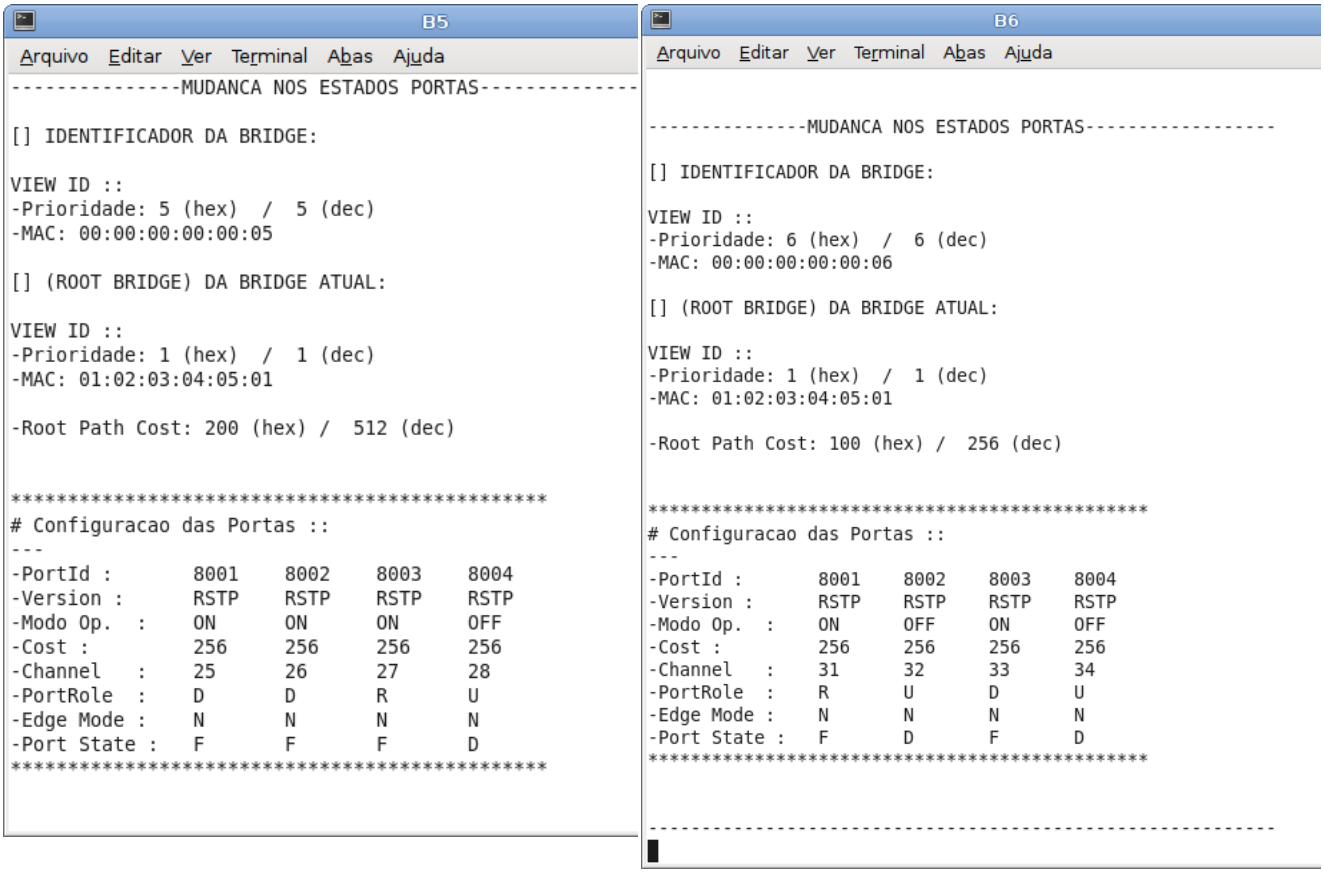


Bridge 1 **Bridge 2**

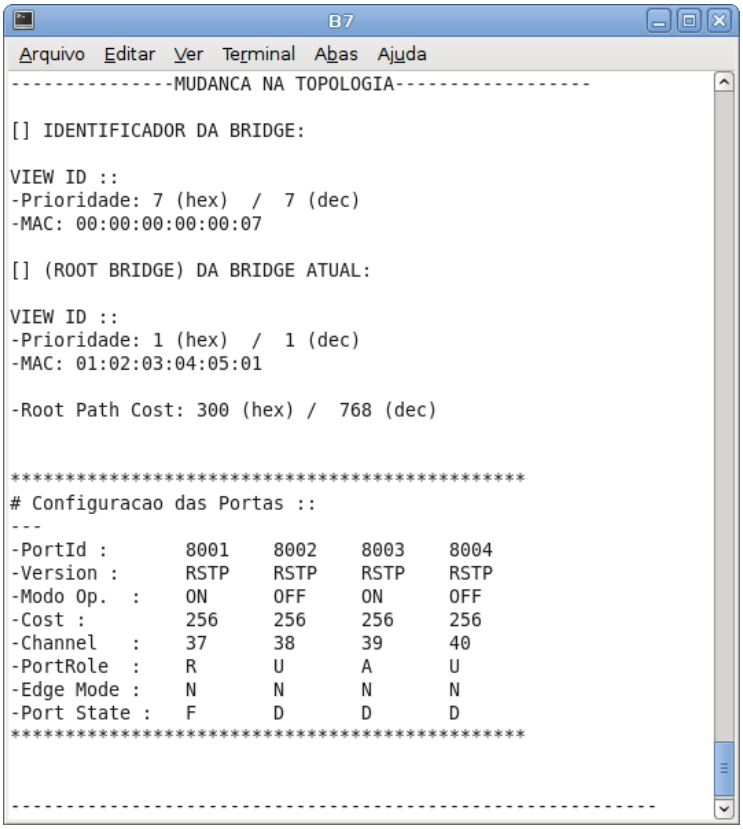


Bridge 3 **Bridge 4**

Figura 54 – Telas de interface das *bridges* 1 a 4 após modificação na topologia.



Bridge 5 **Bridge 6**



Bridge 7

Figura 55 – Telas de interface das *bridges* 5 a 7 após modificação na topologia.

5. APLICAÇÃO EM HARDWARE

Com o propósito de aplicar este algoritmo em uma situação real, o *kernel* do algoritmo RSTP foi implementado em uma plataforma de desenvolvimento, para testar o sistema e validar sua estrutura como um todo. Este procedimento objetiva anexar este algoritmo RSTP em uma *bridge* comercial, para complementar um produto e atender aplicações de nível dois da camada OSI. Esta parte da implementação está apresentada em destaque na Figura 56.

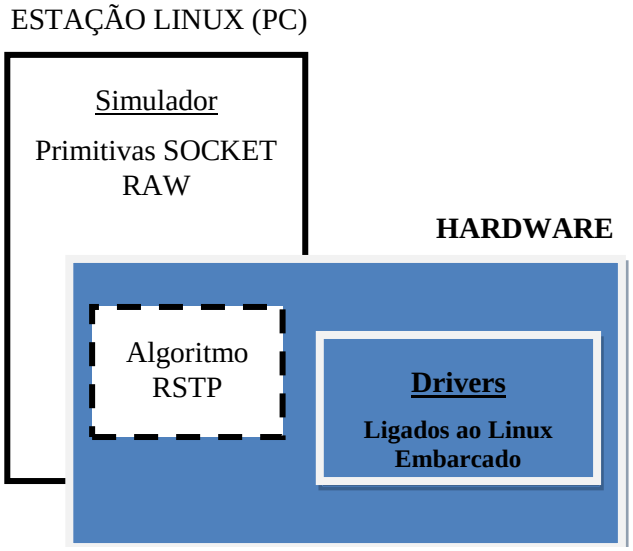


Figura 56 – Representação da implementação em *hardware*.

5.1 Plataforma de Desenvolvimento

A plataforma de desenvolvimento para esta implementação corresponde a MPC8323E *Modular Development System* (MDS), da *FreeScale*. Esta plataforma de *hardware* já possui um *switch ethernet* que provê suporte ao uso do protocolo *Spanning Tree*, permitindo que suas portas sejam colocadas nos estados previstos pelo protocolo, bem como o suporte à configuração da *SpecialTag* (*header* anexado ao pacote com a finalidade de informar a porta do *switch* que recebeu o pacote), necessária para a validação do protocolo em *hardware*. Além disso, tais plataformas são fornecidas com um sistema Linux embarcado pronto para ser modificado onde novos aplicativos podem ser desenvolvidos e incorporados ao sistema.

A Figura 57 mostra a descrição do *hardware* e a Figura 58 exibe o diagrama de blocos do *chip* MPC8323E da plataforma de desenvolvimento da *FreeScale*, apresentada na Figura 59. A placa possui um *switch ethernet* embarcado de quatro portas 10/100 e uma porta *ethernet* de controle com um *host*. As interfaces MII e MDIO possibilitam a configuração e a comunicação do *switch* com a CPU. O acesso à configuração do *switch ethernet* possibilita a atribuição de todos os estados previstos pelos protocolos RSTP/STP as portas do dispositivo, e também habilitar a inserção da *SpecialTag*. Os pacotes (BPDUs) com a *SpecialTag* inserida, são automaticamente enviados à MII1,

para a interpretação do software na CPU principal.

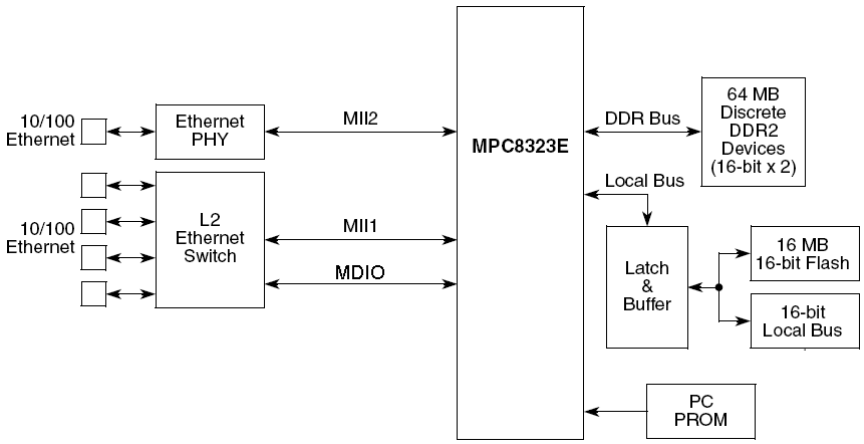


Figura 57 – Descrição do *hardware*.

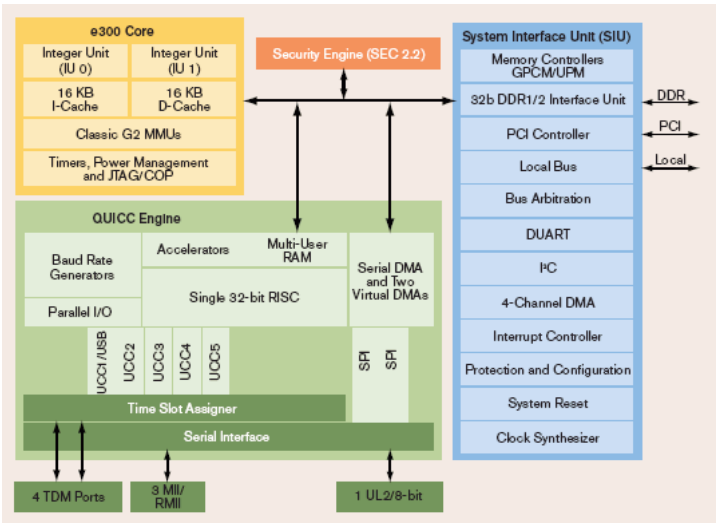


Figura 58 – Diagrama de blocos do *chip* MPC8323E (*hardware)*.

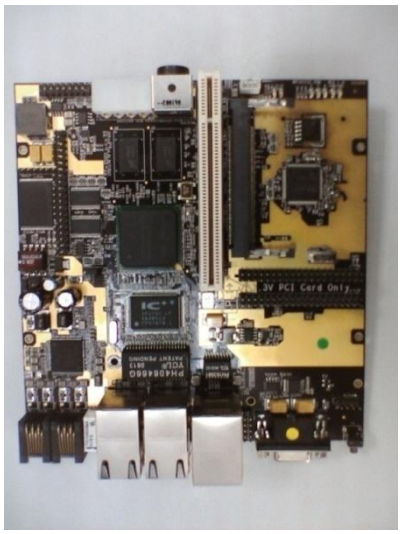


Figura 59 – Foto ilustrativa da plataforma de desenvolvimento.

O protocolo funciona atuando sobre o *switch* do *hardware*. Para atender esta característica, é necessário que a parte do sistema que acessa camadas de níveis mais baixos do protocolo controle os registradores do *hardware* de forma separada, ou seja, cada porta do *switch* em particular. Isso se deve ao fato da necessidade de leitura dos estados das portas (ligado ou desligado) e custo de enlace (depende da velocidade do enlace conectado). Tendo em vista esse detalhe, é preciso alterar o código fonte de um *driver* UCC, da seção NET da biblioteca do Linux, colocando funções que leiam e escrevam informações nos registradores do *switch*. Ocorrendo essa modificação, é implementada uma API que resgata as informações trazidas por esse *driver*, abstraindo a complexidade para a camada de usuário (OSI 7).

O envio de pacotes (BPDUs) é realizado através de *socket raw*, da mesma forma como é realizado no *Simulador de Conexões*. A diferença é a interface utilizada, onde no simulador é utilizado a *l0*, e no *hardware* é utilizada a interface correspondente ao *switch ethernet* da placa.

Sobre o sistema operacional utilizado na placa, é usado um *software* chamado LTIB, que compila o Linux embarcado, com a possibilidade de alteração do *kernel*, otimizando o sistema para sua necessidade, e carregando o sistema operacional com alguma aplicação, no caso, o protocolo RSTP. O LTIB possibilita realizar mudanças no sistema operacional, alterando características do Linux, com a finalidade de deixar o sistema mais enxuto para uma determinada aplicação. O código do algoritmo é compilado dentro do terminal LTIB, que possui um c*ross compiler* específico para as placas. Outra biblioteca específica, chamada LIBPCAP, é utilizada com o objetivo de capturar pacotes da rede. A biblioteca possui filtros prontos para captura e análise de pacotes, facilitando a implementação do algoritmo.

O controle do *hardware* pode ser feito através do software MINICOM, que virtualiza o terminal Linux da plataforma de desenvolvimento no *host* (PC). Essa comunicação é feita através da porta *serial* da plataforma com o *host*. Todas as informações geradas no *hardware* são capturadas por essa interface através do *software*, possibilitando supervisionar e controlar o sistema operacional da placa, tanto como as aplicações em execução na placa.

Devido à complexidade de usar portas seriais para todas as placas, é utilizada uma aplicação nativa do Linux, o TELNET, como ferramenta de controle e análise das placas. Cada placa é acessada por uma sessão de TELNET independente em um único *host*, concentrando todo o controle das placas. Este método facilita a análise do algoritmo de todas as placas simultaneamente, como a administração e configuração das mesmas. O acesso as placas pelo TELNET é feito através da porta *ethernet* de controle que cada placa possui, independente das portas do *switch* embarcado. Desta forma é possível monitorar o protocolo executando como aplicação no Linux embarcado do *hardware*, como forma de *debug* de uma situação física real.

Para fazer o *upload* da imagem do Linux embarcado com a aplicação do protocolo, gerado pelo LTIB, é necessário utilizar duas aplicações do Linux, os servidores TFTP (*Trivial File Transfer Protocol*) e NFS (*Network File System)*. O servidor TFTP é utilizado para transferir pequenos arquivos entre *hosts* numa rede, neste caso, entre um *host* e a plataforma de desenvolvimento. Os arquivos

enviados para a plataforma com esse sistema só são carregados após o *boot*. O servidor NFS tem o fim de compartilhar arquivos e diretórios entre *hosts* na rede, em tempo de execução dos sistemas. Ou seja, é possível modificar o diretório incluindo novos arquivos, e executar os mesmos na plataforma sem necessidade de *boot*, diferente do TFTP. A imagem do Linux embarcado é transferida para o *hardware* através do servidor TFTP, pois é necessário fazer isso com o sistema em estado de configuração, ou seja, nada está rodando no sistema. Outros arquivos anexos podem ser enviados ou compartilhados pelo servidor NFS, pois não são arquivos críticos ao sistema, não fundamentais ao *boot* da plataforma.

No *host* de controle, existem pastas de arquivos específicas para cada placa. Assim, cada placa recorre a sua pasta para o *boot* do sistema. Por exemplo, a "placa 1" está configurada para efetuar a procura de seus arquivos de *boot* na pasta do *host*, no local "/tftp/placa1". Esta pasta contém os arquivos de configuração da placa (endereço de MAC e IP) e também a imagem do *kernel* do Linux.

5.2 Integração da Plataforma de Desenvolvimento ao *Kernel* do Algoritmo

O algoritmo RSTP funciona com base nas informações das portas da *bridge*. Para que o programa tivesse acesso às informações no nível de *hardware*, foi necessário realizar modificações no *kernel* do Linux.

Dentro *kernel* do Linux, mais específico no *driver* UCC, foram adicionadas funções para possibilitar a leitura e escrita no barramento da interface MDIO. Nestas funções é utilizado como parâmetro o número do PHY, o registrador e o valor a ser escrito ou lido. Os PHYs relacionados ao *switch* totalizam 32, dos quais oito são utilizados, cinco para endereçar as portas, e três para configurações [ICP08]. Os demais PHYs são utilizados em caso de expansão do *hardware.*

Estas modificações possibilitaram o desenvolvimento de uma biblioteca chamada PHY_HAL, que realiza a interação entre o algoritmo e o *hardware*. Esta biblioteca contém uma função, *mdio_init*, que inicializa o *socket* da interface MDIO, possibilitando o uso do driver UCC modificado. Além desta função, a biblioteca faz a interface com as funções de leitura e escrita do *driver* UCC, através do controle de entrada e saída do Linux, assim as funções ficam acessíveis em nível de aplicação.

A partir dessas funções, foi criada a biblioteca SWITCH_CONTROL. Esta biblioteca, utilizando das funções da biblioteca PHY_HAL, realiza a configuração do *switch* para operar tanto em modo RSTP quanto STP, procede com a limpeza das tabelas de MAC e faz a leitura do estado físico e configuração de cada porta.

Esta biblioteca também configura o controlador do *switch* para realizar a inserção da *SpecialTag* [ICP06] nos pacotes recebidos. Esta *tag* é anexada pelo *switch* entre os campos MAC_SRC e LENGTH/TYPE do protocolo *ethernet*. Isso faz com que a identificação da porta do *switch ethernet* que recebeu o pacote seja armazenada no pacote, possibilitando ao algoritmo tratar esta informação para configuração. Para o envio de pacotes (BPDU), o algoritmo insere a *SpecialTag* correspondente

a porta desejada, monta o pacote e envia. O *switch ethernet* por sua vez, identifica que o pacote contém a *SpecialTag*, o endereça a porta correspondente e retira a *SpecialTag*, enviando o pacote na porta correspondente.

Esta inclusão da *SpecialTag* impossibilita a manipulação de *softwares* diferentes do algoritmo implementado, que utilizem protocolos de rede na placa. Isso ocorre porque o algoritmo implementado faz o tratamento desta *SpecialTag*, o que pode não acontecer com os outros *softwares*. Isso gera um tratamento errado dos pacotes. Para a correção do problema, é necessário a modificação em nível de *kernel* do Linux no *driver* de rede, para realizar a remoção da *SpecialTag* dos pacotes que não possuem o endereço MAC STP no campo MAC DESTINO do protocolo *ethernet*.

A Figura 60 [ICP06] exemplifica o funcionamento da inclusão e remoção da *SpecialTag* no *header* do protocolo *ethernet*. Os campos TP ID e TCI são adicionados ao *header*, representando a *SpecialTag*. O campo TP ID é referente à porta desejada (os dois últimos caracteres – [81xx]), e o campo TCI existe para preenchimento do *header* (recebe zero), referente ao tamanho ocupado pela *SpecialTag*.

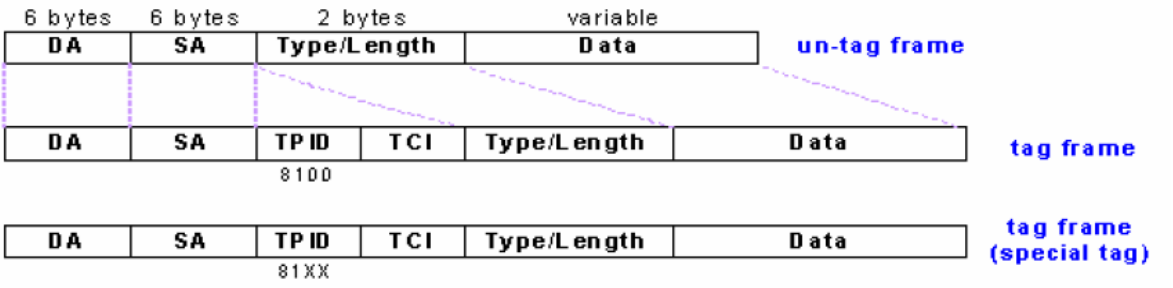


Figura 60 – Inserção e remoção da *SpecialTag*.

As principais funções desta biblioteca são apresentadas a seguir:

- *StpConfig*: Habilita a configuração da inclusão da *SpecialTag*, habilita o MAC estático para STP (permitindo somente a entrada de pacotes STP, caso a porta não esteja em estado *forwarding*) e coloca todas as portas para o estado *forwarding*. Detalhamento da função:
    - A função escreve no registrador MII23 do PHY29, o valor 0x03C2 (hex). Este valor permite a inserção da *SpecialTag* na porta cinco do barramento MII0 (barramento que direciona informação ao CPU) e realiza a remoção da *SpecialTag* das portas 0 a 3 (portas 1 a 4) na saída dos pacotes.
    - A função escreve no registrador MII26 do PHY26, o valor 0x20E0 (hex). Este valor habilita o endereço MAC estático do *spanning tree* (01:80:C2:00:00:00), limitando o trafego de pacotes a *spanning tree*, de acordo com o estado da porta.
    - A função escreve no registrador MII16 do PHY30, o valor 0x1F9F (hex). Este

valor coloca todas as portas no estado *forwarding*, e ativa a inserção da *SpecialTag*.

- *CleanMacTable:* Realiza a limpeza da tabela de MAC do *switch.* Detalhamento da função:
    - A função escreve no registrador MII0 do PHY30, o valor 0x175C (hex). Este valor realiza o *flush* do *switch*, mas mantém os dados nos registradores de configuração, apena apaga informações dinâmicas armazenadas.
- *SetPortState*: Configura estado da porta em questão, no *hardware*. Detalhamento da função:
    - A função escreve no registrador MII16 do PHY30, o valor relacionado a um estado de porta especificado pelo *spanning tree*. Os valores para atribuição na porta estão relacionados com os quadros dos registradores e os estados de porta, apresentados na Figura 61 e Figura 62 abaixo [ICP08]. Por exemplo, caso seja necessário colocar a porta um para o estado *learning*, é realizado o seguinte procedimento: lê-se o valor do registrador MII16 relacionado ao PHY30; realiza-se a operação E lógico com o complemento da máscara 0x0202 (seleção dos *bits* 16.1 e 16.11), e a operação OU lógico com a máscara 0x0200, colocando para 1 o *bit* 16.11 e para 0 o *bit* 16.1 do registrador MII16 do PHY30. Quando uma porta transita para o estado *forwarding*, é ativado em conjunto o estado *learning*. Ou seja, para passar a porta 1 para o estado *forwarding*, é lido o valor do registrador e realizado o OU lógico com a máscara 0x0202 (hex)
- *GetPortCost*: Obtém o valor do custo da porta no *hardware*. Detalhamento da função:
    - A função lê o registrador MII0 dos PHYs das portas e armazena o valor da velocidade da conexão.
- *MacHWRecovery*: Obtém o valor MAC do *switch*, utilizando a biblioteca LIBNET.
    - A função obtém o endereço MAC utilizando o nome da interface do Linux, e usando a função auxiliar *libnet_get_hwaddr*.

Através destas bibliotecas é possível manipular todas as informações necessárias em nível de *hardware*, para o funcionamento do algoritmo RSTP implementado, integrado a plataforma de desenvolvimento.

Spanning tree register

PHY	MII	ROM	R/W	Description	Default
30	16.15	--	RO	Fast mode for simulation, 1: Fast mode, 0: normal mode Default value TEST2=0 / TEST2=1 0 / Pin 66 FASTMODE (0)	*
	16.12	66.4	R/W	Learning enable 1: enable address learning capability of port 4 0: disable address learning capability of port 4	1
	16.11	66.3	R/W	Learning enable 1: enable address learning capability of port 3 0: disable address learning capability of port 3	1
	16.10	66.2	R/W	Learning enable 1: enable address learning capability of port 2 0: disable address learning capability of port 2	1
	16.9	66.1	R/W	Learning enable 1: enable address learning capability of port 1 0: disable address learning capability of port 1	1
	16.8	66.0	R/W	Learning enable 1: enable address learning capability of port 0 0: disable address learning capability of port 0	1
	16.7	65.7	R/W	Stag_en special tagging function enable If this function is enabled, IP175C/IP175CH inserts source port information in tag header. 1: enable special tagging function 0: disable special tagging function	0
	16.4	65.4	R/W	Forward enable 1: enable receiving function of port 4 0: disable receiving function of port 4	1
	16.3	65.3	R/W	Forward enable 1: enable receiving function of port 3 0: disable receiving function of port 3	1
	16.2	65.2	R/W	Forward enable 1: enable receiving function of port 2 0: disable receiving function of port 2	1
	16.1	65.1	R/W	Forward enable 1: enable receiving function of port 1 0: disable receiving function of port 1	1
	16.0	65.0	R/W	Forward enable 1: enable receiving function of port 0 0: disable receiving function of port 0	1

Figura 61 – Tabela de registradores *spanning tree*, com os estados de porta.

State	Fwd BPDU packet to CPU	Fwd BPDU packet from CPU	Address learning	Fwd all packet normally	(Forward enable, Learning enable)
Disable	X (note 2)	X (note 2)	X	X	(0,0)
Blocking	O	X (note 3)	X	X	(0,0)
Listening	O	O	X	X	(0,0)
Learning	O	O	O	X	(0,1)
Forwarding	O	O	O	O	(1,1)

Note1: O: enabled, X: disabled
Note2: CPU should not send packets to IP175C/IP175CH and should discard packets from IP175C/IP175CH.
Note3: CPU should not send packets to IP175C/IP175CH.

Figura 62 – Relação dos estados de porta *spanning tree*.

5.3 Avaliação do Algoritmo RSTP na Plataforma de Desenvolvimento

A validação do algoritmo RSTP com a plataforma de desenvolvimento será realizada usando seis plataformas de desenvolvimento juntamente com dois *switches* proprietários. Um deles, da empresa MRV, executando o sistema RSTP, e outro, da empresa NETLINK, utilizando o sistema STP. O papel do *switch* STP proprietário será prover acesso a internet as demais *bridges* da topologia. O *switch* RSTP proprietário tem a função de validar o algoritmo RSTP implementado com os equipamentos disponíveis no mercado.

As telas de interface do algoritmo nas seis placas são obtidas através de TELNET na porta *ethernet* de controle de cada placa.

A topologia utilizada se assemelha a topologia testada para o simulador. Está topologia de teste está exemplificada na Figura 63.

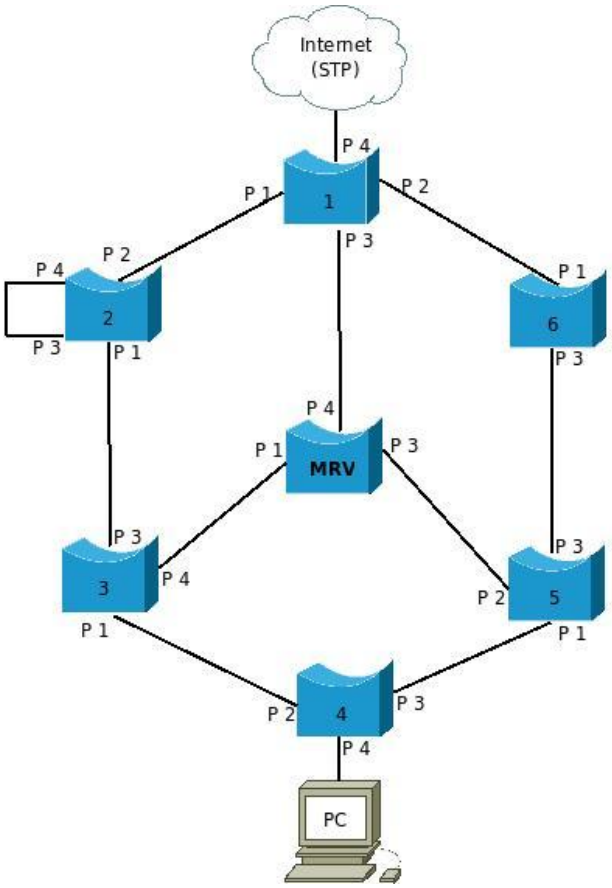


Figura 63 – Topologia de teste do *hardware*.

Este ambiente de testes é exposto nas Figuras abaixo (Figura 64, Figura 65, Figura 66), demonstrando a complexidade do sistema para executar a topologia teste proposta.

Figura 64 – Ambiente de teste (foto 1).

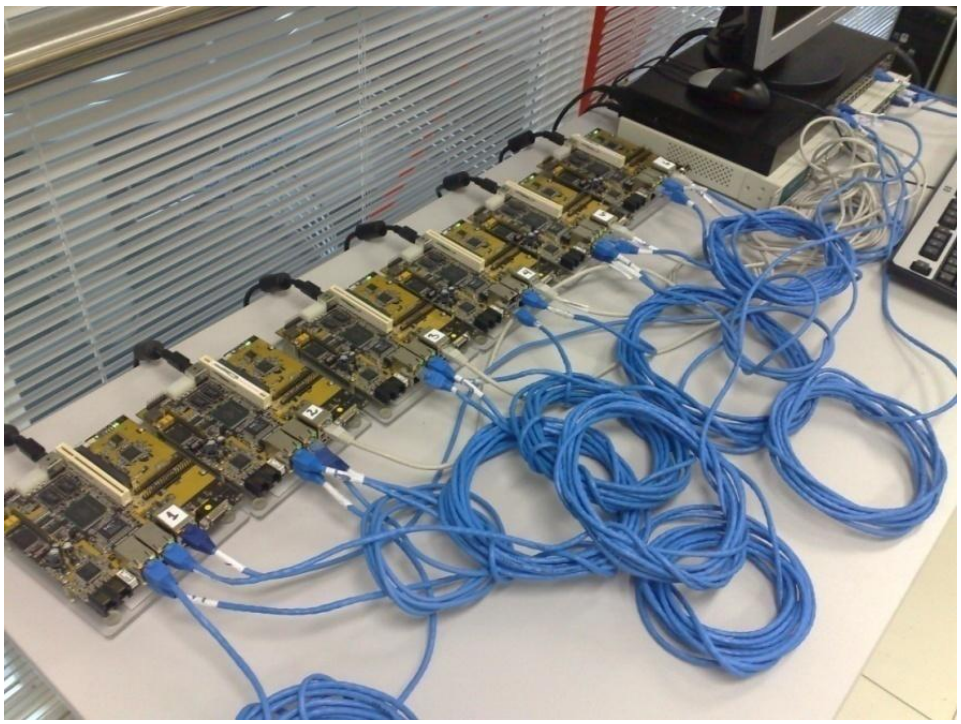


Figura 65 – Ambiente de teste (foto 2).

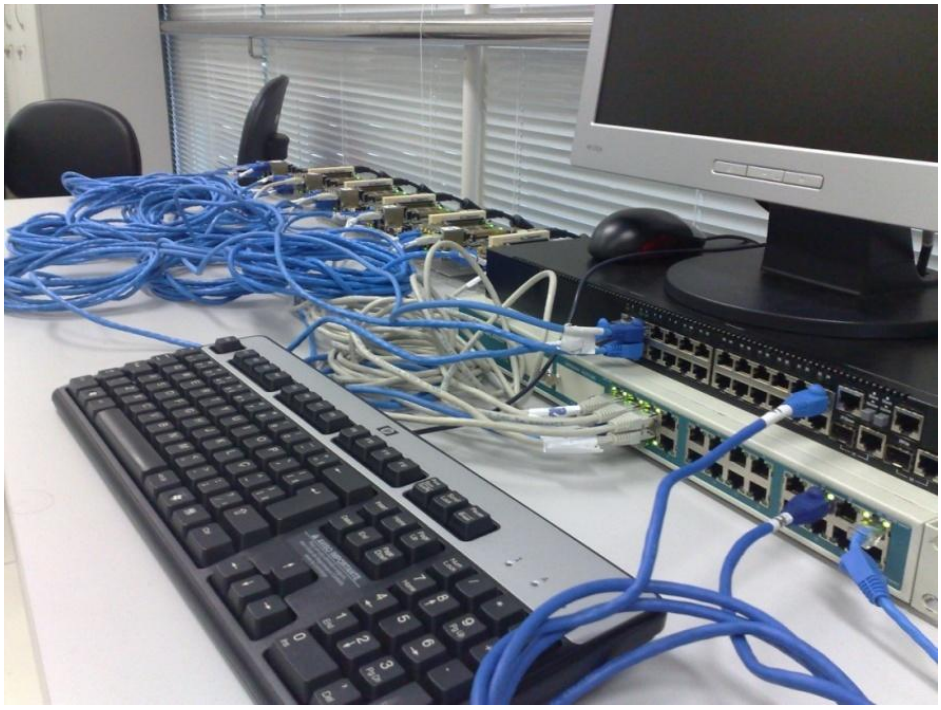


Figura 66 – Ambiente de teste (foto 3).

Ao iniciar a execução do algoritmo RSTP nas seis placas, a configuração das portas do algoritmo é apresentada na Figura 67. Está configuração inicial é a mesma para as seis placas, pois é utilizado o mesmo número de portas e mesmo custo de enlace. O que difere é a identificação de cada *bridge* executando em cada placa.

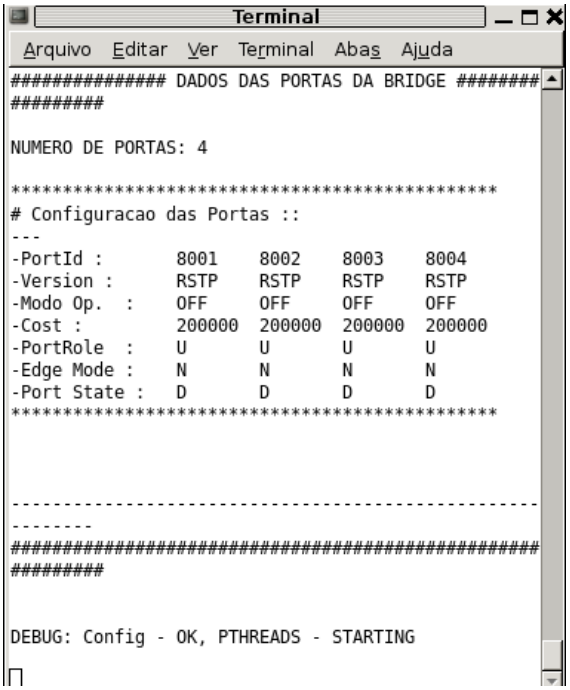
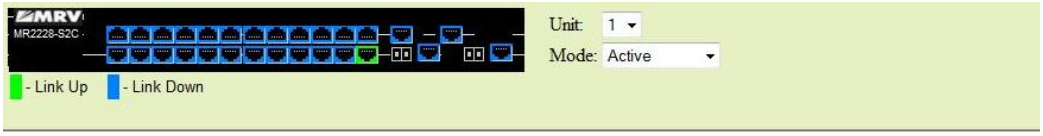


Figura 67 – Configuração inicial das portas das *bridges* nas placas.

O switch MRV apresenta as configurações iniciais apresentadas na Figura 68. Neste teste, as portas conectadas as placas referem-se às portas um, três e quatro destacadas. As outras portas são desprezadas para o entendimento do teste.



STA Port Information

Port	Spanning Tree	STA Status	Forward Transitions	Designated Cost	Designated Bridge	Designated Port	Oper Link Type	Oper Edge Port	Port Role	Trunk Member
1	Enabled	Discarding	0	100019	32768.00201A261B5B	128.1	Point-to-Point	Disabled	Disabled	
2	Enabled	Discarding	0	100019	32768.00201A261B5B	128.2	Point-to-Point	Disabled	Disabled	
3	Enabled	Discarding	0	100019	32768.00201A261B5B	128.3	Point-to-Point	Disabled	Disabled	
4	Enabled	Discarding	0	100019	32768.00201A261B5B	128.4	Point-to-Point	Disabled	Disabled	
5	Enabled	Discarding	0	100019	32768.00201A261B5B	128.5	Point-to-Point	Disabled	Disabled	
6	Enabled	Discarding	0	100019	32768.00201A261B5B	128.6	Point-to-Point	Disabled	Disabled	
7	Enabled	Discarding	0	100019	32768.00201A261B5B	128.7	Point-to-Point	Disabled	Disabled	
8	Enabled	Discarding	0	100019	32768.00201A261B5B	128.8	Point-to-Point	Disabled	Disabled	
9	Enabled	Discarding	0	100019	32768.00201A261B5B	128.9	Point-to-Point	Disabled	Disabled	
10	Enabled	Discarding	0	100019	32768.00201A261B5B	128.10	Point-to-Point	Disabled	Disabled	
11	Enabled	Discarding	0	100019	32768.00201A261B5B	128.11	Point-to-Point	Disabled	Disabled	
12	Enabled	Discarding	0	100019	32768.00201A261B5B	128.12	Point-to-Point	Disabled	Disabled	
13	Enabled	Discarding	0	100019	32768.00201A261B5B	128.13	Point-to-Point	Disabled	Disabled	
14	Enabled	Discarding	0	100019	32768.00201A261B5B	128.14	Point-to-Point	Disabled	Disabled	
15	Enabled	Discarding	0	100019	32768.00201A261B5B	128.15	Point-to-Point	Disabled	Disabled	
16	Enabled	Discarding	0	100019	32768.00201A261B5B	128.16	Point-to-Point	Disabled	Disabled	
17	Enabled	Discarding	0	100019	32768.00201A261B5B	128.17	Point-to-Point	Disabled	Disabled	
18	Enabled	Discarding	0	100019	32768.00201A261B5B	128.18	Point-to-Point	Disabled	Disabled	
19	Enabled	Discarding	0	100019	32768.00201A261B5B	128.19	Point-to-Point	Disabled	Disabled	
20	Enabled	Discarding	0	100019	32768.00201A261B5B	128.20	Point-to-Point	Disabled	Disabled	
21	Enabled	Discarding	0	100019	32768.00201A261B5B	128.21	Point-to-Point	Disabled	Disabled	
22	Enabled	Discarding	0	100019	32768.00201A261B5B	128.22	Point-to-Point	Disabled	Disabled	
23	Enabled	Discarding	0	100019	32768.00201A261B5B	128.23	Point-to-Point	Disabled	Disabled	
24	Enabled	Forwarding	1	19	32768.00051D0202BA	128.20	Point-to-Point	Disabled	Root	
25	Enabled	Discarding	0	100019	32768.00201A261B5B	128.25	Point-to-Point	Disabled	Disabled	
26	Enabled	Discarding	0	100019	32768.00201A261B5B	128.26	Point-to-Point	Disabled	Disabled	

Figura 68 – Configurações inicias do *switch* MRV.

Após a inicialização das seis *bridges* e do *switch* MRV, se inicia a troca de mensagens entre as mesmas. Com essa troca de BPDUs, a topologia converge, configurando as *bridges* e suas portas de forma a restar apenas um único caminho válido entre as *bridges*. Após esse acontecimento, resultante do algoritmo RSTP, se obtém o resultado apresentado na Figura 69, semelhante ao resultado da topologia simulada anteriormente. A diferença consiste na configuração dos enlaces do *switch* MRV, os quais estão configurados com um custo melhor (prioridade mais baixa), gerando uma convergência resultante diferente da apresentada no simulador.

A Figura 48 apresentada no Capítulo 4.2 contém a legenda para compreensão do diagrama.

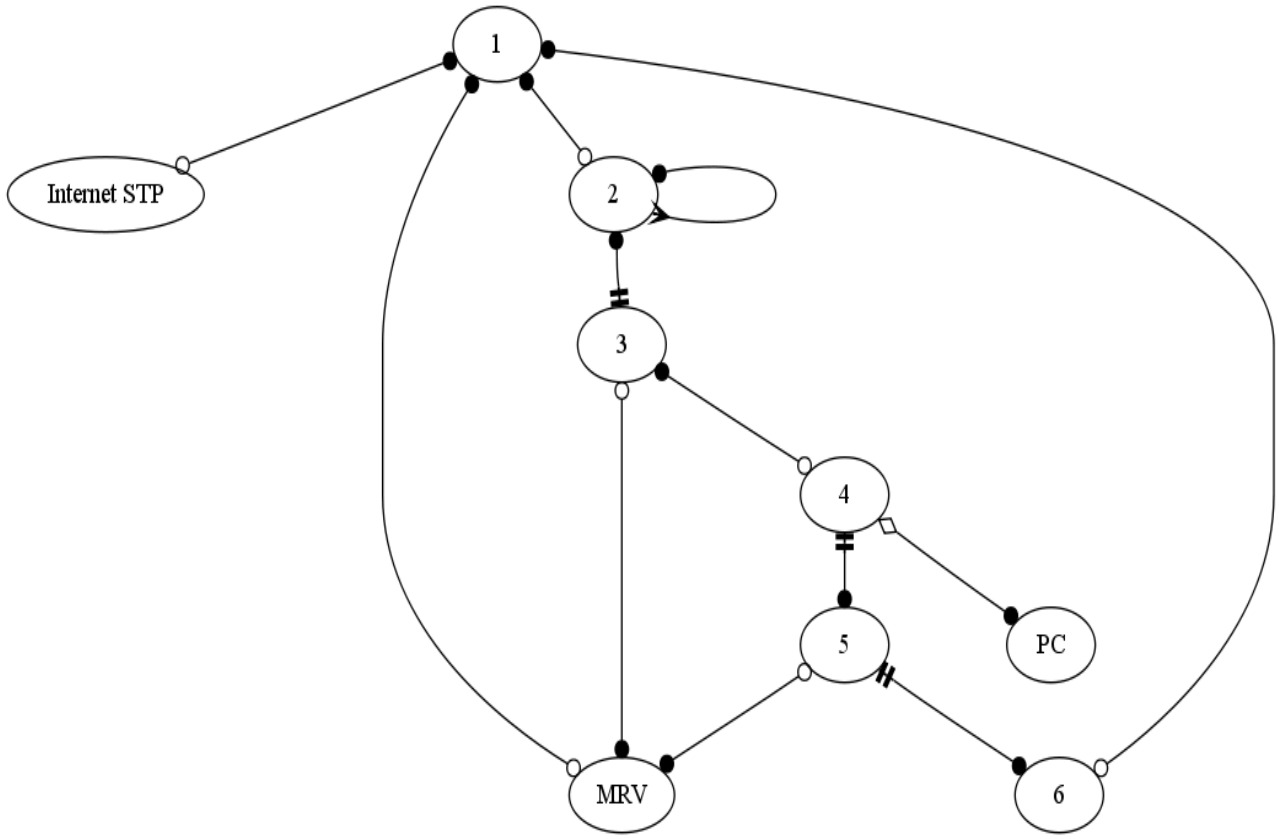
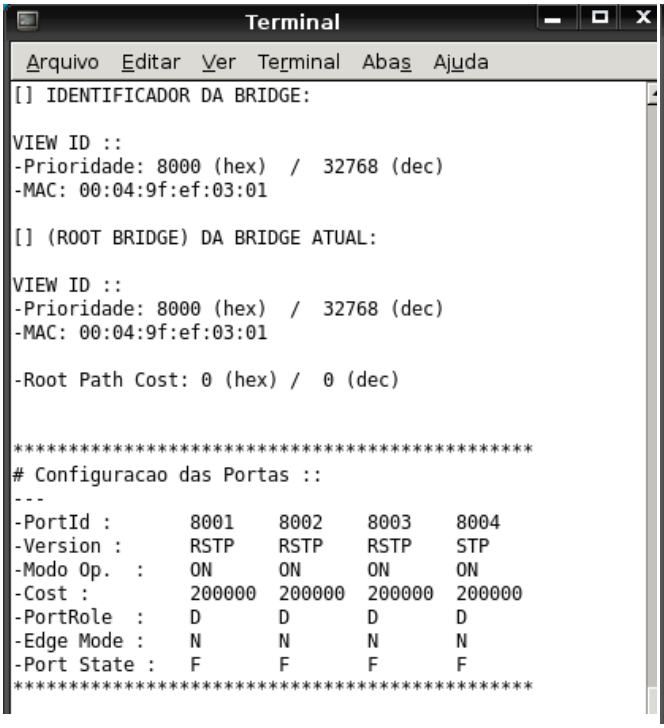
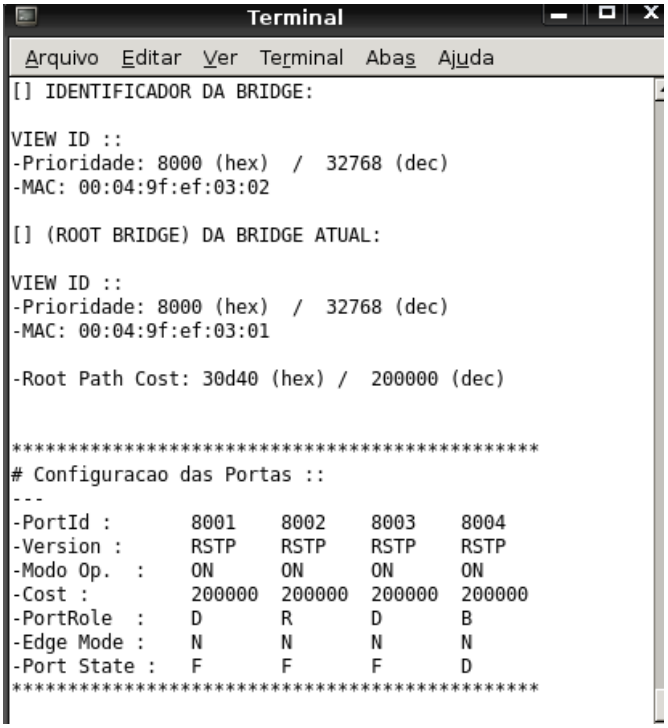


Figura 69 – Diagrama da topologia teste resultante após convergência (HW).

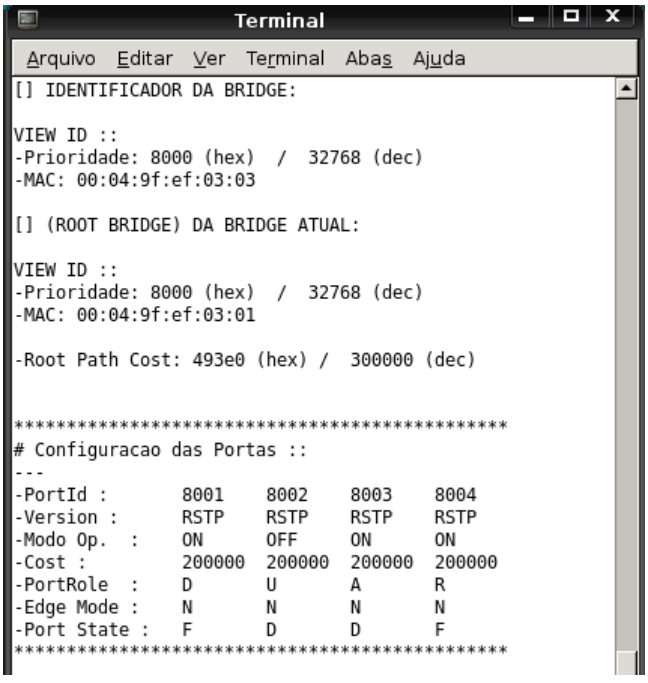
Este resultado pode ser conferido analisando as telas de interface do algoritmo, exibindo as seis *bridges* com suas portas configuradas. Esta ilustração é apresentada na Figura 70 e Figura 71.



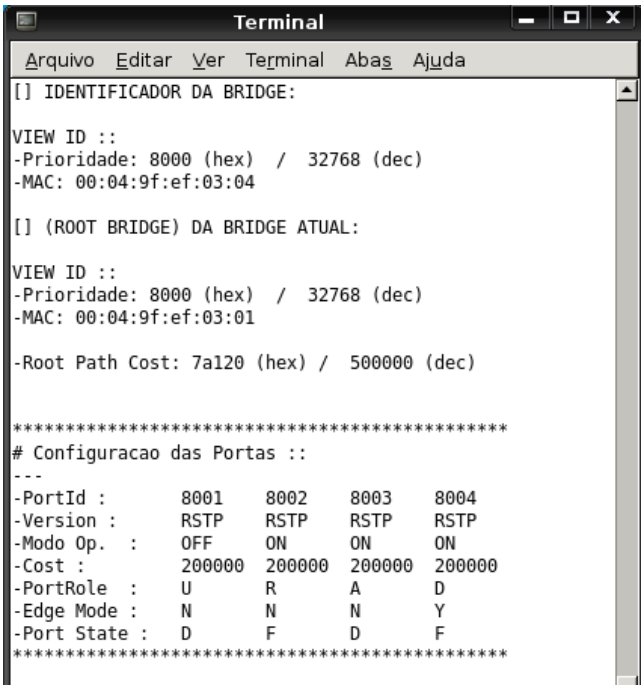
Bridge 1



Bridge 2

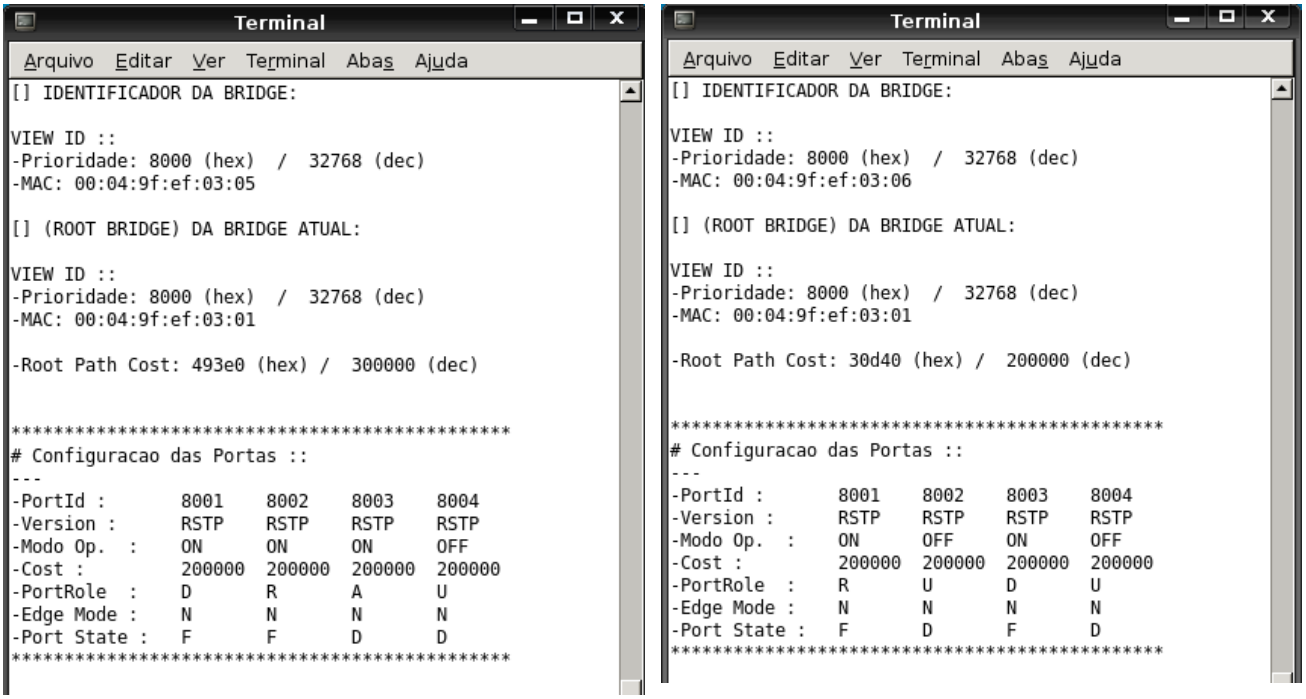


Bridge 3



Bridge 4

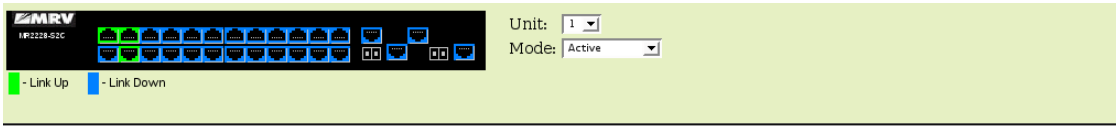
Figura 70 – Telas de interface das *bridges* 1 a 4 após convergência (HW).



Bridge 5 **Bridge 6**

Figura 71 – Telas de interface das *bridges* 5 e 6 após convergência (HW).

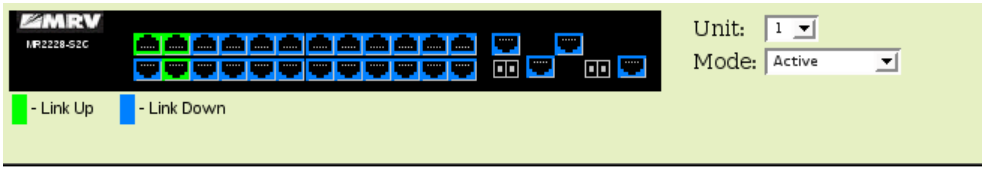
Após convergência, o *switch* MRV apresenta as seguintes configurações ilustradas na Figura 72. A Figura 73 exibe informações sobre a porta pela qual o *switch* está ligado a *root bridge*, além de exibir a identificação da *root bridge* e suas configurações.



STA Port Information

Port	Spanning Tree	STA Status	Forward Transitions	Designated Cost	Designated Bridge	Designated Port	Oper Link Type	Oper Edge Port	Port Role	Trunk Member
1	Enabled	Forwarding	3	100000	32768.00201A261B5B	128.1	Point-to-Point	Disabled	Designated	
2	Enabled	Discarding	0	100000	32768.00201A261B5B	128.2	Point-to-Point	Disabled	Disabled	
3	Enabled	Forwarding	1	100000	32768.00201A261B5B	128.3	Point-to-Point	Disabled	Designated	
4	Enabled	Forwarding	1	0	32768.00049FEF0301	128.3	Point-to-Point	Disabled	Root	
5	Enabled	Discarding	0	100000	32768.00201A261B5B	128.5	Point-to-Point	Disabled	Disabled	
6	Enabled	Discarding	0	100000	32768.00201A261B5B	128.6	Point-to-Point	Disabled	Disabled	
7	Enabled	Discarding	0	100000	32768.00201A261B5B	128.7	Point-to-Point	Disabled	Disabled	
8	Enabled	Discarding	0	100000	32768.00201A261B5B	128.8	Point-to-Point	Disabled	Disabled	
9	Enabled	Discarding	0	100000	32768.00201A261B5B	128.9	Point-to-Point	Disabled	Disabled	
10	Enabled	Discarding	0	100000	32768.00201A261B5B	128.10	Point-to-Point	Disabled	Disabled	
11	Enabled	Discarding	0	100000	32768.00201A261B5B	128.11	Point-to-Point	Disabled	Disabled	
12	Enabled	Discarding	0	100000	32768.00201A261B5B	128.12	Point-to-Point	Disabled	Disabled	
13	Enabled	Discarding	0	100000	32768.00201A261B5B	128.13	Point-to-Point	Disabled	Disabled	
14	Enabled	Discarding	0	100000	32768.00201A261B5B	128.14	Point-to-Point	Disabled	Disabled	
15	Enabled	Discarding	0	100000	32768.00201A261B5B	128.15	Point-to-Point	Disabled	Disabled	
16	Enabled	Discarding	0	100000	32768.00201A261B5B	128.16	Point-to-Point	Disabled	Disabled	
17	Enabled	Discarding	0	100000	32768.00201A261B5B	128.17	Point-to-Point	Disabled	Disabled	
18	Enabled	Discarding	0	100000	32768.00201A261B5B	128.18	Point-to-Point	Disabled	Disabled	
19	Enabled	Discarding	0	100000	32768.00201A261B5B	128.19	Point-to-Point	Disabled	Disabled	
20	Enabled	Discarding	0	100000	32768.00201A261B5B	128.20	Point-to-Point	Disabled	Disabled	
21	Enabled	Discarding	0	100000	32768.00201A261B5B	128.21	Point-to-Point	Disabled	Disabled	
22	Enabled	Discarding	0	100000	32768.00201A261B5B	128.22	Point-to-Point	Disabled	Disabled	
23	Enabled	Discarding	0	100000	32768.00201A261B5B	128.23	Point-to-Point	Disabled	Disabled	
24	Enabled	Discarding	0	100000	32768.00201A261B5B	128.24	Point-to-Point	Disabled	Disabled	
25	Enabled	Discarding	0	100000	32768.00201A261B5B	128.25	Point-to-Point	Disabled	Disabled	
26	Enabled	Discarding	0	100000	32768.00201A261B5B	128.26	Point-to-Point	Disabled	Disabled	

Figura 72 – Configuração das portas do *switch* MRV após convergência.



STA Information

Spanning Tree:

Spanning Tree State	Enabled	Designated Root	32768.00049FEF0301
Bridge ID	32768.00201A261B5B	Root Port	1
Max Age	20	Root Path Cost	500000
Hello Time	2	Configuration Changes	2
Forward Delay	15	Last Topology Change	0 d 0 h 0 min 49 s

Figura 73 – Informações de configurações atuais e porta *root* do *switch* MRV.

Após a convergência das *bridges*, é proposta a remoção de um enlace principal da topologia, como demonstrado no *Simulador de Conexões*, pelo qual o caminho até a *root bridge* é realizado. Neste caso, o enlace principal passa pela porta quatro do *switch* MRV, que será retirado. A Figura 74 ilustra esta alteração.

A modificação tem o propósito de causar o maior *stress* possível na topologia resultante, obrigando o algoritmo a refazer os caminhos lógicos até a *root bridge*. Esta alteração demonstra um funcionamento robusto do algoritmo implementado, interagindo corretamente com um equipamento proprietário.

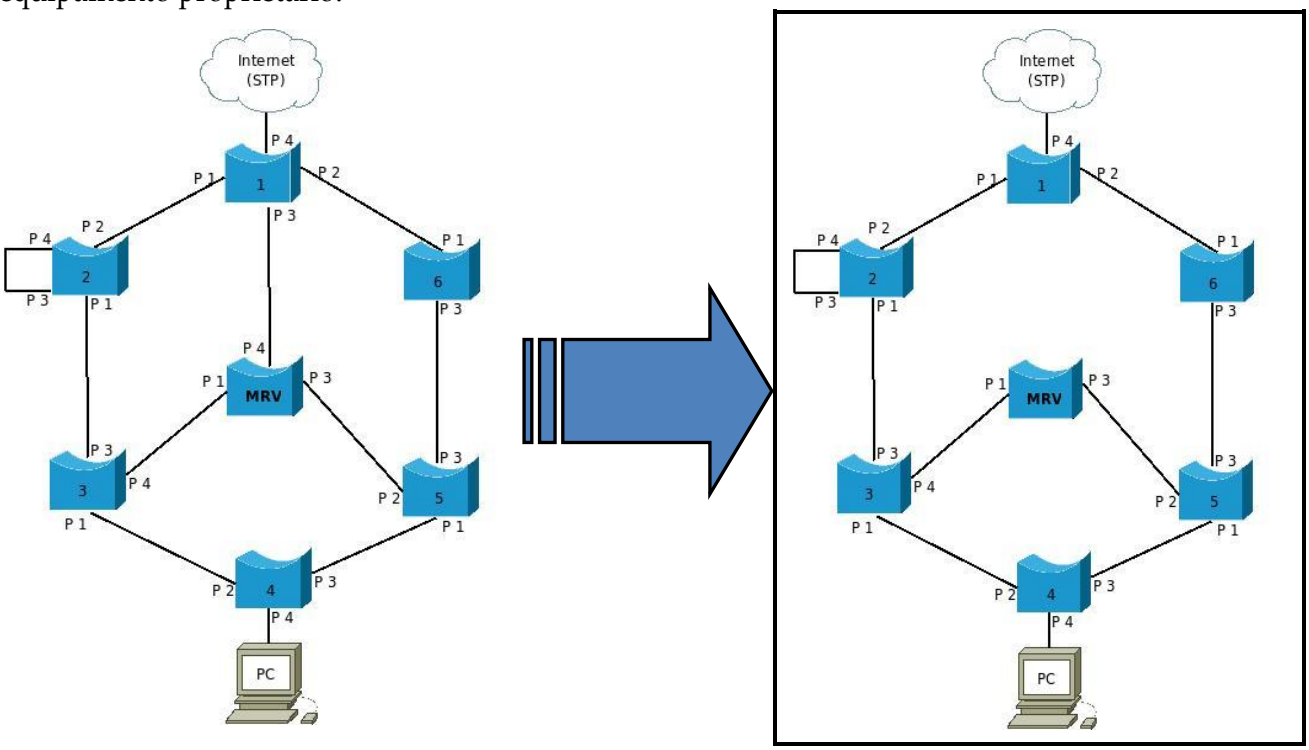


Figura 74 – Remoção do enlace da porta 4 do *switch* MRV.

Removendo o enlace do *switch* MRV, o algoritmo RSTP começa a refazer a convergência com esta nova situação, devido a esta alteração crítica no caminho até a *root bridge*. Assim, a Figura 75 mostra o diagrama da topologia convergida após a alteração. Esta alteração foi efetuada da mesma

maneira no simulador, mas os resultados foram diferentes. Como mencionado anteriormente, no caso do simulador, os enlaces das *bridges* tem o mesmo custo, já no caso do *hardware*, o *switch* MRV possui a configuração do custo dos enlaces menores que os custos aplicados as seis placas. Assim, a topologia resultante é diferente da simulada anteriormente.

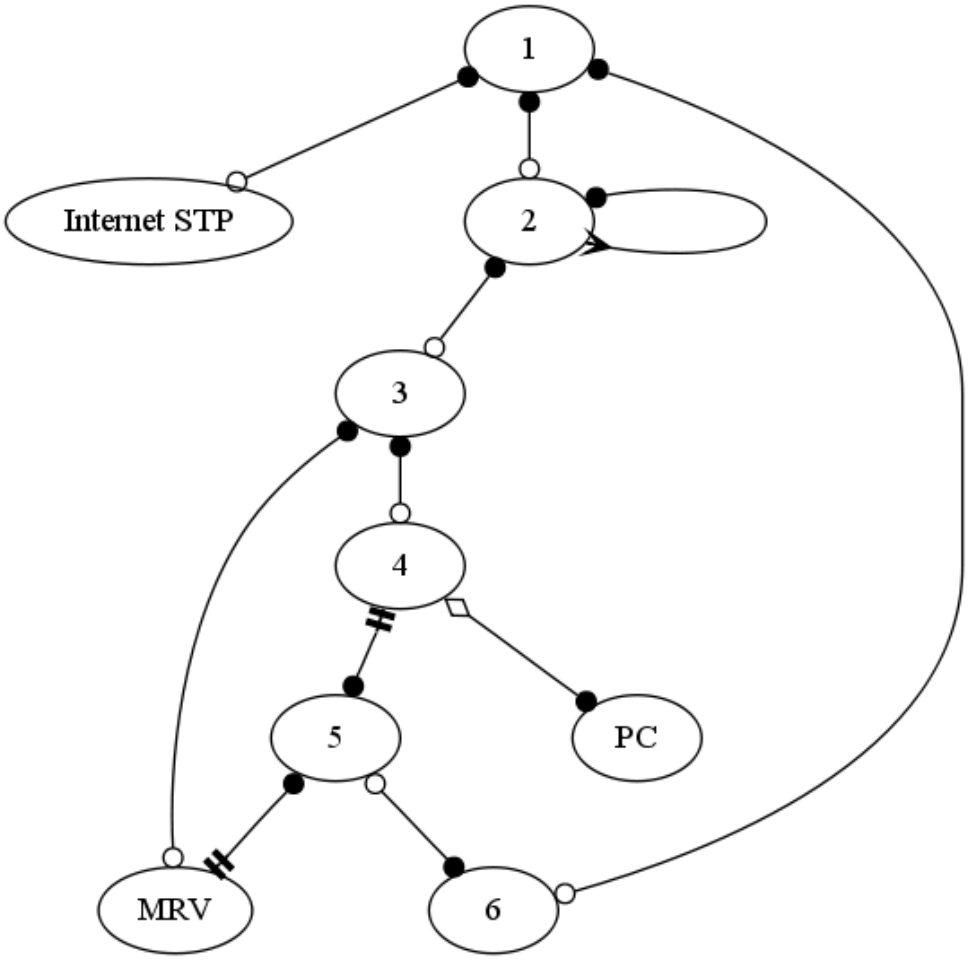
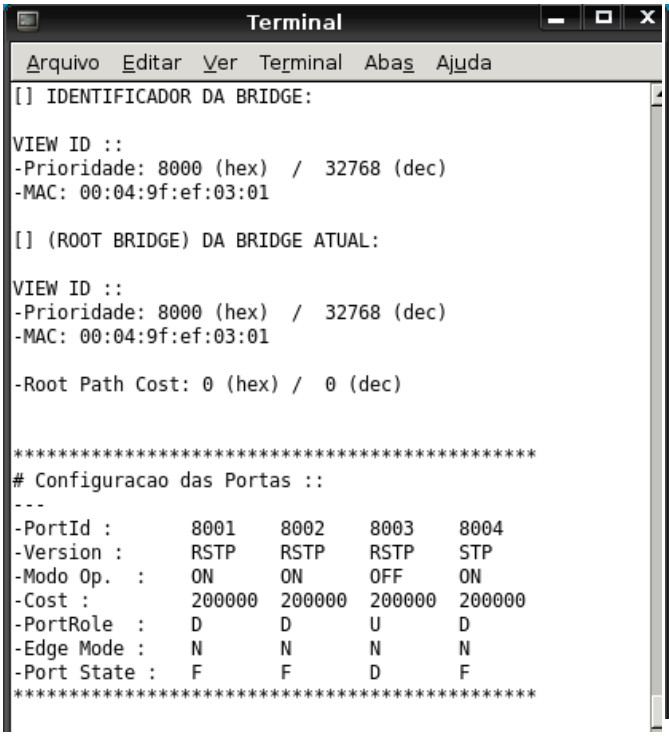
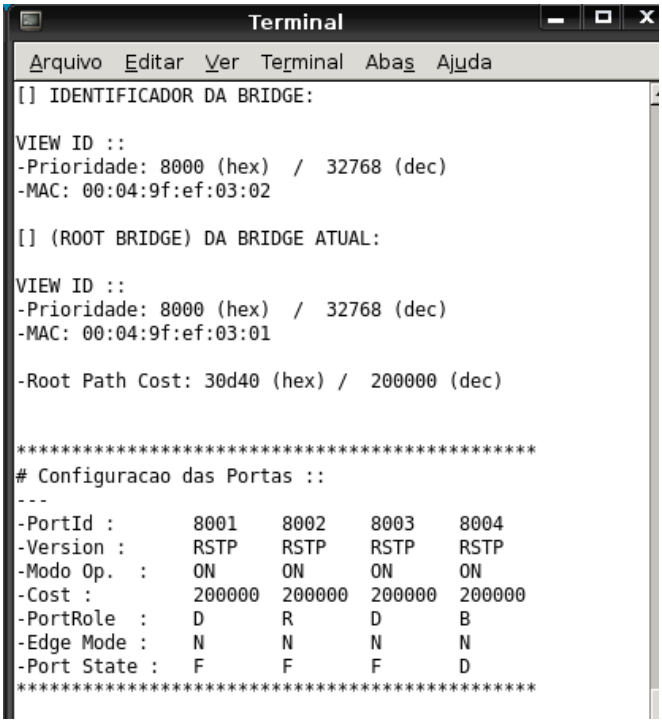


Figura 75 – Diagrama da topologia convergida após alteração (HW).

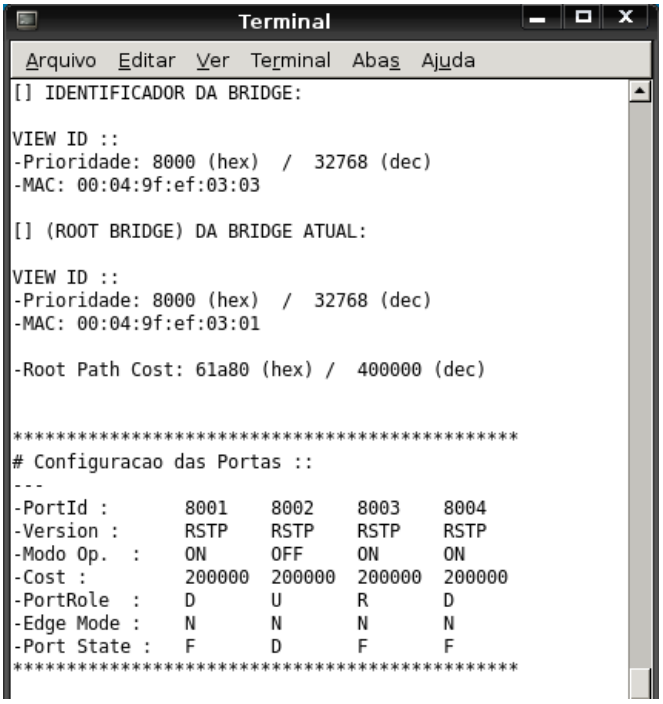
Esta modificação é exibida nas telas de interface do algoritmo, mostrando as *bridges* com suas respectivas portas configuradas. As telas de interface estão expostas na Figura 76 e Figura 77.



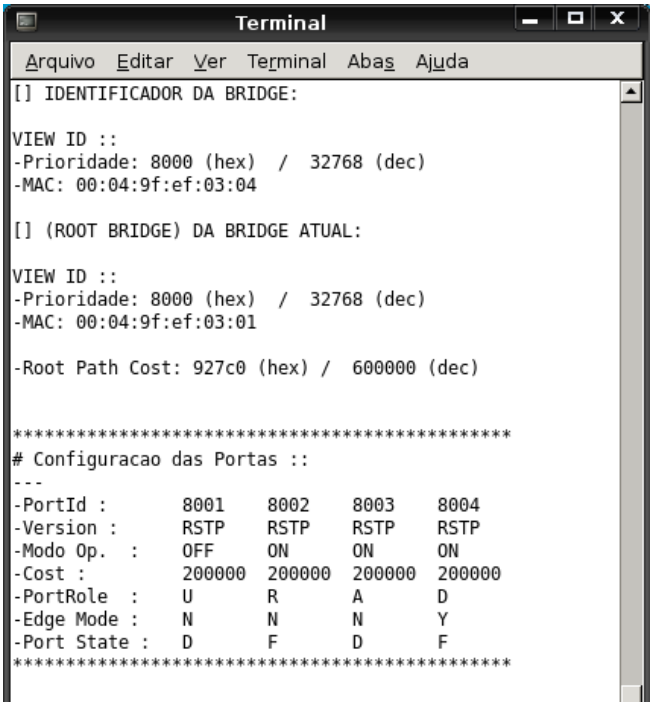
Bridge 1



Bridge 2

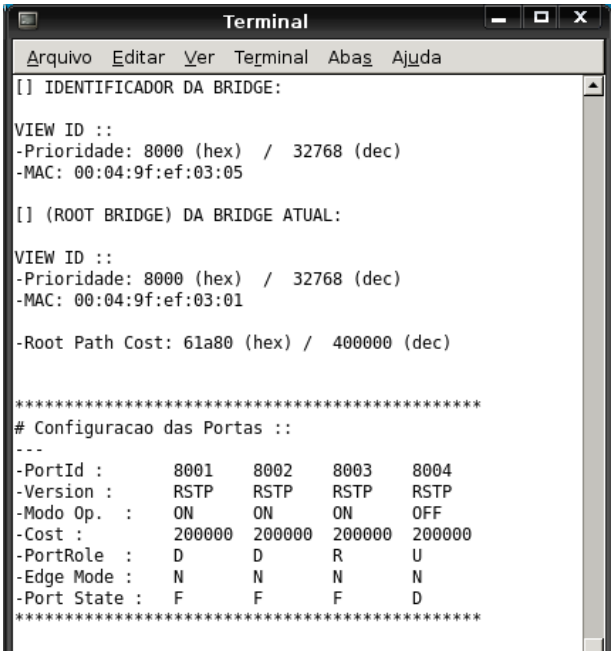


Bridge 3

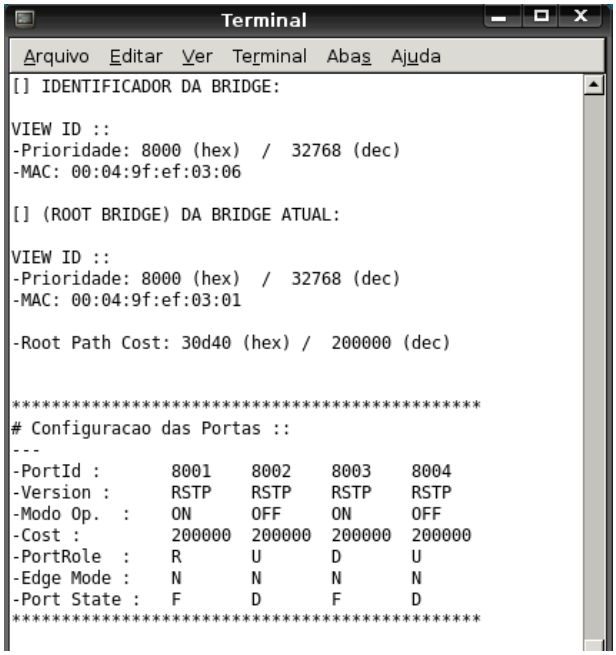


Bridge 4

Figura 76 – Telas de interface das *bridges* 1 a 4 após alteração (HW).



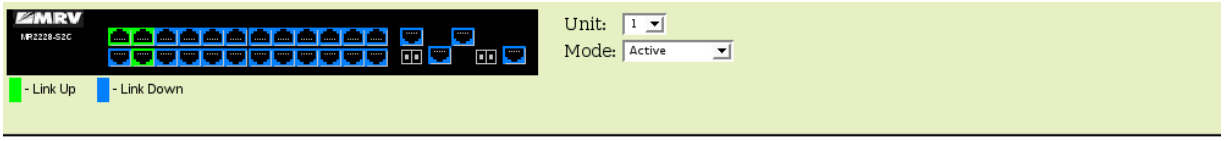
Bridge 5



Bridge 6

Figura 77 – Telas de interface das *bridges* 5 e 6 após alteração (HW).

Já que o enlace retirado estava conectado ao *switch MRV* na sua porta quatro, este altera suas configurações, como mostrado na Figura 78.



STA Port Information

Port	Spanning Tree	STA Status	Forward Transitions	Designated Cost	Designated Bridge	Designated Port	Oper Link Type	Oper Edge Port	Port Role	Trunk Member
1	Enabled	Forwarding	3	400000	32768.00049FEF0303	128.4	Point-to-Point	Disabled	Root	
2	Enabled	Discarding	0	500000	32768.00201A261B5B	128.2	Point-to-Point	Disabled	Disabled	
3	Enabled	Discarding	1	400000	32768.00049FEF0305	128.2	Point-to-Point	Disabled	Alternate	
4	Enabled	Discarding	1	500000	32768.00201A261B5B	128.4	Point-to-Point	Disabled	Disabled	
5	Enabled	Discarding	0	500000	32768.00201A261B5B	128.5	Point-to-Point	Disabled	Disabled	
6	Enabled	Discarding	0	500000	32768.00201A261B5B	128.6	Point-to-Point	Disabled	Disabled	
7	Enabled	Discarding	0	500000	32768.00201A261B5B	128.7	Point-to-Point	Disabled	Disabled	
8	Enabled	Discarding	0	500000	32768.00201A261B5B	128.8	Point-to-Point	Disabled	Disabled	
9	Enabled	Discarding	0	500000	32768.00201A261B5B	128.9	Point-to-Point	Disabled	Disabled	
10	Enabled	Discarding	0	500000	32768.00201A261B5B	128.10	Point-to-Point	Disabled	Disabled	
11	Enabled	Discarding	0	500000	32768.00201A261B5B	128.11	Point-to-Point	Disabled	Disabled	
12	Enabled	Discarding	0	500000	32768.00201A261B5B	128.12	Point-to-Point	Disabled	Disabled	
13	Enabled	Discarding	0	500000	32768.00201A261B5B	128.13	Point-to-Point	Disabled	Disabled	
14	Enabled	Discarding	0	500000	32768.00201A261B5B	128.14	Point-to-Point	Disabled	Disabled	
15	Enabled	Discarding	0	500000	32768.00201A261B5B	128.15	Point-to-Point	Disabled	Disabled	
16	Enabled	Discarding	0	500000	32768.00201A261B5B	128.16	Point-to-Point	Disabled	Disabled	
17	Enabled	Discarding	0	500000	32768.00201A261B5B	128.17	Point-to-Point	Disabled	Disabled	
18	Enabled	Discarding	0	500000	32768.00201A261B5B	128.18	Point-to-Point	Disabled	Disabled	
19	Enabled	Discarding	0	500000	32768.00201A261B5B	128.19	Point-to-Point	Disabled	Disabled	
20	Enabled	Discarding	0	500000	32768.00201A261B5B	128.20	Point-to-Point	Disabled	Disabled	
21	Enabled	Discarding	0	500000	32768.00201A261B5B	128.21	Point-to-Point	Disabled	Disabled	
22	Enabled	Discarding	0	500000	32768.00201A261B5B	128.22	Point-to-Point	Disabled	Disabled	
23	Enabled	Discarding	0	500000	32768.00201A261B5B	128.23	Point-to-Point	Disabled	Disabled	
24	Enabled	Discarding	0	500000	32768.00201A261B5B	128.24	Point-to-Point	Disabled	Disabled	
25	Enabled	Discarding	0	500000	32768.00201A261B5B	128.25	Point-to-Point	Disabled	Disabled	
26	Enabled	Discarding	0	500000	32768.00201A261B5B	128.26	Point-to-Point	Disabled	Disabled	

Figura 78 – Tela de configuração do *switch* MRV após alteração no enlace.

6. CONCLUSÃO E TRABALHOS FUTUROS

O presente Trabalho de Conclusão apresentou um estudo do protocolo RSTP. Para compreensão do protocolo, foi utilizada documentação presente das normas técnicas e de fabricantes. O estudo dos padrões, a posterior proposta de solução a partir das análises feitas e a implementação de tal solução, aprimoraram conhecimentos em várias áreas relacionadas à engenharia. Destas, as de maior influência sobre o trabalho são as áreas de algoritmos, sistemas embarcados, telecomunicações e sistemas operacionais, tornando o presente trabalho multidisciplinar e de fundamental importância para a solidificação dos conhecimentos adquiridos durante o curso de Engenharia de Computação.

A implementação do protocolo RSTP como módulo integrante do LINUX foi finalizada com sucesso, atingindo os objetivos propostos. A experiência do desenvolvimento de um sistema embarcado, em parceria com uma empresa (PARKS) e com trabalho em grupo enriqueceram este Trabalho de Conclusão, por representar uma situação real de trabalho de Engenharia, com metas, prazos e restrições a atender.

Os trabalhos futuros compreendem: (*i*) porte do algoritmo para equipamento da empresa PARKS; (*ii*) otimização do algoritmo; (iii) realização de testes complexos a fim de validar um funcionamento completo do sistema.

O porte do algoritmo para equipamento da empresa PARKS se faz necessário devido ao modo que o protocolo foi implementado. O algoritmo foi desenvolvido atrelado ao funcionamento da plataforma de desenvolvimento utilizada. Assim, para realizar a instalação do algoritmo implementado em equipamentos da empresa PARKS, ajustes e configurações precisam ser feitos na parte de integração do algoritmo com o *hardware*.

A funcionalidade do protocolo RSTP foi implementada por completo, mas otimizações diversas, possíveis de serem feitas em seções de configuração do algoritmo, poderiam ser realizadas com mais tempo de projeto. Estas otimizações no algoritmo permitiriam aumentar a velocidade de convergência, além de aumentar sua tolerância a falhas.

Concluindo, após o porte do algoritmo para a plataforma da empresa, e a realização das otimizações do algoritmo, testes exaustivos serão realizados com o objetivo de certificar a implementação realizada, confirmando a funcionalidade do protocolo com o equipamento proprietário da empresa PARKS.

REFERÊNCIAS

[RED09] Wikipedia. “Rede de Computadores“. Capturado em: http://pt.wikipedia.org/wiki/Rede_de_computadores, Setembro 2009.

[CIS09] Cisco Systems Inc. “*Understanding Rapid Spanning Tree Protocol (802.1w)“*. Capturado em: http://www.cisco.com/en/US/tech/tk389/tk621/technologies_white_paper09186a0080094cfa.shtml, Setembro 2009.

[INS98] Institute of Eletrical and Electronics Engineers, Inc. “*Media Access Control (MAC) Bridges*”, Std 802.1d, 1998 Edition, pp 58 – 113.

[INS04] Institute of Eletrical and Electronics Engineers, Inc. “*Media Access Control (MAC) Bridges*”, Std 802.1d, 2004 Edition, pp 57 – 65.

[INS01] Institute of Eletrical and Electronics Engineers, Inc. “*Media Access Control (MAC) Bridges*”, Std 802.1w, 2001 Edition, pp 10 – 18, pp 24 – 76.

[SPA09] Wikipedia. “Spanning Tree Protocol”. Capturado em: http://en.wikipedia.org/wiki/Spanning_tree_protocol, Setembro 2009.

[WOJ03] Wojdak,W. “*Rapid Spanning Tree Protocol: A New Solution From a Old Technology*”. Capturado em: http://www.compactpci-systems.com/dl.php?pdf=/pdfs/PerfTech.Mar03.pdf , Setembro 2009.

[ICP08] IC Plus, Corp. “*Data Sheet - Ethernet Integrated Switch*”, 2008 Edition.

[ICP06] IC Plus, Corp. “*Special Tag Application*”, 2006 Edition.

[ETH09] Wikipedia. “Ethernet”. Capturado em: http://pt.wikipedia.org/wiki/Ethernet, Dezembro 2009.