

**ALOCAÇÃO DE ENDEREÇOS IPV6 EM REDES
MULTI-HOP DE RÁDIOS DE BAIXA POTÊNCIA**

BRUNA SOARES PERES

**ALOCAÇÃO DE ENDEREÇOS IPV6 EM REDES
MULTI-HOP DE RÁDIOS DE BAIXA POTÊNCIA**

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR: OLGA NIKOLAEVNA GOUSSEVSKAIA

Belo Horizonte
Fevereiro de 2015

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Peres, Bruna Soares.

P437a Alocação de endereços IPv6 em redes multi-hop de
Rádios de baixa potência / Bruna Soares Peres —
Belo Horizonte, 2015.

xx, 73 f. : il. ; 29cm.

Dissertação (mestrado) - Universidade Federal de
Minas Gerais – Departamento de Ciência da
Computação.

Orientadora: Olga Nikolaevna Goussevskaja

1. Computação - Teses. 2. Redes de sensores sem
fio - Teses. 3. Redes de computadores – Protocolos-
4. Roteamento (Administração de redes de
computadores). I. Orientadora. II. Título.

CDU 519.6*22.(043)




UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

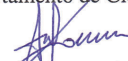
FOLHA DE APROVAÇÃO

Alocação de endereços IPv6 em redes sem fio multi-hop de baixa potência.

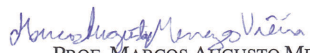
BRUNA SOARES PERES

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:


PROFA. OLGA NIKOLAEVNA GOUSSEVSKAIA - Orientadora
Departamento de Ciência da Computação - UFMG


PROF. ANTONIO ALFREDO FERREIRA LOUREIRO
Departamento de Ciência da Computação - UFMG


PROF. ÍTALO FERNANDO SCOTÁ CUNHA
Departamento de Ciência da Computação - UFMG


PROF. MARCOS AÚGUSTO MENEZES VIEIRA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 23 de fevereiro de 2015.

Agradecimentos

Agradeço primeiramente a minha família pelo apoio durante esses dois anos. A meus amigos e colegas, que me ensinaram preciosas lições e me proporcionaram excelentes momentos. À Olga, por toda dedicação e atenção e pelos valiosos conselhos. Agradeço também à FAPEMIG pelo financiamento deste projeto. Por fim, agradeço a todos os professores que fizeram parte dessa minha conquista.

Resumo

Muitas redes sem fio de baixa potência são baseadas em mecanismos que mantêm topologias acíclicas para suportar aplicações de coleta de dados. A rede é tipicamente otimizada para o tráfego ascendente de dados (dos nós à raiz), e eventuais pacotes que precisam ser enviados em uma direção diferente devem seguir por caminhos mais longos e são frequentemente descartados devido à falta de memória para as tabelas de roteamento necessárias, o que pode exigir $O(n)$ espaço de memória, onde n é o tamanho da sub-árvore com raiz em cada nó. Neste trabalho propomos uma estratégia para redes *multi-hop* que explora estruturas de rede acíclicas em redes sem fio de baixa potência para gerar e atribuir endereços IPv6 para os nós. O objetivo é permitir o roteamento descendente de maneira eficiente e robusta, com baixo consumo de memória. Isso é alcançado ao gerar e atribuir endereços IPv6 que refletem a topologia da rede sem fio. Como resultado, as tabelas de roteamento para tráfego descendente de dados passam a ter tamanho $O(k)$, onde k é o número de filhos de cada nó. Nós implementamos a estratégia proposta como uma rotina do protocolo RPL, usando o sistema operacional Contiki OS. Foram realizadas experimentos para demonstrar que a estratégia é eficiente em termos de tempo e número de mensagens, além de ser robusta à dinâmica da rede, decorrente de mudanças dos pais preferenciais e falhas em conexões e nós da rede.

Palavras-chave: IPv6, Redes sem Fio, Redes de Baixa Potência, Protocolo de Roteamento.

Abstract

Many low-power wireless networks are based on mechanisms that maintain cycle-free topologies to support data-collection applications. The network is typically optimized for bottom-up data traffic, and eventual packets that need to be sent in a different direction must follow longer paths and are frequently dropped due to lack of memory for the necessary routing tables, which can require $O(n)$ memory space, where n is the size of the sub-tree rooted at each routing node. In this work we propose MHCL: a Multihop Host Configuration strategy that explores cycle-free network structures in low-power wireless networks to generate and assign IPv6 addresses to nodes. The objective is to enable efficient and robust top-down routing with low memory footprint. MHCL generates and assigns IPv6 addresses which reflect the topology of the underlying wireless network. As a result, routing tables for top-down data traffic are of size $O(k)$, where k is the number of children of each routing node. We implemented our strategy as a subroutine of RPL protocol in Contiki OS. We performed experiments to show that our strategy is efficient in time and number of messages, and is robust to network dynamics, caused by changes in preferred parents and failures in network links and nodes.

Keywords: IPv6, Wireless Network, Low-Power Network, Routing Protocol.

Lista de Figuras

2.1	Pacote da mensagem DIO.	10
2.2	Pacote da mensagem DAO.	11
2.3	Aresta $e = (u, v)$ e as portas i e j que ela conecta. Fonte: [28]	12
4.1	Exemplos de particionamento do espaço de endereçamento da estratégia Top-down.	29
4.2	Exemplos de particionamento do espaço de endereçamento da estratégia Bottom-up.	30
4.3	Grafo 1 - Prefixo Base 1	31
4.4	Grafo 2 - Prefixo Base 1	32
4.5	Grafo 1 - Estática	33
4.6	Grafo 2 - Estática	34
4.7	Estrutura da Mensagem DIO MHCL (esquerda) e DAO MHCL (direita).	37
5.1	Tempo de inicialização em função da altura do DAG.	48
5.2	Tempo de inicialização em função da altura do DAG - Topologia Regular - Falha de Transmissão.	48
5.3	Tempo de inicialização em função da altura do DAG - Topologia Regular - Falha no Recebimento.	49
5.4	Tempo de inicialização em função da altura do DAG - Topologia Uniforme - Falha de Transmissão.	49
5.5	Tempo de inicialização em função da altura do DAG - Topologia Uniforme - Falha no Recebimento.	50
5.6	Número de DIOS em função do número de nós.	51
5.7	Número de DIOS em função do número de nós - Topologia Regular - Falha de Transmissão.	51
5.8	Número de DIOS em função do número de nós - Topologia Regular - Falha no Recebimento.	52

5.9	Número de DIOs em função do número de nós - Topologia Uniforme - Falha de Transmissão.	52
5.10	Número de DIOs em função do número de nós - Topologia Uniforme - Falha no Recebimento.	53
5.11	Número de DAOs em função do número de nós.	53
5.12	Número de DAOs em função do número de nós - Topologia Regular - Falha de Transmissão.	54
5.13	Número de DAOs em função do número de nós - Topologia Regular - Falha no Recebimento.	54
5.14	Número de DAOs em função do número de nós - Topologia Uniforme - Falha de Transmissão.	55
5.15	Número de DAOs em função do número de nós - Topologia Uniforme - Falha no Recebimento.	55
5.16	Sucesso de endereçamento em função do número de nós.	56
5.17	Sucesso de endereçamento em função do número de nós - Topologia Regular - Falha de Transmissão.	56
5.18	Sucesso de endereçamento em função do número de nós - Topologia Regular - Falha no Recebimento.	57
5.19	Sucesso de endereçamento em função do número de nós - Topologia Uniforme - Falha de Transmissão.	57
5.20	Sucesso de endereçamento em função do número de nós - Topologia Uniforme - Falha no Recebimento.	58
5.21	Sucesso de entrega de mensagens de aplicação ascendentes em função do número de nós.	59
5.22	Sucesso de entrega de mensagens de aplicação ascendentes em função do número de nós - Topologia Regular - Falha de Transmissão.	59
5.23	Sucesso de entrega de mensagens de aplicação ascendentes em função do número de nós - Topologia Regular - Falha no Recebimento.	60
5.24	Sucesso de entrega de mensagens de aplicação ascendentes em função do número de nós - Topologia Uniforme - Falha de Transmissão.	60
5.25	Sucesso de entrega de mensagens de aplicação ascendentes em função do número de nós - Topologia Uniforme - Falha no Recebimento.	61
5.26	Sucesso de entrega de mensagens de aplicação descendentes em função do número de nós.	61
5.27	Sucesso de entrega de mensagens de aplicação descendentes em função do número de nós - Topologia Regular - Falha de Transmissão.	62

5.28	Sucesso de entrega de mensagens de aplicação descendentes em função do número de nós - Topologia Regular - Falha no Recebimento.	62
5.29	Sucesso de entrega de mensagens de aplicação descendentes em função do número de nós - Topologia Uniforme - Falha de Transmissão.	63
5.30	Sucesso de entrega de mensagens de aplicação descendentes em função do número de nós - Topologia Uniforme - Falha no Recebimento.	63

Lista de Tabelas

4.1	Comparação entre as estratégias de endereçamento	36
5.1	Valor dos parâmetros de estabilização nos algoritmos	46
5.2	Teste-t para os valores que apresentaram sobreposição de intervalo de confiança mas a média de um não estava inclusa no intervalo de outro.	58

Sumário

Agradecimentos	vii
Resumo	ix
Abstract	xi
Lista de Figuras	xiii
Lista de Tabelas	xvii
1 Introdução	1
1.1 Internet das Coisas	1
1.2 Redes sem fio de baixa potência (<i>Low Power and Lossy Networks, LLNs</i>)	2
1.3 Motivação	2
1.4 Solução Proposta	3
1.5 Objetivos	5
1.6 Contribuições	5
1.7 Organização do Trabalho	6
2 Fundamentação Teórica	7
2.1 Sistemas 6LoWPAN	7
2.1.1 Low-Power and Lossy Networks (LLNs)	7
2.1.2 Endereçamento IPv6	8
2.1.3 Protocolo RPL	9
2.2 Modelos de Computação Distribuída	12
2.2.1 Componentes do Sistema	12
2.2.2 Modelo Síncrono	13
2.2.3 Modelo Assíncrono	13
2.2.4 Modelo com Falhas	13

2.2.5	Terminologia de Grafos	15
3	Trabalhos Relacionados	17
3.1	IPv6 e 6LoWPAN	17
3.2	Protocolos de Roteamento	19
3.3	Função de Agregação	21
3.4	Trickle	22
3.5	Técnicas de Alocação de Endereços em Redes	24
4	MHCL: Alocação de Endereços IPv6 em Redes Multi-hop de Rádios de Baixa Potência	27
4.1	Alocação de Endereços IPv6	28
4.1.1	Top-down	28
4.1.2	Bottom-up	29
4.1.3	Outras Estratégias	30
4.1.4	IPv6 - Formato EUI-64	34
4.2	Mensagens	36
4.3	Comunicação e Temporizadores	37
4.4	Tabelas de Roteamento e Encaminhamento	39
4.5	Análise de Complexidade	42
5	Resultados Experimentais	45
5.1	Configurações da Emulação	45
5.2	Inicialização da Rede	46
5.2.1	Tempo de Inicialização	47
5.2.2	Número de Mensagens	48
5.3	Sucesso de Endereçamento	51
5.4	Sucesso de Entrega de Mensagens de Aplicação	54
6	Conclusões e Trabalhos Futuros	65
6.1	Conclusões	65
6.2	Trabalhos Futuros	66
	Referências Bibliográficas	69

Capítulo 1

Introdução

1.1 Internet das Coisas

A Internet das Coisas é um novo paradigma que vem ganhando destaque recentemente na área de redes sem fio. Sua proposta é conectar qualquer coisa à Internet para que objetos comuniquem entre si e entre usuários, com o propósito de gerar e processar informação. Desafios no projeto de sistemas para a Internet das Coisas inclui o desenvolvimento de novos protocolos capazes de atender às necessidades e limitações da rede, além de novas aplicações sem-fio que utilizem essa tecnologia. O IPv6 é a versão mais recente do *Internet Protocol* (IP), o padrão usado para a comunicação entre todos os computadores ligados à Internet. Ele foi proposto para atender ao crescimento da web, uma vez que utiliza 128 bits para os endereços para cada computador na rede, enquanto sua versão anterior, IPv4, utiliza 32 bits. Do ponto de vista de arquitetura, o IPv6 está construído sobre os princípios fundamentais da arquitetura IP. Em comparação com sua versão anterior, ele oferece espaço de endereçamento de várias ordens de magnitude maior, juntamente com características muito úteis para a Internet das Coisas.

O maior espaço de endereçamento se mostrou necessário para redes de grande escala: embora algumas redes, como as redes de automação doméstica, podem consistir apenas de algumas dezenas de nós, em muitos outros casos, o número desses nós pode ser muito maior do que redes convencionais, em *Smart Grids*, por exemplo [36]. O IPv6 contém características que fizeram dessa versão do protocolo a escolhida para a Internet das Coisas. Dentre essas características estão o tamanho do espaço para endereçamento, permitindo que mais coisas estejam conectadas à Internet, além das vantagens de integração com a arquitetura da Internet [34]. Com isso, um *IETF Working Group* [iet] padronizou o uso de IPv6 em IEEE 802.15.4, que é um padrão

de camada física e controle de acesso ao meio para *Low-Rate Wireless Personal Area Networks* (6LoWPAN).

1.2 Redes sem fio de baixa potência (*Low Power and Lossy Networks, LLNs*)

A principal função de uma rede sem fio de baixa potência é, normalmente, algum tipo de coleta de dados [26; 23]. Aplicações baseadas na coleta de dados são muitas e os exemplos incluem monitoramento do meio ambiente [31], segurança [37] e observações científicas [38]. De modo a realizar a coleta de dados, uma estrutura de grafo acíclico é tipicamente mantida e um *convergecast* (fluxo de dados de baixo para cima a partir das folhas em relação à raiz) é implementado nessa topologia de rede. Muitos sistemas operacionais para nós sensores (por exemplo, *Tiny OS* [24] e Contiki OS [8; con]) implementam mecanismos (por exemplo, *Collection Tree Protocol* (CTP) [11] ou o *IPv6 Routing Protocol for Low-Power and Lossy Networks* (RPL) [39]) para manter topologias de rede acíclicas para suportar aplicações de coleta de dados. Em algumas situações, entretanto, o fluxo de dados na direção oposta – a partir da raiz, ou roteador de borda, no sentido das folhas torna-se necessário.

1.3 Motivação

O tráfego de dados a partir da raiz em direção às folhas pode surgir nas rotinas de configuração da rede, consultas específicas de dados (que podem ser transmitidas por *broadcast*, *multicast* ou *unicast* a um único nó destino), ou aplicações para automação residencial. Imagine uma rede sem fio de baixa potência conectando aparelhos e outros dispositivos eletrônicos de uma casa para um *gateway*, ou roteador de borda. Além disso, imagine que você está no escritório e deseja se conectar à sua geladeira em casa para verificar se o leite acabou e é preciso comprar mais em seu caminho para casa. Para cumprir essa tarefa, uma mensagem é enviada para o endereço IPv6 da sua geladeira. Esta mensagem vai primeiro ser entregue ao roteador de borda da sua casa, ou *gateway*, e, em seguida, ser encaminhada para baixo através da (sub)rede sem fio *multihop* que conecta seus aparelhos. Cada nó na rede deve agir como um roteador e decidir qual o caminho para encaminhar a mensagem, de modo que ela atinja a geladeira.

Além disso, um algoritmo de coleta de dados muito comum em redes sem fio, conhecido como *convergecast*, pode ativar o tráfego de dados no sentido descendente em cenários onde falhas são comuns. O algoritmo de *convergecast* pode ser formalmente

definidos como se segue: em um ambiente consistindo de nós de sensores sem fios, cada nó i possui uma unidade de dados que deve ser transmitida para a estação base. Cada nó possui um alcance de transmissão, que pode ser ajustado até um valor máximo de r . Todos os nós dentro da área de transmissão são considerados os seus vizinhos. Um nó sensor, se dentro da faixa, pode transmitir seus dados para a estação de base diretamente (*singlehop*) ou através de outro nó (*multihop*) [33]. Quando a rede opera em um cenário onde falhas são comuns, é possível que os nós optem por receber uma confirmação da raiz sobre o recebimento de suas mensagens, ou seja, a raiz ou roteador de borda deve enviar uma mensagem descendente para um determinado nó, assim que receber uma mensagem de coleta desse nó.

1.4 Solução Proposta

Mesmo o tráfego de dados de cima para baixo não sendo tipicamente o principal alvo das redes sem fio *multihop* de baixa potência, ele é ativado por alguns esquemas de roteamento populares como uma função de exceção, o que significa que as rotas são otimizadas para o tráfego de baixo para cima e mensagens descendentes podem seguir por rotas mais longas ou não serem entregues devido a restrições de memória. No protocolo RPL, por exemplo, é mantida uma topologia de rede acíclica, chamada DAG (*Directed Acyclic Graph*), em que cada nó mantém na memória uma pequena lista de nós pais, ordenados pela qualidade da conexão, para os quais ele encaminha os dados em direção ascendente à raiz. Se um nó quer atuar não somente como uma fonte de dados, mas como destino, ele envia uma mensagem DAO (*Destination Advertisement Object*) para cima através do DAG, e os nós intermediários (se eles operam no assim chamado modo *storing*) adicionam uma entrada para este destino na tabela de roteamento criada especificamente para rotas descendentes.

O tamanho de cada tabela deste tipo é potencialmente $O(n)$, onde n é o tamanho da sub-árvore com raiz em cada nó, ou seja, o número total de descendentes do nó. Dado que a capacidade de memória pode ser altamente restringida em redes sem fio de baixa potência, muitas vezes nem todas as rotas podem ser armazenadas e, conseqüentemente, pacotes serão descartados. Isso resulta na perda de um número elevado de mensagens e um alto consumo de memória. A fim de reduzir o tamanho da tabela de roteamento, seria desejável agregar vários endereços de nós de destinos próximos em uma única entrada da tabela. Entretanto, porque os endereços IPv6 têm os seus últimos 64 bits derivados do endereço MAC único de cada nó, eles são basicamente aleatórios e não contêm informações sobre a topologia da rede.

Neste trabalho, propomos o MHCL (*IPv6 Multihop Host Configuration for Low-power wireless networks*)—uma estratégia de configuração *multihop* que explora estruturas de redes acíclicas em redes sem fio de baixa potência para gerar e atribuir endereços IPv6 para os nós. O objetivo é permitir tráfego descendente eficiente e robusto (*broadcast*, *multicast* e *unicast*) com baixo consumo de memória, ou seja, tabelas de roteamento pequenas. Propomos e analisamos duas estratégias de alocação de endereços: *Top-down* e *Bottom-up*.

Na abordagem *Top-down*, cada nó, começando com a raiz e terminando com as folhas, realiza um particionamento do espaço de endereços disponível, ou que lhe é atribuído por um dos pais, em “fatias” de endereços de tamanhos iguais e distribui essas fatias a seus filhos (deixando uma faixa de endereços reserva para eventuais futuros filhos). A abordagem *Bottom-up* contém uma fase inicial de agregação, em que nós computam seu número de descendentes.¹ Uma vez que a raiz recebe o número total de descendentes de cada um dos seus filhos (imediatos), atribui uma fatia/partição de endereços proporcional ao tamanho da sub-árvore com raiz em cada um de seus filhos (também deixando um espaço de endereço de reserva para futuras conexões). Uma vez que um nó recebe uma fatia de endereço de seu pai preferencial, ele particiona essa faixa entre seus próprios filhos, usando o mesmo algoritmo. O MHCL gera e atribui endereços IPv6 que refletem a topologia da rede sem fio subjacente. Como resultado, as tabelas de roteamento para tráfego descendente são de tamanho $O(k)$, onde k é o número de filhos (imediatos) de cada nó de roteamento em uma topologia acíclica. Foram usados temporizadores inspirados no algoritmo *Trickle* [25] para ajustar o mecanismo de comunicação em nosso protocolo, de modo que ele é capaz de se adaptar rapidamente à dinâmica na topologia da rede, sem inundar a rede com mensagens de controle.

Analisamos quão robusta é nossa estratégia em relação à dinâmica da rede, decorrente de mudanças nos pais preferenciais e falhas em enlaces e nós. Mostramos que, se os endereços são atribuídos na fase de configuração inicial da rede, as mensagens podem ser entregues com alta taxa de sucesso, apesar de falhas locais em links ou nós. Demonstramos por meio de emulações (usando o simulador *Cooja* [10] do sistema operacional Contiki OS [8] com implementação do protocolo RPL) que o MHCL é robusto à dinâmica da topologia de redes sem fio de baixa potência.

¹Em caso de DAG, onde um nó pode ter vários pais, o pai preferido é usado para fins da função de agregação.

1.5 Objetivos

O principal objetivo deste trabalho é propor um protocolo para o endereçamento IPv6 hierárquico de nós em uma rede sem fio de baixa potência. Isto deve ser feito de forma que não haja um gasto excessivo dos recursos da rede em comparação com a estratégia utilizada atualmente no IPv6. Além disso, é necessário que o endereço de um nó reflita sua posição na topologia da rede. Um outro objetivo, decorrente do endereçamento hierárquico, é a diminuição das tabelas dos protocolos de roteamento para redes sem fio que utilizam o IPv6. Com isso, o protocolo MHCL foi desenvolvido com duas estratégias de coleta de informação para o endereçamento baseado na divisão de faixas sequenciais de endereços.

1.6 Contribuições

As principais contribuições do trabalho proposto são apresentadas abaixo.

- Análise e comparação de diferentes técnicas de endereçamento IPv6, em relação ao possível desperdício de endereços e a limitações de topologia impostas por cada estratégia;
- Implementação de duas estratégias de endereçamento IPv6 hierárquico, avaliando a eficiência e o gasto de recursos dos algoritmos. As duas estratégias implementadas como uma rotina do protocolo RPL, no sistema operacional Contiki, apresentam versão final preparada para o funcionamento em uma rede real (com nós *OpenMote* [Ope], por exemplo);
- Avaliação e comparação da execução de um protocolo de roteamento utilizando três diferentes técnicas de endereçamento IPv6: a estratégia padrão e as duas propostas e desenvolvidas neste trabalho (MHCL Bottom-up e MHCL Top-down). A estratégia padrão do IPv6 considerada é a versão implementada no *Contiki RPL*, onde endereços IPv6 são gerados a partir de um endereço da interface. Para a avaliação, foi considerada a geração de endereços IPv6, com os 64 bits de *host* baseados em um endereço MAC de 48-bits, de acordo com a Função IEEE EUI-64[16] (veja Seção 4.1.4). Com base nas avaliações, o protocolo RPL que emprega o endereçamento do MHCL se mostrou mais eficiente quanto ao gasto de memória (tabelas de roteamento menores: $O(k)$ onde k é o número de filhos de cada nó, ao invés de uma entrada para cada descendente), energia (envio de menos mensagens) e apresentou uma taxa de sucesso no roteamento descendente

até $3\times$ maior em comparação com o RPL que emprega o endereçamento IPv6 padrão.

1.7 Organização do Trabalho

O restante deste trabalho está organizado da seguinte forma. No Capítulo 2, descrevemos brevemente sistemas 6LoWPAN, com foco nas redes sem fio de baixa potência, o endereçamento IPv6 e o protocolo RPL. No Capítulo 3 discutimos os trabalhos relacionados, que apresentam os problemas existentes nos sistemas 6LoWPAN e protocolos de roteamento para essas redes, além das funções de agregação e algoritmos distribuídos. No Capítulo 4 apresentamos as versões Top-down e Bottom-up do MHCL, as estratégias de endereçamento não implementadas e uma análise teórica de cada estratégia. No Capítulo 5 apresentamos nossos resultados experimentais. Por fim, no Capítulo 6 encerramos com algumas conclusões finais e trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Sistemas 6LoWPAN

Nesta seção apresentamos alguns conceitos sobre sistemas 6LoWPAN (*IPv6 over Low power Wireless Personal Area Networks*¹). Resumidamente, essas redes são compostas de dispositivos de baixo custo que permitem comunicação sem fio, porém com recursos limitados, obedecendo ao padrão IEEE802.15.4. Inicialmente, introduzimos as redes sem fio de baixa potência com perdas (LLNs: *Low-power and Lossy Networks*). Em seguida, discutimos o endereçamento IPv6, contextualizando o 6LoWPAN. Em seguida, apresentamos o protocolo RPL, com os principais conceitos utilizados no desenvolvimento deste trabalho.

2.1.1 Low-Power and Lossy Networks (LLNs)

Low Power and Lossy Networks (LLNs) são redes compostas de dispositivos embarcados, com poder de processamento, memória e energia limitados, com conexões propensas a perdas. Nessas redes, perdas de pacotes são extremamente frequentes, e os links podem se tornar inutilizáveis por algum tempo devido a inúmeras razões [36]. Existem diversas aplicações para esse tipo de rede. Dentre elas estão automação industrial e residencial, monitoramento de ambientes, redes urbanas, etc. Devido a características, tais como constante perda de conectividade de nós, limitação de energia e memória bem como possibilidade de conectar milhares de nós, uma solução de roteamento encontra diversos desafios. Por isso, um *IETF Working Group* foi criado para definir requisitos de roteamentos específicos para as LLNs, surgindo assim o protocolo de roteamento RPL [32].

¹<http://datatracker.ietf.org/doc/rfc4919/>

2.1.2 Endereçamento IPv6

O 6LoWPAN possui um mecanismo de compressão de cabeçalho, que permite que pacotes IPv6 sejam encaminhados em redes sem fio de baixa potência, particularmente em IEEE 802.15.4 [17]. Entre o 6LoWPAN e uma rede IPv6 existe um roteador de borda que realiza a compressão do cabeçalho. O grande espaço de endereçamento do IPv6 permite que os dispositivos possuam diversos endereços, que podem ser utilizados para vários fins. Por exemplo, a comunicação local pode ser realizada por meio do endereço de *loopback* e o endereço de conexão local, enquanto a comunicação global com a Internet é realizada utilizando o endereço global.

O IPv6 utiliza 128 bits para representar um endereço (enquanto sua versão anterior, IPv4, utiliza apenas 32 bits). Normalmente, os 128 bits do endereço IPv6 são divididos em duas partes: o prefixo da rede (64 bits) e o endereço do nó (64 bits). O mecanismo de compressão de cabeçalho 6LoWPAN omite os bits do prefixo de rede, pois eles são fixos para uma determinada rede 6LoWPAN. Uma vez que os outros 64 bits podem abordar um espaço de endereçamento muito grande, o 6LoWPAN fornece opções para comprimir o endereço do nó (o uso comum é de 16 bits)². Neste trabalho, estamos considerando os 64 bits menos significativos (endereço do nó) do endereço IPv6, comprimido em 16 bits.

Similarmente ao IPv4, existem duas maneiras diferentes para os dispositivos obterem seus endereços: dinâmica ou estática. A atribuição de endereço estático precisa ser configurada manualmente em cada dispositivo. A configuração dinâmica é realizada pelo protocolo DHCPv6 (*Dynamic Host Configuration Protocol version 6*). Isso requer a execução de um servidor DHCPv6 em algum lugar na rede. Durante a inicialização, nós se configuram apenas com endereço de enlace local. Eles enviam solicitações DHCPv6 ao agente DHCPv6 local para encontrar servidores vizinhos ou agentes de retransmissão. Cada roteador também executa um agente de retransmissão DHCPv6, que encaminha solicitações de endereço em direção ascendente da árvore para o roteador de borda (*gateway*), onde se presume que o servidor DHCPv6 está localizado. Uma vez que o servidor DHCPv6 atribuiu ao nó um endereço global ele então começa a executar o protocolo de roteamento (RPL).

²<http://datatracker.ietf.org/doc/rfc6282/>

2.1.3 Protocolo RPL

2.1.3.1 Grafo Acíclico Direcionado

O RPL é um protocolo de roteamento desenhado para LLNs e utiliza endereços IPv6. Dispositivos de rede executando o protocolo estão conectados de maneira acíclica. Para esse propósito, um Grafo Acíclico Direcionado Orientado ao Destino (DODAG) é criado. O grafo é construído com o uso de uma função objetivo (OF) que define como as métricas de roteamento são computadas de acordo com as restrições (usando o número esperado de transmissões (ETX) ou a quantidade atual de energia da bateria de um nó, por exemplo). Para iniciar o processo de construção do DAG a raiz anuncia informações sobre o grafo por meio de mensagens. Os nós vizinhos à raiz receberão essa mensagem (potencialmente também receberão DIO de outros nós vizinhos) e devem tomar decisões baseadas em sua OF. Uma vez que o nó se junta ao grafo, ele escolhe a raiz como seu pai e calcula seu *rank*. O *rank* é uma métrica que indica as coordenadas do nó na hierarquia da rede [35]. Os demais nós irão repetir esse processo de seleção do pai e notificação de informação do DAG, por meio de DIOS. Quando esse processo se estabiliza, a coleta de dados pode começar.

2.1.3.2 Mensagens

O RPL especifica três tipos de mensagens utilizando o ICMPv6: *DODAG Destination Advertisement Object* (DAO), *DODAG Information Object* (DIO) e *DODAG Information Solicitation Message* (DIS).

1. DIS: Esse tipo de mensagem é similar a mensagens de solicitação de rotas (RS) do IPv6, e são usadas para descobrir DODAGs na vizinhança e solicitar DIOS de nós vizinhos, sem possuir corpo de mensagem.
2. DIO: Mensagens desse tipo são utilizadas para anunciar um DODAG e suas características. Dessa forma, elas são usadas para a descoberta de DODAGs, sua formação e manutenção. O intervalo de envio de DIO é controlado pelo algoritmo Trickle [25]. A Figura 2.1 apresenta o modelo de uma mensagem do tipo DIO. O *RPLInstanceID* é um dos três valores utilizados para se identificar um DODAG na rede. Com o valor do *RPLInstanceID* é possível identificar um conjunto de DODAGs em uma instância RPL. O *Version Number* é responsável por armazenar a quantidade de vezes que uma reconstrução do DODAG foi necessária. O *Rank* identifica qual o *rank* do nó que está enviando o DIO. Esse valor determina qual a posição do nó no DODAG em relação à raiz e é utilizado em mecanismos respon-

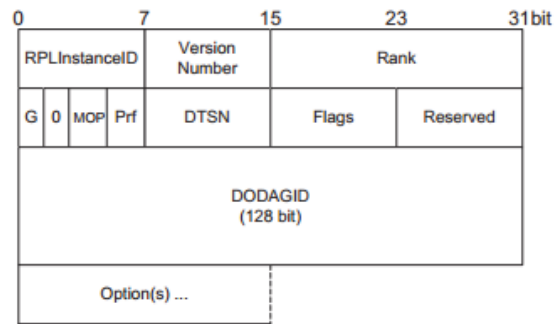


Figura 2.1: Pacote da mensagem DIO.

sáveis pela manutenção da topologia como, por exemplo, no algoritmo para evitar *loops* e na escolha do pai preferido. O campo *Grounded* (G) indica se naquele DODAG a raiz é a meta da função objetivo. O campo MOP é definido pela raiz DODAG e define qual o modo de operação usado para o roteamento descendente. O campo Prf define quão preferível o nó raiz é comparado com outros nós raiz. O último campo usado, DTSN, é necessário para salvar um número de sequência. Esse número é mantido pelo nó que emite a mensagem DIO e indica quão fresca a mensagem é [32; 36].

3. DAO: Mensagens DAO são utilizadas durante o processo de notificação de rotas descendentes. Elas são enviadas em sentido ascendente (dos nós que manifestam o desejo de receber mensagens para seus pais preferenciais) para propagar informações de destino ao longo do DODAG. Essas informações são utilizadas para o preenchimento das tabelas de roteamento descendente, que permitem o tráfego P2MP (ponto a multi-ponto) e P2P (ponto a ponto). O campo *RPLInstanceID* é o valor de identificação da topologia que foi aprendido com DIO. O sinalizador *K* indica se o nó espera receber uma confirmação ou não (*DAO-ACK*). O próximo sinalizador, *D*, é responsável por informar se o campo *DODAGID* está presente no pacote. O DODAGID pode ser omitido pois seu valor representa o endereço IPv6 da raiz, e no caso de existir apenas uma raiz ele não é necessário. Por fim, o *DAOSequence* é um contador incrementado toda vez que uma mensagem DAO é enviada pelo remetente.

2.1.3.3 Rotas Descendentes

O protocolo especifica dois modos de operação para o roteamento descendente: *storing* e *non-storing*. Ambos os modos requerem a geração de mensagens DAO, que são

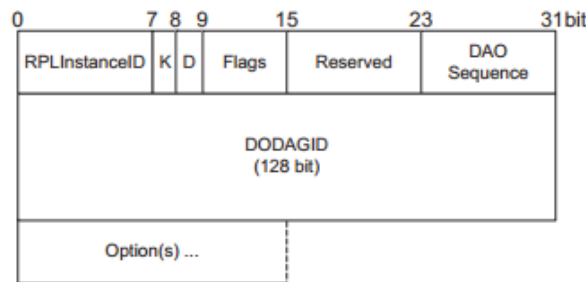


Figura 2.2: Pacote da mensagem DAO.

enviadas e utilizadas de maneira diferente em cada modo de operação. Os dois modos de operação são descritos a seguir:

1. Modo *storing*: No modo de operação *storing*, cada roteador RPL deve armazenar rotas para seus destinos em um DODAG. Essas informações são repassadas dos nós para seus pais preferenciais. Isso faz com que, em nós mais próximos da raiz, o armazenamento necessário seja definido pelo número de destinos na rede. Com isso, a memória necessária em um nó próximo à raiz e um outro distante da raiz pode variar drasticamente, o que faz com que algum tipo de implementação e manutenção administrativa contínua nos dispositivos sejam necessárias, conforme a rede evolui [6]. Entretanto, tal intervenção é inviável, devido ao perfil dinâmico da rede.

2. Modo *non-storing*: No modo *non-storing* cada nó gera e envia mensagens DAO para a raiz do DODAG. O intervalo no qual o DAO é enviado varia de acordo com a implementação. Entretanto, a especificação do protocolo sugere que esse intervalo seja inversamente proporcional à distância do nó a raiz. Dessa forma, um nó folha gera mais mensagens que um nó intermediário, por exemplo. Após coletar toda informação necessária, a raiz agrega essa informação. Se ela precisa enviar uma mensagem descendente na rede, ela deve utilizar um cabeçalho IPv6 para roteamento de origem (*source routing*). Dessa forma, os nós encaminham a mensagem até que ela alcance seu destino [32]. Ou seja, caso os nós não possuam capacidade de armazenamento para operarem no modo *storing*, a rede sofre maior risco de fragmentação e, portanto, perda de pacotes de dados, consumindo a capacidade da rede com o roteamento de origem [6].

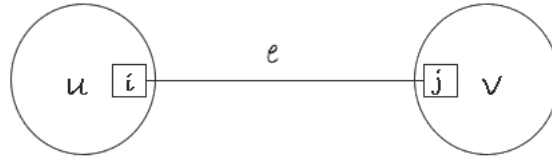


Figura 2.3: Aresta $e = (u, v)$ e as portas i e j que ela conecta. Fonte: [28]

2.2 Modelos de Computação Distribuída

Nesta seção serão apresentados os modelos de computação distribuída utilizados neste trabalho. Para a análise teórica do MHCL é considerado o modelo síncrono e sem falhas. Entretanto, a avaliação experimental, por meio de emulações, considera o modelo assíncrono e com falhas. Além disso, algumas terminologias da teoria dos grafos são apresentadas.

2.2.1 Componentes do Sistema

O modelo teórico de comunicação consiste em uma rede de comunicação ponto a ponto, descrita por um grafo $G = (V, E)$, onde os vértices $V = \{v_1, v_2, \dots, v_n\}$ representam os processadores (nós) na rede e as arestas representam canais de comunicação bidirecionais operando entre os nós [28]. Inicialmente, identificadores únicos são atribuídos para os processadores do grafo G . Para completude, podemos assumir que esses identificadores são atribuídos de um conjunto ordenado de inteiros $S = \{s_1, s_2, \dots\}$, onde $s_i < s_{i+1}$ para cada $i \geq 1$. Dessa forma, a atribuição de identificadores é um mapeamento um pra um ($ID : V \rightarrow S$). Com isso, o vértice v pode também ser referenciado por $ID(v)$ na descrição de rotinas e conceitos. Esse identificador deve ser definido como um valor fixo atribuído ao processador a nível de *hardware*. O modelo de comunicação é dado como segue. Cada vértice v possui $grau_G(v)$ portas, ou seja, pontos de conexões externas, numeradas de um a $grau_G(v)$. O conjunto de arestas adjacentes ao vértice v contém exatamente $grau_G(v)$ arestas, com exatamente uma aresta conectada a cada porta de v . Cada aresta (u, v) em E corresponde a um par $((u, i), (v, j))$, onde $1 \leq i \leq grau_G(u)$ e $1 \leq j \leq grau_G(v)$, representando um canal conectando a porta i de u à porta j de v , como mostra a Figura 2.3

O vértice u envia uma mensagem para seu vizinho v por carregá-la na porta apropriada, i . Essa mensagem é então recebida em v pela porta j . Para isso, cada vértice possui um *buffer* de entrada para cada conexão adjacente, e cada mensagem que chega é colocada nesse *buffer* e é lida do *buffer* pelo processador. Além disso, é assumido que no máximo uma mensagem, em qualquer direção, pode ocupar o canal

de comunicação em qualquer dado momento. Dessa forma, o canal de comunicação de u a v só fica disponível depois que a última mensagem trocada entre os nós tenha sido removida pelo processador receptor do *buffer* de entrada.

2.2.2 Modelo Síncrono

O modelo síncrono definido é aquele em que toda a computação prossegue em uma sequência de rodadas distintas. A cada rodada cada processo envia mensagens, possivelmente para cada outro processo, baseadas nas mensagens que eles receberam nas rodadas anteriores [22]. Em uma rede completamente síncrona é assumido que todos os tempos de conexão são delimitados. Dessa forma, cada processador mantém um *clock* local cujo pulso deve satisfazer a seguinte propriedade: uma mensagem enviada de um nó v para seu vizinho u em um pulso p de v deve chegar em u antes do pulso $p + 1$ de u . Dessa forma, o ciclo de cada processador pode ser descrito como a rotina de três passos: (1) enviar mensagens para (alguns dos) vizinhos; (2) Recebe mensagens de (alguns dos) vizinhos; (3) Realiza alguma computação local [28].

2.2.3 Modelo Assíncrono

Em um modelo completamente assíncrono é assumido que mensagens são eventualmente entregues e processos eventualmente respondem, mas não é feita uma suposição de quanto tempo isso pode levar [22]. No modelo assíncrono algoritmos são eventos impulsionados, isto é, processadores não podem acessar um *clock* global para decidirem sua ação. Mensagens enviadas de um vértice para seus vizinhos chegam em um tempo finito mas imprevisível. Isso implica que não é possível se basear no tempo de espera corrido para deduzir que uma mensagem não foi enviada por um vizinho em um dado tempo ou que a mensagem foi perdida; sempre pode ser o caso que a mensagem ainda está a caminho. Também é impossível, no modelo assíncrono, depender da ordem de chegada das mensagens de diferentes vizinhos para inferir na ordem de diversos eventos computacionais, uma vez que a ordem de chegada pode se inverter devido à diferentes velocidades de transmissão de mensagens [28].

2.2.4 Modelo com Falhas

Em modelos de troca de mensagens é possível considerar falha em processos (ou nós) e falha na comunicação (ou conexão). Normalmente, falha de comunicação resulta apenas na perda de mensagens. Modelos podem permitir erros transitórios que destroem mensagens individuais ou podem considerar apenas falhas de conexões individuais.

Uma falha de comunicação pode causar a perda de todas as mensagens enviadas naquela conexão. Uma preocupação em particular com a queda de conexões é se deve ser considerada ou não a possibilidade da partição da rede, onde o grafo de comunicação pode se tornar desconexo, tornando impossível para alguns pares de nós de se comunicarem. Modelos de falhas são problemáticos porque é difícil determinar quão precisamente eles descrevem o comportamento do sistema real [22]. O espectro de tipos de falhas que podem ocorrer em sistemas distribuídos inclui falhas temporárias e permanentes e se estende a falhas de *software* e *hardware* em algumas máquinas. Isso pode fazer com que uma máquina fique defeituosa por um período de tempo ou até mesmo fique completamente inoperante, ou pode gerar problemas sérios de comunicação que envolvem a queda de uma conexão, ou erros temporários como a perda de uma mensagem. Entretanto, a falha de um componente não necessariamente resulta na falha de outros componentes. Dessa forma, é esperado que o sistema seja capaz de continuar funcionando mesmo na presença de algumas falhas (no processo ou conexão). Com isso, surgiram os algoritmos tolerantes a falhas. Tais algoritmos são esperados a continuarem executando corretamente e produzindo resultados corretos (os mesmos resultados que eles deveriam apresentar se não houvessem falhas), apesar de falhas ocasionais em alguns nós ou conexões [28].

Na avaliação prática do MHCL, por meio de emulações, dois modelos de falhas são implementados. As falhas são intermitentes, ou seja, não são permanente e ocorrem em períodos, podendo ocorrer em um enlace ou um componente (nó) da rede. Limitadas pela implementação do simulador, as falhas ocorrem a nível de pacote. Entretanto, é possível relacionar essa implementação com as falhas mais comuns em redes sem fio *multihop* com perdas. Dessa forma, a queda de um nó pode ocorrer quando ele deseja enviar uma mensagem: essa mensagem não é recebida por nenhum outro nó, indicando que ele estava falho. A falha em um enlace, por sua vez, ocorre quando uma determinada mensagem não é recebida apenas pelo destinatário: todos os outros nós que fazem parte do caminho recebem o pacote, entretanto o canal de comunicação que incide no nó destino está falho. Como apresentado na Seção 2.1.1, perda de pacotes e queda de nós intermitentes são as falhas mais comuns nos cenários das LLNs.

Falhas permanentes em um nó ou enlace podem comprometer a topologia da rede, sendo necessário uma reestruturação, o que interfere diretamente no algoritmo de endereçamento. Caso essa falha ocorra antes de começar o endereçamento, a topologia é refeita e o MHCL aguarda para começar o endereçamento após uma estabilização. Entretanto, caso ela ocorra durante a fase de endereçamento, uma sub-árvore com raiz no nó desconectado ou conectada pelo enlace falho ficará sem o endereçamento hierárquico, comprometendo o funcionamento do roteamento. Caso a falha ocorra após

os nós serem endereçados, o roteamento pode ser comprometido e pacotes de mensagens com destinatário em um descendente do nó falho ou do nó com o enlace falho serão descartados, prejudicando a aplicação da rede. Para evitar problemas decorrentes de falhas permanentes e outras alterações na topologia da rede, algoritmos propostos na Seção 6.2 devem ser implementados no MHCL.

2.2.5 Terminologia de Grafos

Para a discussão sobre os algoritmos distribuídos e a análise de complexidade do MHCL, é necessário compreender algumas terminologias básicas da teoria dos grafos. A seguir serão descritas as principais terminologias necessárias, de acordo com [9; 28]:

1. Caminho: Um caminho c de um vértice u a um vértice v em um grafo G é uma sequência de arestas $\{e_1, \dots, e_n\}$ de tal forma que cada aresta consecutiva é incidente em vértices consecutivos ao longo do caminho. O tamanho de um caminho é a quantidade de arestas que ele possui. Quando G é um grafo simples, o caminho pode ser representado pelo conjunto de vértices $\{v_1, \dots, v_n\}$ pelo qual ele passa. Um caminho é chamado de *circuito* se ele começa e termina em um mesmo vértice.
2. Árvore: Um grafo conectado que não contém ciclos é chamado acíclico. Se $G(V, E)$ é um grafo acíclico com apenas um componente, ele é chamado de árvore. Além disso, uma árvore T com n vértices possui $n - 1$ arestas e a adição de qualquer aresta a T cria exatamente um ciclo.
3. Raiz, pai, filho e folha: Uma árvore é dita *enraizada* se possui um determinado vértice, chamado raiz, no qual as arestas possuem uma orientação natural em direção a ele ou direção contrária. Em uma árvore com raiz, o pai de um vértice é o vértice conectado a ele no caminho da raiz. Por consequência, o filho de um vértice v é aquele que possui v como seu pai. Por fim, uma folha é um vértice que não possui filhos.
4. Grafo Acíclico Direcionado: um *grafo direcionado* $G(V, E)$ consiste em um conjunto não vazio de vértices V e um conjunto de arestas direcionadas E onde cada aresta $e \in E$ é associada com um conjunto ordenado de vértices. Uma aresta e que é associada com a par ordenado de vértices (u, v) é descrita como começando em u e terminando em v . Um *ciclo* é um circuito de tamanho pelo menos 3, sem a repetição de vértices, a não ser pelo primeiro e último. Dessa forma, um Grafo Acíclico Direcionado (DAG) é um grafo direcionado que não possui nenhum ciclo.

5. Distância: Para dois vértices u, w , considerando um grafo G sem peso nas arestas, definimos como $dist_G(u, w)$ o tamanho do menor caminho em G entre esses dois vértices, onde o tamanho do caminho é definido pelo número de arestas que o compõe, em um grafo não ponderado.
6. Raio: Para um vértice $v \in V$, o $Raio(v, G)$ denota a distância de v para o vértice mais distante dele no grafo G : $Raio(v, G) = \max\{dist_G(v, w)\}, w \in V$.
7. Profundidade: A profundidade de um vértice v em uma árvore T com raiz r_0 é a distância de v até a raiz, que pode ser dada por: $Profundidade_T(v) = dist_T(v, r_0)$. A profundidade da árvore T é a maior profundidade dentre todos os vértices ou, alternativamente, o raio em relação à raiz r_0 : $Profundidade(T) = Raio(r_0, T)$.

Capítulo 3

Trabalhos Relacionados

Neste capítulo serão discutidos os trabalhos relacionados aos sistemas 6LoWPAN e protocolos de roteamento para redes de baixa potência, além das funções de agregação distribuídas. Primeiramente, será feita uma análise dos sistemas 6LoWPAN, com foco na aplicação do protocolo IP em redes sem fio de baixa potência (Seção 3.1). Em seguida, na Seção 3.2, será feita uma avaliação dos protocolos de roteamento para redes de baixa potência, apresentando seus principais desafios. Na Seção 3.3 serão apresentados trabalhos que desenvolvem e definem funções de agregação. Por fim, serão apresentadas técnicas de endereçamento em redes móveis *ad-hoc* na Seção 3.5.

3.1 IPv6 e 6LoWPAN

Após cerca de 40 anos de sua criação, a Internet apresenta diferentes características e aplicações. Ela foi criada para conectar computadores entre si para permitir a transferência de arquivos, login remoto e acesso a alguma computação distante. Atualmente encontramos processamento computacional embutido em quase todos os dispositivos, máquinas, aparelhos, e instrumentos com cada vez mais capacidade de comunicação. O surgimento dessa nova Internet tem se tornado possível devido à intensa pesquisa em redes sem fio embarcadas de baixa potência ou *sensornets*. Entretanto, desde seu surgimento, esse mesmo impulso evitou a utilização dos princípios de *design* e limitações da arquitetura da Internet, argumentando que o modelo de camadas convencional era impraticável para os dispositivos de recursos limitados que estavam sendo incorporados no mundo físico. Ao mesmo tempo, partes da comunidade de *design* da Internet utilizaram essas mesmas questões de acomodar um grande número de *hosts*, configuração automática e extensibilidade para abraçar a nova versão do protocolo IP, o IPv6.

Ao longo da última década, os pesquisadores *SensorNet* abordaram uma série de desafios importantes em redes. Com a observação de que rádio de escuta ociosa domina o consumo de energia do sistema, inúmeros protocolos de enlace baseados em ciclo de trabalho com base em escuta amostrada e agendamento foram propostos [14]. Além disso, enquanto a Internet foi bem sucedida utilizando apenas mecanismos de realimentação fim a fim, os pesquisadores *SensorNet* demonstram a necessidade de realimentação salto a salto para alcançar uma maior visibilidade da rede, reagir mais rapidamente às condições locais, e evitar os altos custos de mecanismos fim a fim. Incorporar *sensornets* em uma rede IP não é diferente do que com qualquer outra tecnologia de rede. Uma *SensorNet* simplesmente constitui uma outra sub-rede dentro de uma rede IP. Uma sub-rede *SensorNet* é definida pelo conjunto de nós que podem se comunicar dentro do *SensorNet*. Roteadores de borda conectam sub-redes *SensorNet* a outras sub-redes, possivelmente construídas em diferentes tecnologias de enlace. De uma perspectiva de interligação de redes, nada foi alterado sobre o IP. Os desafios vêm de suportar IP e seus serviços essenciais dentro de uma sub-rede *SensorNet*. Enquanto IP tem demonstrado grande sucesso na utilização de uma ampla gama de tecnologias de enlace (incluindo *wireless*), as limitações de recursos inerentes à *sensornets* desafiam os pressupostos comuns.

Com isso, em [14] são apresentadas as principais alterações realizadas no IPv6 para a sub-rede *SensorNet* e os resultados obtidos. No núcleo de trazer IPv6 ao *sensornets* está o questionamento sobre emular um único domínio de *broadcast*, sobre o escopo e a autoconfiguração do IPv6, roteamento, encaminhamento, descoberta de vizinhos dentre outros aspectos. Para a avaliação, foi implementada uma pilha de rede IPv6 para *sensornets*. A avaliação do desempenho foi realizada sobre métricas mais importantes para aplicações embarcadas: baixo consumo de memória, alta confiabilidade e baixo consumo de energia. A pilha de rede inclui implementações completas do IPv6 ND (descoberta de vizinhos, do inglês *neighbor discovery*), encaminhamento, roteamento, dentre outros. Na camada de transporte foram implementados o TCP e UDP. Durante um período de 12 meses, todas as implantações alcançaram uma taxa de sucesso de entrega superior a 99% e ciclo médio de trabalho bem abaixo de 1%. Esses números foram alcançados pois os autores não emularam um único domínio de *broadcast* e sim usaram uma arquitetura "*route-over*" que minimiza o custo de *multicast link-local* e permite que protocolos baseados em IP operem em conjunto com a topologia de rádio. Além disso, um protocolo de roteamento que comunica de forma eficiente as informações de roteamento e evita contagem ao infinito foi implementado e técnicas

para escuta de baixo consumo com otimizações de agendamento foram utilizadas.

3.2 Protocolos de Roteamento

O protocolo RPL, desde sua criação, foi muito estudado e, por isso, existem diversos trabalhos avaliando sua especificação e funcionamento. O protocolo RPL possui dois modos de roteamento descendente, como apresentado anteriormente. Em [20] é apontado o desafio de fazer com que os dois modos de operação do RPL funcionem simultaneamente. Esse problema não é grave quando a rede possui nós homogêneos, uma vez que os nós irão operar com apenas um modo de roteamento descendente. Entretanto, aplicações como automação residencial possuem elementos de rede completamente heterogêneos, em função da capacidade de memória, processamento, fonte de energia, etc. Isso faz com que alguns nós optem por operar no modo *storing* e outros necessitem operar no modo *non-storing*. Para resolver esse problema de inoperabilidade, [20] sugere que em primeiro lugar nós que operam no modo *storing* sejam capazes de entender e trabalhar com o cabeçalho do roteamento de origem (*source routing*), que é utilizado no modo *non-storing* do protocolo. Além disso, é proposto que nós operando no modo *non-storing* enviem mensagens de notificação de rotas salto a salto, ao invés de fim a fim. Com essas duas alterações propostas, implementações do protocolo alcançaram maior performance de interoperabilidade. Em [40] os dois modos de operação do RPL também são avaliados, em comparação com os caminhos mínimos P2P disponíveis. A diferença em termos de custos entre as rotas P2P baseadas na topologia do DAG e as rotas P2P de menor custo disponíveis na rede é medida e soluções para melhorias no RPL são apresentadas. Para resolver esses problemas de rotas longas, uma extensão do RPL chamada RPL-P2P [12] foi desenvolvida pelo IETF. O RPL-P2P define um novo modo de operação que fornece ao RPL uma abordagem reativa para descobrir melhores caminhos entre uma fonte e destino arbitrários, sem ter que passar pela raiz ou o primeiro ancestral comum dessa fonte e destino.

Outros problemas do protocolo são expostos em [5]. Nesse trabalho é realizada uma avaliação crítica do protocolo RPL, cerca de dois anos após sua criação. A falta de especificação de algumas partes do protocolo é um problema apontado, que conta com diversos pontos ainda não bem definidos do protocolo. Enquanto as mensagens do tipo DIO utilizam o *Trickle* para especificar a temporização de mensagens de maneira simples e de fácil entendimento, detalhes sobre mensagens do tipo DAO são obscuros e nenhum temporizador para essas mensagens é definido. Um outro ponto abordado é o mecanismo de reparo local que, se implementado de maneira incorreta, pode gerar

ciclos na rede levando à perda de dados. Em [19] e [42] o protocolo RPL é avaliado de acordo com sua implementação em cada um dos sistemas operacionais. No primeiro trabalho, [19], é apresentada uma implementação do RPL, chamada *TinyRPL*. Em seu trabalho, eles usam a pilha IP de baixa potência de *Berkley (BLIP)* no *TinyOs* [24], que interage com sua implementação do protocolo. Mais precisamente, o plano de controle do *TinyRPL* comunica com a pilha BLIP que oferece uma implementação de plano de encaminhamento. A implementação do *TinyRPL* conta com todos os mecanismos básicos da definição do RPL, enquanto omite todos os opcionais. Para a validação, o protocolo RPL é comparado com o *Collection Tree Protocol (CTP)* [11], o protocolo padrão para coleta de dados do *TinyOs*. Um dos benefícios do RPL, em comparação com o CTP, é a possibilidade de tipos variados de padrão de tráfego (P2P, MP2P e P2MP), além de sua capacidade de conectar nós à Internet diretamente, trocando pacotes com endereços IPv6 global.

Em [7] é apresentada uma implementação de um protocolo de roteamento IPv6 para dispositivos com pouca memória em redes sem fio. O protocolo se baseia no RPL, um protocolo de roteamento para LLNs. A principal contribuição do artigo está no fato de, diferente das demais versões implementadas desse protocolo, essa versão utiliza menos de 1KB de memória RAM. O RPL organiza sua topologia em um grafo direcional acíclico (DAG) que é particionado em '*subDAGs*', cada um com uma raiz, chamado DODAG. Cada nó no DODAG pode possuir mais de um pai, definindo o preferencial, para facilitar o reparo da rede no caso de um nó falhar. Para que o RPL tenha suporte P2P, a raiz precisa ser capaz de rotear pacotes para seu destino. Além disso, os nós na rede devem apresentar uma tabela de roteamento. Na construção do DODAG é utilizado o conceito do *rank*. Cada nó apresenta um *rank*, que é referente ao seu posicionamento na rede, como foi apresentado no Capítulo 2 deste trabalho. O *rank* pode ser calculado de diferentes maneiras, mas ele sempre deve respeitar algumas regras. Um nó nunca pode ter um *rank* menor que seu pai. Caso isso aconteça, é detectada uma inconsistência na rede e um reparo é realizado. O objetivo dessa rotina é detectar e evitar *loops*. Na implementação realizada em [7], algumas modificações foram realizadas em relação ao RPL. A principal delas está relacionada com o consumo de memória. Como os dispositivos têm memória limitada e o objetivo é gastar o mínimo possível de RAM, os dispositivos não armazenam tabelas de roteamento. Apenas o nó *sink* armazena essas tabelas e esses nós são definidos como máquinas mais poderosas como, por exemplo, um notebook. Além disso, o *rank* de um nó é computado de acordo com o ETX (do inglês *expected transmission count*) que mede a qualidade de um caminho entre dois nós de uma rede de dados em pacotes sem fios. Como RAM é um recurso escasso para sistemas embarcados, são descritos dois aspectos para lidar

com essa restrição: uma estrutura de dados de fila circular e como transferir dados da memória RAM para memória *Flash*. Por meio de testes em ambiente real foi possível determinar que o algoritmo funciona e satisfaz o requisito de gastar menos de 1KB de memória RAM.

Finalmente, em [42] é estudada a performance do protocolo RPL no sistema operacional Contiki OS [8]. O algoritmo é simulado a fim de mostrar na prática implicações baseadas na descrição do protocolo. Por exemplo, de acordo com o mecanismo de inicialização da rede, é possível imaginar que mensagens de controle podem inundar a rede durante a construção do DAG e que essa etapa pode levar muito tempo. Os resultados mostram que inicialmente o número de mensagens de controle é superior ao número de mensagem de dados, mas essas mensagens tendem a diminuir após cerca de 60 segundos, o que pode ser determinado como o tempo de inicialização da rede, para os casos avaliados.

3.3 Função de Agregação

Para ser possível o endereçamento hierárquico proposto neste trabalho, é necessário realizar uma computação preliminar, a fim de coletar informações sobre a topologia. Dessa forma, funções de agregação são de extrema importância para o MHCL. Em [21], dado um grafo qualquer G conectado, composto de n nós e com diâmetro D , onde cada nó possui um elemento numérico, o objetivo do algoritmo de *k-selection* é determinar qual o *k-ésimo* menor elemento. É provado em [21] que a seleção distribuída, o problema apresentado anteriormente (*k-selection*), requer mais trabalho computacional do que outras funções de agregação como encontrar a média dos elementos ou o maior deles. É bem conhecido que funções distribuídas de agregação podem ser computadas usando a operação *convergecast* executada em uma árvore. A rotina dessa operação é descrita em [21] como segue. A raiz da árvore inunda a rede com mensagens para as folhas, pedindo para que elas comecem a agregação. Os nós internos da árvore aguardam até que eles recebam os dados agregados de seus filhos. Eles então aplicam a função de agregação a seus dados e os dados agregados recebidos e encaminha o valor resultante a seu pai. Isso é realizado até que a raiz receba todos os dados agregados de seus filhos. Essa função *convergecast* é rápida e termina em, no máximo, 2 vezes a profundidade da árvore. Entretanto, para o problema do *k-ésimo* elemento apresentado, definido como uma função de agregação holística, a operação de *convergecast* não pode ser aplicada. Isso porque em funções do tipo holística é necessário que os valores estejam centralizados em um único nó.

Em [43] funções de agregação para calcular soma, média, contagem dentre outros fatores são aplicadas em uma rede de sensores sem fio para realizar o monitoramento dessa rede. As funções de agregação citadas são calculadas com base em diferentes fatores da rede como, por exemplo, a taxa de perda, o nível de energia e a contagem de pacotes. Para isso, é proposto um algoritmo que permite a computação de determinadas funções de agregação com eficiência de energia e que não interfiram na computação da rede. Além disso, é mostrado em [43] que artefatos de comunicação podem impactar a computação dessas propriedades agregativas, em alguns casos o erro no valor agregado computado chega a 50%. Para a implementação, é assumido que a rede de sensores consiste em n nós distribuídos de maneira *ad-hoc* e cada nó possui um identificador único. Nós podem falhar, cair e outros nós podem se juntar à rede, entretanto os nós são estáticos ou se movem raramente. Cada nó pode se comunicar com seus vizinhos em um certo alcance. A comunicação entre os nós pode ser perdida devido à ruídos ou colisões. Não é assumido nenhum protocolo, mas assume-se a capacidade do rádio de fazer *broadcast* de mensagens para seus vizinhos. Com isso, é realizada uma agregação *in-network*, na qual cada nó computa um resultado parcial da função e passa esse resultado para algum vizinho, em sentido ascendente em direção à raiz.

3.4 Trickle

O algoritmo *Trickle* permite que nós em um meio compartilhado com perdas possam trocar informações de uma maneira altamente robusta, simples e escalável, com um baixo consumo de energia [27]. Ao ajustar as janelas de transmissão dinamicamente, o Trickle é capaz de difundir novas informações sobre a escala de tempo de transmissão de camada de enlace enquanto envia apenas algumas mensagens por hora quando a informação não se altera. No algoritmo trickle, esporadicamente um nó transmite dados (como dados sobre estados de rota ou versão de atualização, por exemplo) a não ser que ele escute alguma outra transmissão com dados que sugiram que sua própria transmissão é redundante [25]. Existem dois resultados possíveis de uma mensagem do Trickle. Cada nó que escuta a mensagem pode perceber que os dados da mensagem são consistentes com seu próprio estado ou um receptor pode detectar uma inconsistência.

O algoritmo Trickle possui três parâmetros de configuração:

- Imin: menor tamanho do intervalo;
- Imax: maior tamanho do intervalo;
- k: uma constante de redundância que possui o valor de um número natural.

Além disso, o Trickle trabalha com três parâmetros:

- I : o tamanho do intervalo atual;
- t : um tempo dentro do intervalo atual;
- c : um contador.

O algoritmo do Trickle segue as seguintes rotinas, como apresentado em [27]:

1. Quando o algoritmo começa sua execução, ele atribui a I um valor no intervalo $[I_{min}, I_{max}]$;
2. Quando um intervalo começa, o Trickle restabelece o valor de c em 0 e define t em um ponto aleatório no intervalo de $[I/2, I]$;
3. Sempre que o Trickle escuta uma transmissão consistente, ele incrementa o contador c ;
4. No tempo t , o Trickle transmite se e somente se o contador c for menos que a constante de redundância k ;
5. Quando o intervalo I expira, o Trickle dobra o tamanho do intervalo. Se o tamanho do novo intervalo for maior que o I_{max} , então I recebe o valor de I_{max} ;
6. Se o Trickle escuta uma transmissão inconsistente e I é maior que I_{min} , ele redefine o valor do temporizador Trickle. Para redefinir o temporizador, o Trickle atribui a I o valor de I_{min} e começa um novo intervalo como no passo 2.

Dessa forma, o algoritmo Trickle pode ser utilizado para o estabelecimento de eventual consistência de uma rede sem fio e para otimizar a disseminação de informações sobre a topologia, como é o caso de sua utilização no protocolo RPL [39]. No MHCL, o Trickle foi utilizado como base para a definição de um temporizador responsável pela estabilização da rede. Quando uma informação sobre a topologia não se altera, o temporizador tem seu valor dobrado até atingir o máximo. Caso a informação se altere, o valor é redefinido ao inicial e o nó deve esperar até o temporizador chegar ao valor máximo para ter certeza que a topologia está estável.

3.5 Técnicas de Alocação de Endereços em Redes

A atribuição de endereços a componentes de uma rede é o principal objetivo deste trabalho. Para auxiliar no desenvolvimento da técnica de alocação proposta pelo MHCL, alguns trabalhos sobre a alocação de endereços em redes móveis foram considerados. Em [4] é apresentado um esquema de endereçamento para redes móveis *ad-hoc* (MANET). O esquema de endereçamento provê soluções eficazes e eficientes para três principais problemas: (1) reconfiguração de endereços quando duas MANETs colidem; (2) a ligação de uma MANET a uma porta de entrada com fio e (3) a atribuição de um endereço único a um nó que se junta a uma rede. Realizar um endereçamento utilizando o IPv4 de 32 bits ou o IPv6 de 128 bits é uma solução possível, mas resulta em uma sobrecarga desnecessária. Para isso, é proposta uma solução que utiliza apenas o número mínimo de bits necessários para endereçar de maneira única a todos os nós que compõem uma determinada MANET, de tal forma que o comprimento de um endereço de ser expandido para acomodar novos nós, conforme necessário. O valor inicial utilizado para o endereço é de quatro bits e esse espaço aumenta em quatro bits a cada vez que uma expansão de endereço é necessária. O endereçamento é realizado em sequência e os nós precisam entrar em acordo sobre qual o próximo endereço disponível para um novo nó que deseja se juntar a rede. Além disso, quando duas redes se juntam, os endereços em cada uma pode ser feito único apenas adicionando um bit 1 no início de cada endereço de uma rede e adicionando o bit 0 no início de cada endereço da outra rede, no caso das duas redes possuírem endereços de mesmo tamanho. Por fim, quando a rede entra em contato com uma porta de entrada para a Internet, por exemplo, que utiliza IPv6, basta notificar o tamanho do endereço utilizado no momento e qual o maior endereço alocado e o componente com fio será capaz de gerar e atribuir um endereço IPv6 único para cada nó na MANET se comunicar com a rede com fio.

Em [41] é proposto um protocolo dinâmico de alocação de endereços que atribui um endereço IP exclusivo para cada novo nó em uma MANET, utilizando um algoritmo distribuído baseado em algoritmo genético. Além disso, o novo protocolo também propõe um mecanismo para detectar duplicatas e gerencia as operações da MANET, como partição e fundição de MANETs. A ideia do protocolo é baseada na atribuição de um "líder" em cada partição de rede, que possui as seguintes responsabilidades: (1) quando duas redes se detectam, o líder fornece a identificação de sua própria rede; (2) o líder aloca endereços para novos nós, semelhante a servidores DHCP em redes cabeadas; (3) ele também deve detectar endereços duplicados ou conflitantes e (4) quando duas redes se mesclam os líderes atuam como estações de retransmissão na rede. A solução de [41] utiliza o endereço MAC do nó e se baseia em um algoritmo

genético para gerar endereços IPv4. Nos 32 bits, são utilizados os primeiros 8 bits como a identificação de rede (NID), os últimos 24 bits que são gerados a partir dos bits menos significativos do endereço MAC e do endereço IP do nó que aceitou a entrada desse nó na rede utilizando o algoritmo genético. Em [30] é realizado um estudo de técnicas dinâmicas para o endereçamento de redes móveis. Entre principais requisitos de uma técnica de endereçamento está a detecção de duplicidade de endereços. Essa característica não se aplica ao MHCL, uma vez que os endereços gerados devem ser globalmente únicos, ou seja, dois nós não irão possuir o mesmo endereço. Entretanto, se adaptar à dinâmica da rede é uma característica importante apresentada em [30] que o MHCL deve implementar. Nós estão constantemente se movendo e podem ser desconectados por motivos como, por exemplo, a falta de energia ou uma falha temporária. Dessa forma, um deve ser capaz de obter endereços IP dinamicamente, sem a configuração manual ou estática. Além disso, é esperado que o protocolo seja robusto, ou seja, deve se adaptar à dinâmica da rede, incluindo as partições e fusões e que ele seja escalável, evitando a degradação significativa do desempenho quando o tamanho da rede aumenta. Esses quatro objetivos apresentados em [30] (Dinâmica, Unicidade, Robustez e Escalabilidade) foram almejados no desenvolvimento do MHCL. As técnicas de endereçamento foram divididas em três grupos: abordagem descentralizada, abordagem baseada em líder e abordagem melhor esforço. Os protocolos descentralizados abordam o problema da atribuição de endereços dinâmicos como um problema distribuído de entrar em acordo. Em abordagens baseadas em líder, por outro lado, os nós obtêm endereços IP a partir de um líder eleito na rede. Abordagens de melhor esforço, no entanto, permite aos nós atribuir seus próprios endereços sem o envolvimento de todos os outros nós na rede.

Capítulo 4

MHCL: Alocação de Endereços IPv6 em Redes Multi-hop de Rádios de Baixa Potência

O objetivo do MHCL (*IPv6 Multi-hop Host Configuration in Low-power wireless networks*) é implementar um esquema de alocação de endereços IPv6 para tráfego descendentes (*broadcast, unicast e multicast*) que consuma pouca memória, ou seja, necessite de pequenas tabelas de roteamento. O espaço de endereços disponível do roteador de borda, os 64 bits menos significativos do endereço IPv6 (ou 16 bits em uma representação comprimida), é particionado de maneira hierárquica entre os nós conectados no roteador de borda por meio de uma topologia *multihop* acíclica (implementada por protocolos padrões, como o RPL [39] ou CTP [11]). Cada nó recebe uma faixa (ou “fatia”) de endereços IPv6 de seu pai e particiona essa faixa entre seus filhos, até que todos os nós tenham sido endereçados. Uma vez que a alocação de endereços é realizada de maneira hierárquica, a tabela de roteamento de cada nó deve ter k entradas, onde k é o número de filhos (diretos). Cada entrada da tabela de roteamento agrega o endereço de todos os nós na subárvore com raiz no nó filho correspondente.

Para decidir como o espaço de endereçamento disponível deve ser particionado, os nós precisam coletar informações sobre a topologia da rede. Uma vez que uma versão *estável* da topologia da rede é alcançada, a raiz começa a distribuir as fatias de endereços. Note que o conceito de estabilidade é importante para implementar um particionamento de endereços coerente. Portanto, o MHCL possui uma fase de inicialização, durante a qual informações sobre a topologia são atualizadas progressivamente, até que um período de tempo (pré-definido) suficientemente longo se passe sem alterações na topologia. Para implementar essa abordagem adaptativa, usamos

temporizadores inspirados no algoritmo *Trickle* [25] para disparar mensagens, como explicamos em detalhes na Seção 4.3. Note que, uma vez que a rede atinge um estado de estabilidade inicial, mudanças de topologia futuras são esperadas ser de natureza local, provocadas por uma conexão falha ou falha de um nó, ou uma mudança no pai preferencial de um nó. Nesses casos, a atribuição de endereços não precisa ser atualizada, já que mecanismos locais de retransmissão de mensagens são suficientes para alcançar uma alta garantia de entrega de mensagens. Isto é confirmado pelos nossos resultados experimentais no Capítulo 5. No (atípico) caso de mudanças globais e permanentes na topologia da rede, o algoritmo MHCL deve ser reiniciado pela raiz.

No restante deste capítulo são descritos os principais componentes do MHCL: a alocação de endereços IPv6 (Seção 4.1), os tipos de mensagens (Seção 4.2), as rotinas de comunicação e temporizadores (Seção 4.3) e as tabelas de roteamento e rotinas de encaminhamento (Seção 4.4). Ao final, uma análise de complexidade teórica do algoritmo MHCL baseado no modelo distribuído descrito no Capítulo 2 é apresenta na Seção 4.5

4.1 Alocação de Endereços IPv6

A base para o endereçamento IPv6 dos nós é o endereçamento hierárquico, que é capaz de refletir a topologia da rede nos endereços. A distribuição de IPs nas estratégias implementadas acontece por meio de faixas de endereços. Dado um espaço de endereçamento disponível, um nó é responsável por guardar uma parcela para reserva, para o endereçamento de possíveis futuros filhos, e distribuir o restante da faixa de endereço que possui em faixas para seus filhos.

4.1.1 Top-down

A estratégia top-down tem como objetivo distribuir faixas endereços de tamanho proporcional à quantidade de filhos que o nó possui. O objetivo é mandar menos mensagens e gastar menos tempo. Na primeira etapa, os nós (exceto a raiz) devem decidir quem é o pai preferencial. Isso porque, durante a formação do DAG, o nó pode ir alterando seu pai até estabilizar e ter certeza de qual pai corresponde à melhor escolha, dada sua função objetivo. Após um nó ter certeza que contabilizou todos os filhos, por meio de temporizadores, caso o nó já possua uma faixa de endereços ele começa a realizar a distribuição das faixas entre seus filhos.

A Figura 4.1 apresenta uma ilustração dessa estratégia, considerando 8 bits de endereçamento. Neste exemplo, cada nó, antes de distribuir para seus filhos, armazena

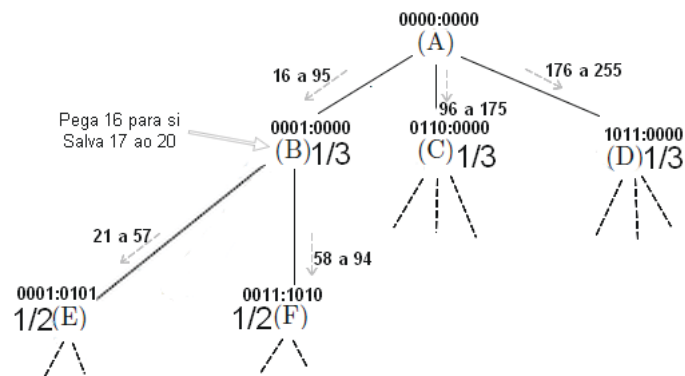


Figura 4.1: Exemplos de particionamento do espaço de endereçamento da estratégia Top-down.

6,25% de sua faixa de endereços para endereçar futuros filhos, uma vez que é necessário considerar o crescimento da rede. A partir disso, ele divide o restante da faixa de endereços em proporções, de acordo com a quantidade de filhos que possui. O valor de 6,25% é utilizado apenas como exemplo, sendo um parâmetro configurável do algoritmo.

Dessa forma, de acordo com a Figura 4.1, teremos 256 endereços possíveis (0 a 255). A raiz irá guardar 6,25% desse total, o que representa 16 endereços (0 a 15). Restando 240 endereços para distribuir entre seus filhos. Como ele possui 3 filhos, cada um receberá um terço desse valor, 80 endereços. B então, com 80 endereços salva 6,25%, o que consiste em 5 endereços (16 a 20). Como B possui 2 filhos, o restante dos 75 endereços são divididos entre os dois filhos, E e F, cada um recebendo 37 endereços.

4.1.2 Bottom-up

Na estratégia de endereçamento Bottom-up é realizada uma função de agregação responsável por calcular a quantidade de descendentes de cada nó. O objetivo é que cada nó possa realizar um endereçamento justo em função do número de descendentes de cada filho: quem possui mais descendentes deve receber uma faixa maior de endereços. Para isso, mensagens são enviadas em sentido ascendente com a quantidade de descendentes que o nó conhece, e deve ser atualizada sempre que ele calcular um novo valor. Ao final da etapa de contagem, o nó raiz deve começar o endereçamento.

A Figura 4.2 apresenta uma ilustração dessa estratégia, considerando 8 bits de endereçamento. Neste exemplo, cada nó, antes de distribuir para seus filhos, armazena 6,25% de sua faixa de endereços para endereçar futuros filhos, uma vez que é necessário considerar o crescimento da rede. A partir disso, ele divide o restante da faixa de endereços em proporções, de acordo com a quantidade de descendentes em cada filho.

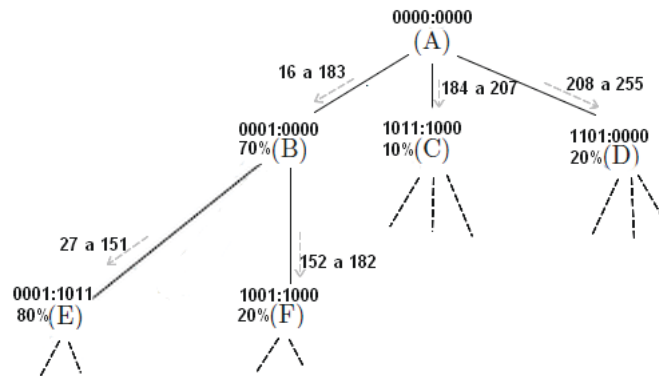


Figura 4.2: Exemplos de particionamento do espaço de endereçamento da estratégia Bottom-up.

Dessa forma, de acordo com a Figura 4.2, teremos 256 endereços possíveis (0 a 255). A raiz irá guardar 6,25% desse total, o que representa 16 endereços (0 a 15). Restando 240 endereços para distribuir entre seus filhos, B recebe 70% desse total, sendo 168 endereços (16 a 183). Continuando com a lógica de endereçamento, B então guarda para si 6,25%, o que representa 10 endereços (16 a 25). Ele então possui 158 endereços para distribuir para seus dois filhos (26 a 183). Como E possui 80% de descendentes de B e F possui 20%, E recebe 126 dos 158 endereços (26 a 151) e F recebe 32 endereços (152 a 183).

Podemos observar que no caso de uma topologia de árvore completamente balanceada o endereçamento Top-down e Bottom-up apresentam uma mesma configuração de endereços, uma vez que a porção de endereços será dividida de maneira igual entre cada filho, a cada nível da árvore. Além disso, em uma topologia muito desbalanceada (onde uma subárvore contém quase todos os nós da rede, enquanto outra subárvore contém apenas alguns) é possível afirmar que a estratégia Bottom-up apresenta uma vantagem em relação à Top-down, por levar em consideração o número de descendentes de cada nó.

4.1.3 Outras Estratégias

Estratégias de endereçamento IPv6 com a utilização de prefixos também foram estudadas. Dessa forma, foram definidas duas estratégias: *Prefixo Base 1* e *Estática*. A vantagem das duas estratégias está na utilização de prefixos. Por meio de um algoritmo de busca de maior prefixo comum é possível localizar um nó na rede durante o encaminhamento de mensagens com um menor custo de implementação em *hardware* e menor gasto de memória. Além disso, prefixos de endereço permitem a agregação de

informações de encaminhamento, o que proporcionou o crescimento da Internet [29]. Note que no endereçamento proposto, os prefixos de uma mesma rede podem possuir tamanhos diferentes. Com isso, é necessário armazenar o tamanho de cada prefixo e encaminhar essa informação nos pacotes ou armazenar todo o endereço de 64 bits para ser possível localizar o nó correspondente sem erros. Entretanto, devido às limitações de topologia impostas por essas estratégias, elas não foram implementadas. A tabela 4.1 apresenta uma comparação entre todas as estratégias, incluindo a utilizada atualmente no IPv6. Cada uma dessas estratégias são descritas a seguir, nas subseções 4.1.3.1 e 4.1.3.2.

4.1.3.1 Prefixo Base 1

Na estratégia Prefixo Base 1, para cada nó o número de filhos é correspondente ao número de bits máximo usados para definir o prefixo dos filhos. Nessa estratégia, todos os prefixos devem obrigatoriamente possuir sempre um bit 0 na última posição do prefixo, e pode existir somente um bit 0 por prefixo. Os demais bits do prefixo, caso existam, serão obrigatoriamente 1. Para realizar a distribuição de prefixos, os nós filhos são ordenados de acordo com o número de descendentes e o prefixo do filho com mais descendentes é aquele que apresenta apenas um bit, do segundo filho dois bits, até o n -ésimo filho receber um prefixo com n bits.

Para facilitar o entendimento dessa estratégia, temos o endereçamento de acordo com a Figura 4.3. Além disso, temos um segundo grafo, Figura 4.4 apresentando seu endereçamento. Para facilitar o entendimento e a visualização, apenas os prefixos são atribuídos (e não todo o endereço de *host* de 64 bits).

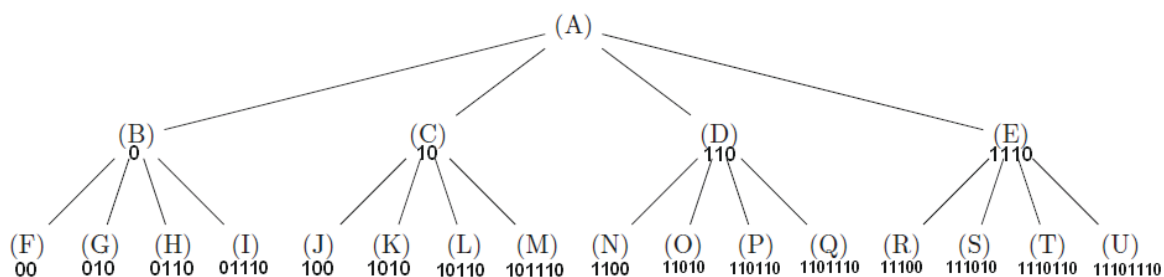


Figura 4.3: Grafo 1 - Prefixo Base 1

Com a utilização dessa estratégia, o roteamento na rede é facilitado, uma vez que temos prefixos para cada nó que vão sendo passados para seus descendentes e a divisão de endereços é realizada de uma maneira justa: o nó que possui mais descendentes recebe uma faixa de endereçamento maior. Entretanto, é possível perceber que essa

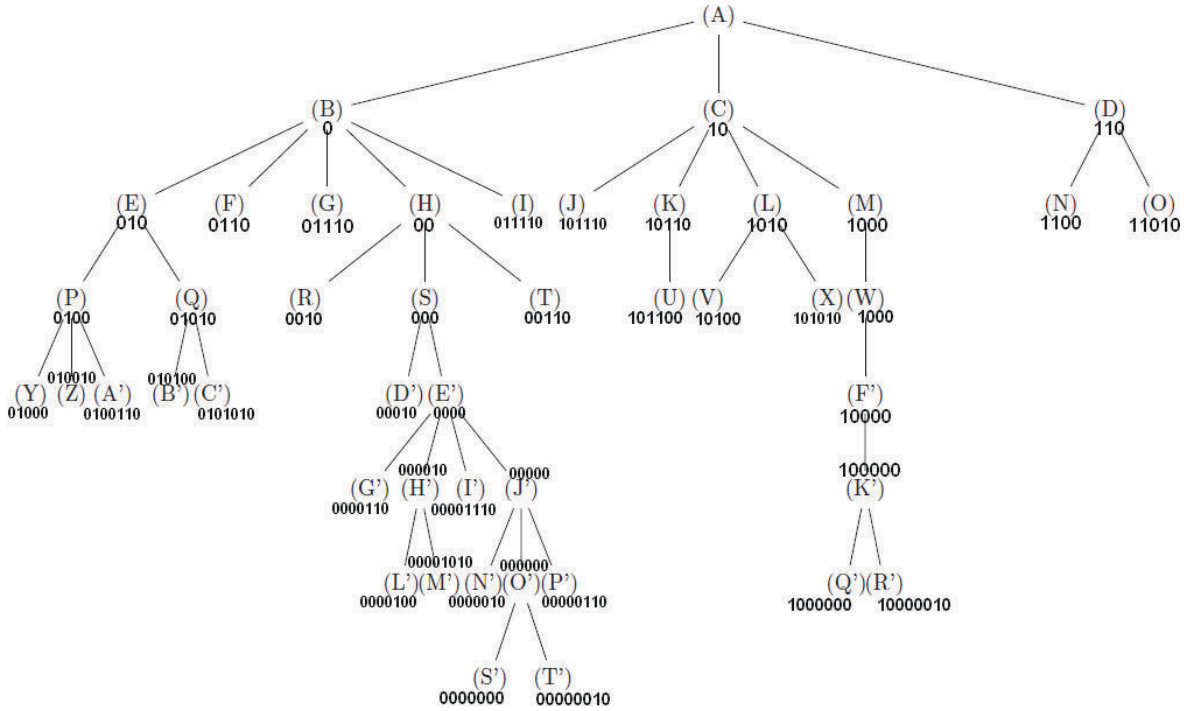


Figura 4.4: Grafo 2 - Prefixo Base 1

justiça não é proporcional ao número de descendentes em cada sub-grafo. O filho com o maior número de descendentes recebe o prefixo 0, ou seja, metade de toda a faixa de endereçamento disponível no nó pai. Com isso, essa estratégia se mostra melhor em um grafo desbalanceado. Além disso, cada nó poderá ter k filhos no máximo, onde k é o número de bits disponíveis naquele nó. Por exemplo, no segundo grafo o nó M' possui o prefixo 00001010, com 56 bits disponíveis. Dessa forma, o nó M' não poderá possuir mais que 56 filhos. Esse fator também limita a profundidade de um sub-grafo de um nó que possui poucos bits disponíveis em seu endereçamento.

Teorema 1. *Não haverá colisão de endereços ao aplicar a estratégia de endereçamento Prefixo Base 1 em um DAG.*

Demonstração. Considerando que houve colisão, existem 2 nós A e B com o mesmo prefixo de tamanho k . Sabendo que sempre existirá pelo menos um ancestral comum de A e B , vamos considerar o LCA_{AB} , (*Lowest Common Ancestor*) como o último ancestral comum responsável por endereçar ancestrais de A e B ou os próprios nós. Com isso, é possível identificar até qual bit está o prefixo do LCA_{AB} , uma vez que os bits 0 delimitam os prefixos de cada nó. Como estamos considerando que o prefixo de A é igual ao de B , então mesmo depois do prefixo do LCA_{AB} , os endereços de A e B permanecem o mesmo. Entretanto, como o LCA_{AB} foi o nó responsável por endereçar

seus filhos, sendo eles um ancestral de A e outro ancestral de B , eles só podem possuir prefixos iguais se forem o mesmo nó. Porém, como LCA_{AB} é o último ancestral comum de A e B , não poderia existir um descendente de LCA_{AB} que seja um ancestral comum de A e B , sendo isso uma contradição. \square

4.1.3.2 Estática (Balanceada)

Na abordagem estática o tamanho do prefixo é definido (tamanho 2 ou tamanho 4, por exemplo). Isso limita o número de filhos que cada nó pode ter e limita a altura da árvore uma vez que, dado o tamanho do prefixo p , apenas combinações de 0's e 1's podem ser realizadas em 2^p prefixos distintos.

Vamos considerar os dois grafos de exemplo, o primeiro com 2 bits de prefixo e o segundo com 4. Começando do primeiro grafo, na Figura 4.5 temos o endereçamento utilizando esta estratégia. Na Figura 4.6, temos a mesma estratégia, endereçando o segundo grafo de exemplo.

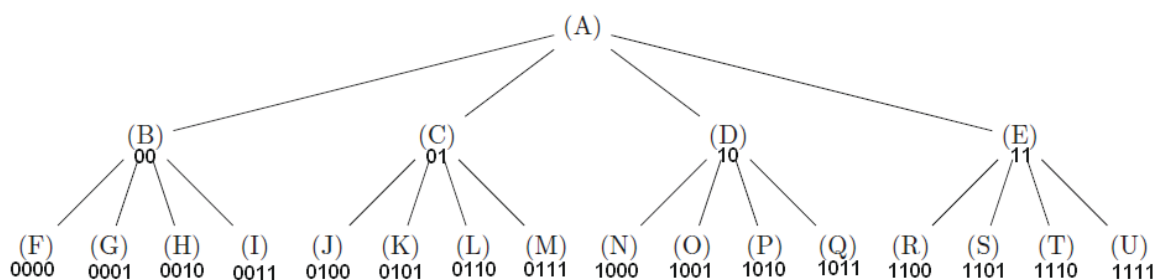


Figura 4.5: Grafo 1 - Estática

Pelos exemplos podemos concluir que a estratégia estática funciona muito bem em uma árvore balanceada. No seu melhor caso, o tamanho do prefixo é definido como: $\lceil \lg_2 C \rceil$, onde C é o número de filhos que um nó possui (C é o mesmo para todos os nós da rede).

Teorema 2. *Não haverá colisão de endereços em uma rede com topologia de DAG ao aplicar a técnica de endereçamento estática.*

Demonstração. Considerando que houve colisão, existem 2 nós A e B com o mesmo prefixo de tamanho k . Assim como na prova anterior, sabendo que sempre existirá

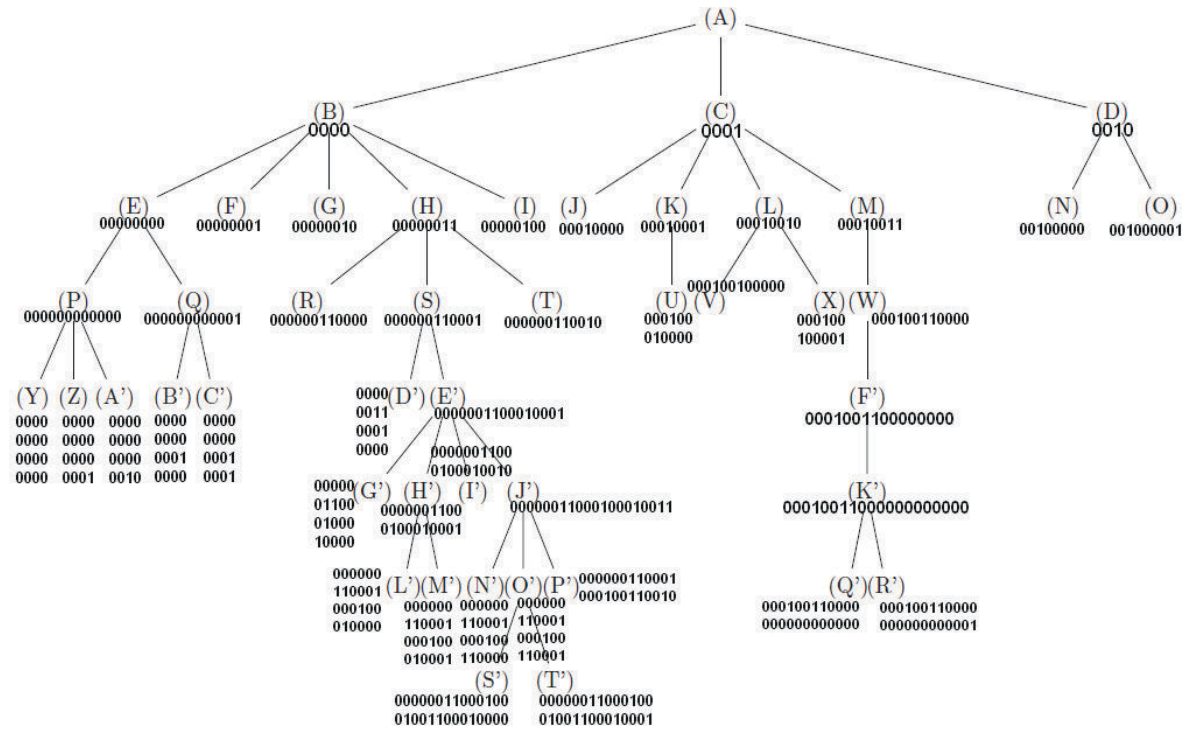


Figura 4.6: Grafo 2 - Estática

peelo menos um ancestral comum de A e B , vamos considerar o LCA_{AB} . Com isso, obrigatoriamente existe um ancestral de A ou o próprio A distinto de um ancestral de B ou o próprio B que foram endereçados pelo LCA_{AB} . Como os nós endereçados por seu pai devem receber prefixos diferentes, a partir do LCA_{AB} , se A e B continuam com o mesmo prefixo é porque eles possuem um mesmo ancestral descendente de LCA_{AB} . Entretanto, como LCA_{AB} é o último ancestral comum de A e B , chegamos a uma contradição. \square

4.1.4 IPv6 - Formato EUI-64

Atualmente, o endereçamento padrão do IPv6 divide igualmente o espaço de endereçamento de 128 bits em um prefixo de sub-rede que identifica a sub-rede dentro da Internet e um identificador de interface. Um mecanismo existente, que será utilizado como referência neste trabalho, realiza uma derivação do endereço MAC de 48 bits para um endereço de 64 bits da interface. Um endereço MAC 00:0C:29:0C:47:D5, por exemplo, se transforma em um identificador único de 64 bits através da inserção de FF:FE no meio, resultando em: 00:0C:29:FF:FE:0C:47:D5. Segundo [13], cada nó em uma rede IPv6 deve possuir um endereço *unicast*. Para todos os endereços *unicast*, o

ID da interface de possuir 64 bits e ser construído no formato *Modified EUI-64* [16]. Dependendo das características de uma determinada conexão ou nó, existem distintas abordagens para a criação do identificador de interface no formato *Modified EUI-64*. Neste trabalho estamos considerando a geração de endereços a partir de um endereço MAC de 48 bits [IEEE]. Para isso é necessário inserir dois octetos, com valores hexadecimais $0xFF$ e $0xFE$ no meio do endereço MAC. Em seguida, o sétimo bit da esquerda, ou o bit universal/local (U/L), deve ser invertido. Esse bit identifica se este identificador de interface é administrado universalmente ou localmente. Se for 0, o endereço é administrado localmente e se 1, o endereço é globalmente único. Note que nos primeiros 24 bits da divisão do endereço MAC, os endereços globalmente únicos atribuídos pelo IEEE sempre têm esse valor definido como 0 enquanto que os endereços criados localmente tem o valor 1. Portanto, quando o bit é invertido, mantém o seu escopo de origem (endereço único global ainda é único global e vice-versa). Ao final, temos um endereço no formato *Modified EUI-64*.

A tabela 4.1 apresenta os valores obtidos da análise de cada estratégia de endereçamento. Nessa análise, foram buscados fatores limitantes impostos por alguma regra do algoritmo. O desperdício de endereços foi avaliado uma vez que uma das principais características do IPv6 é a ampliação do espaço de endereçamento em comparação com sua versão anterior (IPv4). A limitação da topologia avalia, em casos gerais, se o algoritmo possui alguma característica capaz de limitar a quantidade de filhos por nó, a altura ou largura, por exemplo. Para calcular o desperdício de endereços foram considerados os 64 bits do IPv6 destinados aos nós da rede. Dessa forma, seria possível gerar, em uma mesma rede, 2^{64} endereços distintos. Para a limitação da topologia foi analisado se, de alguma maneira, o endereçamento proposto define um valor máximo para algum parâmetro da topologia. Para todas as análises, foi considerada a topologia de árvore, levando em consideração todos os nós do DAG, mas apenas as ligações entre filhos e seus pais preferidos. As estratégias implementadas, Bottom-up e Top-down não apresentaram desperdício do espaço de endereçamento uma vez que o algoritmo divide os 2^{64} possíveis endereços ao longo da topologia. Além disso, nenhuma limitação prévia da topologia é definida em função de algum parâmetro do algoritmo de endereçamento. Note que é possível existir alguma limitação caso uma topologia pré definida seja avaliada. Por exemplo, em uma árvore desbalanceada o algoritmo Top-down pode limitar a profundidade da maior sub-árvore, entretanto, a análise é realizada para casos gerais. Para a análise da estratégia Prefixo Base 1, o valor de k é definido como o tamanho do prefixo de um dado nó (k_i é o tamanho do prefixo do endereço de *host* do nó i) e $\min(k)$ sempre será 1, uma vez que um nó poderá sempre atribuir um prefixo de tamanho 1 para seu filho que apresentar mais descendentes. Na estratégia Estática, k

apresenta sempre um valor estático, sendo ele o tamanho do prefixo estático que é definido. Para não existir um desperdício de endereços nessa estratégia, é necessário que 64 seja múltiplo de k , além do DAG ser balanceado e cada nó possuir exatamente 2^k filhos. Na estratégia adotada atualmente no RPL não existe limitação na topologia da árvore. Entretanto, ela apresenta um desperdício de endereçamento pois, ao converter o endereço MAC em um identificador único de 64 bits, 16 bits são fixados (bits 25 ao 40, com o valor $FF : FE$), fazendo com que apenas 48 bits possam ser combinados, ou seja, apenas 2^{48} endereços distintos podem ser gerados.

Estratégia	Desperdício de Endereços	Limitação da Topologia
MHCL Top-down	Não há desperdício	Não há limitações
MHCL Bottom-up	Não há desperdício	Não há limitações
Prefixo Base 1	Não há desperdício	Número máximo de filhos: $64-k$ Altura máxima: $64/\min(k) = 64$
Estática	Não há desperdício	Número máximo de filhos: 2^k Altura máxima: $64/k$
Padrão IPv6	$2^{64} - 2^{48}$	Não há limitações

Tabela 4.1: Comparação entre as estratégias de endereçamento

4.2 Mensagens

O MHCL implementado como uma rotina do *ContikiRPL* utiliza a mesma estrutura das mensagens do RPL para comunicação ascendente e descendente. Dessa forma, os nomes foram mantidos e pequenas alterações nos pacotes foram realizadas.

Mensagens do tipo DIO_{MHCL} são enviadas em rotas descendentes, os seja, dos pais para os filhos. Essa mensagem é utilizada durante a fase do endereçamento, pois ela carrega as informações necessárias para um nó definir seu endereço e a faixa recebida de seu pai. O pacote é derivado do DIO_{RPL} [32], com o campo *flag* definido em 1 (no RPL o *flag* é definido em 0) e o endereço do nó filho (que é o primeiro endereço da faixa de IPv6 alocada para ele) e o tamanho da faixa alocada definido no campo *options* (veja Figura 4.7). Note que o tamanho do primeiro endereço e o tamanho da partição do endereço atribuído pode ter um comprimento pré-definido pela raiz, de acordo com o espaço de endereço global. Essa informação é suficiente para que o nó filho possa decodificar a mensagem e executar o procedimento de alocação de endereços para seus filhos. Mensagens do tipo $DIOACK_{MHCL}$ são identificadas pelo *flag* 2 e são usadas para confirmar o recebimento de uma faixa de endereços alocada por meio de uma mensagem DIO_{MHCL} . Se nenhuma confirmação é recebida (depois de um *timer* do tipo DIO_{min} se esgotar, veja Seção 4.3), então o pai tenta retransmitir a mensagem.

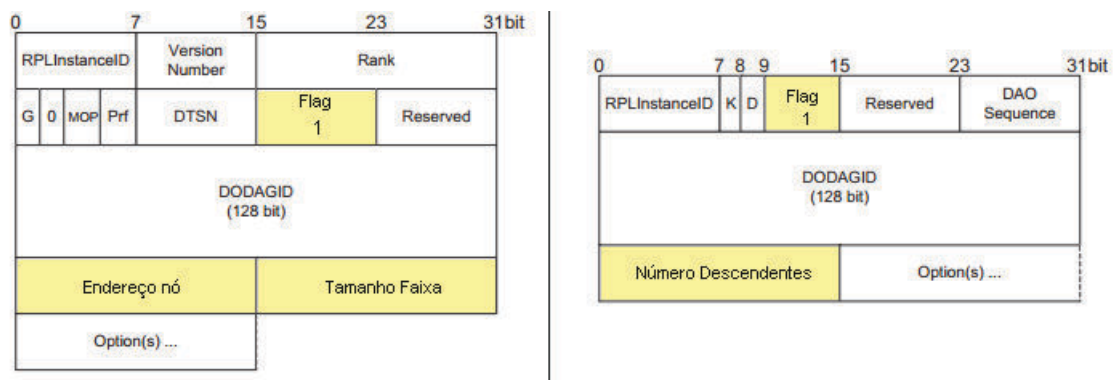


Figura 4.7: Estrutura da Mensagem DIO MHCL (esquerda) e DAO MHCL (direita).

Mensagens do tipo DAO_{MHCL} , assim como no RPL, são utilizadas para enviarem mensagens ascendentes na rede (de filho para pai). Essa mensagem possui duas funções: na abordagem Top-down ela informa ao pai preferido de um nó que este é seu filho; na abordagem Bottom-up, ele carrega o número de descendentes de um nó, usado na fase de agregação. A mensagem DAO_{RPL} foi modificada, alterando o campo *flag* para 1 e enviando o número de descendentes no campo *options* (Figura 4.7). Mensagens de confirmação de DAO ($DAOACK_{MHCL}$) são utilizadas no algoritmo Top-down. Uma vez que, enviando apenas uma mensagem para seu pai, cada nó termina a fase de coleta de informação da rede, é de extrema importância que essa informação seja recebida no nó destino. Para isso, uma mensagem de confirmação é utilizada, para habilitar retransmissões em caso de falhas ou colisões. Como no algoritmo Bottom-up existe uma atualização de informação temporizada, o envio de confirmação não é realizado. Assim como nas mensagens do tipo DIO, a mensagem $DAOACK_{MHCL}$ é diferenciada por possuir o campo *flag* definido em 2.

4.3 Comunicação e Temporizadores

O algoritmo MHCL Top-Down é composto pelas seguintes rotinas:

1. **Informar pai preferencial:** Se um nó não é raiz, ele deve informar a seu pai preferido que ele é um de seus filhos e precisa de um endereço, enviando uma mensagem DAO_{MHCL} . Um nó deve esperar sua lista de potenciais pais se estabilizar e só então informar ao pai escolhido. O MHCL implementa *timers* inspirados no *Trickle* para realizar tais decisões de estabilidade (Algoritmo 1). No Algoritmo 1, duas constantes são usadas: DIO_{min} é um *timer* usado pelo RPL para que um nó defina o menor intervalo para enviar mensagens DIO. pe_{Filho} é um *parâmetro de estabilização* usado para decidir quando o pai preferido está estabilizado (a

Tabela 5.1 contém os valores desses parâmetros usados nas emulações). Uma vez que a escolha do pai está estável, a variável local *definiuPai* é definida em *TRUE* e uma mensagem *DAO_{MHCL}* é enviada para o pai para informar sobre a decisão (linhas 12–18, Algoritmo 1).

2. **Contar filhos:** Cada nó deve receber mensagens (*DAO_{MHCL}*) de seus filhos, salvar uma lista com os filhos¹, atualizar o contador de filhos e confirmar o recebimento do pacote para cada filho, enviando um *DAOACK_{MHCL}*. O processo de contagem dura até a informação se tornar estável, o que é controlado pelo parâmetro *pePai*. Uma vez que o número de filhos se torna estável, a variável local *definiuFilhos* se torna verdadeira (linha 13, Algoritmo 2).
3. **Receber faixa de endereços:** Se um nó não é raiz, ele deve esperar até receber uma faixa de endereços alocada a ele por seu pai (em uma mensagem *DIO_{MHCL}*) e confirmar o recebimento enviando uma mensagem *DIOACK_{MHCL}*
4. **Enviar fatia de endereços para filhos:** O particionamento e alocação das faixas de endereços é iniciado pela raiz, uma vez que ela determina o número de filhos que possui. Assim que esse valor é conhecido (*definiuFilhos* é verdadeiro) e o espaço de endereços foi alocado a um nó não-raiz (recebeu *DIO_{MHCL}* do pai), a faixa de endereços disponível é particionada em porções iguais entre os filhos conhecidos, guardando uma reserva para futuros filhos, ou conexões atrasadas. Uma vez que o endereço é particionado, a fatia correspondente a cada filho é enviada em uma mensagem *DIO_{MHCL}*, como mostra o Algoritmo 3.
5. **Conexões atrasadas:** Se uma mensagem *DAO_{MHCL}* é recebida de um novo filho após a etapa de distribuição de endereços, então o processo de partição e alocação de endereços é repetido, usando a faixa reserva. Dessa forma, caso um nó tenha realizado o endereçamento de seus filhos e recebe uma mensagem solicitando um endereço IPv6, ele considera a faixa reserva como seu todo e realiza a divisão de acordo com a técnica implementada. Note que, dessa forma, ele também armazena uma porção de endereços reservas em cima dessa faixa.

A abordagem Bottom-Up do MHCL primeiramente executa um procedimento de agregação para computar o número total de descendentes de cada nó. Se um nó não é a raiz, e ele já definiu seu pai preferido (*definiuPai* é verdadeiro) ele começa enviando uma mensagem *DAO_{MHCL}* com o contador 0 (Algoritmo 4). Ao receber uma

¹Note que na implementação padrão de estruturas acíclicas em protocolos como RPL e CTP, um nó não salva uma lista de filhos, apenas uma lista de potenciais pais

Algorithm 1 MHCL: Timer Pai Preferido

```

1: definiuPai = FALSE;
2: maxTimer =  $peFilho * DIO_{min}$  ;
3: timer = rand( $1/2 * DIO_{min}, DIO_{min}$ ]; ▷ reseta timer
4: while NÃO definiuPai do
5:   if NÃO-RAIZ and ESTOUROU TIMER then
6:     if MUDOU-PAI then
7:       reseta timer;
8:     else
9:       if  $timer < maxTimer$  then
10:        timer *= 2; ▷ dobra o timer
11:      else
12:        definiuPai = TRUE; ▷ MHCL top-down e bottom-up
13:        if MHCL-Top-down then ▷ top-down
14:          envia  $DAO_{MHCL}$  para pai;
15:          if NÃO  $DAOACK_{MHCL}$  then
16:            envia  $DAO_{MHCL}$  para pai; ▷ retransmissão
17:          end if
18:        end if
19:      end if
20:    end if
21:  end if
22: end while

```

mensagem com o contador em 0, o nó automaticamente incrementa esse valor em 1, pois ele possui mais um descendente (o filho que enviou o DAO_{MHCL}). O nó então, após enviar seu primeiro DAO_{MHCL} espera por mensagens desse tipo de seus filhos, atualizando o número de descendentes a cada mensagem recebida e controlando o *timer* de estabilização. Se um nó é a raiz, então ele atualiza seu número de descendentes até seu número total estar estável (Algoritmo 5). Os parâmetros $peFolha$ e $peRaiz$ são usados para definir o critério de estabilização em nós não raiz e na raiz, respectivamente. Uma vez que a fase de agregação está completa, a variável local da raiz $definiuDescendentes$ é definida em verdadeiro, e o processo de alocação de endereços é começado (Algoritmo 3)

4.4 Tabelas de Roteamento e Encaminhamento

Após a alocação de endereços ser completada, cada nó (não folha) armazena uma tabela de roteamento para tráfego descendente, com uma entrada para cada filho. Cada entrada da tabela contém o endereço final na faixa alocada para o nó correspondente, e todas as entradas da tabela são ordenadas em ordem crescente de endereços finais

Algorithm 2 MHCL-Top-down: Timer do Contador de Filhos

```

1: definiuFilhos = FALSE;
2: maxTimer =  $pePai * DIO_{min}$ ;
3: timer = rand( $1/2 * DIO_{min}, DIO_{min}$ ]; ▷ reseta timer
4: count = 0; ▷ contador do número de filhos recebidos por mensagens  $DAO_{MHCL}$ 
5: while NÃO definiuFilhos do
6:   if ESTOUROU-TIMER then
7:     if MUDOU-CONTADOR then
8:       reseta timer;
9:     else
10:      if  $timer < maxTimer$  then
11:        timer *= 2;
12:      else
13:        definiuFilhos = TRUE;
14:      end if
15:    end if
16:  end if
17: end while

```

Algorithm 3 MHCL: Distribuição de endereços IPv6

```

1: ESTAVEL = FALSE;
2: if MHCL-Top-down then
3:   ESTAVEL = definiuFilhos;
4: else ▷ MHCL-Bottom-up
5:   ESTAVEL = definiuDescendentes or NÃO-RAIZ;
6: end if
7: if ESTAVEL and (É-RAIZ or RECEBEU- $DIO_{MHCL}$ ) then
8:   divide endereços disponíveis;
9:   for cada filho  $c_i$  do
10:    envia  $DIO_{MHCL}$  para  $c_i$ ; ▷ envia uma "fatia" IPv6
11:    if NÃO  $DIOACK_{MHCL}$  then
12:      envia  $DIO_{MHCL}$  para  $c_i$ ; ▷ retransmissão
13:    end if
14:  end for
15: end if

```

Algorithm 4 MHCL-Bottom-up: Timer de Agregação

```

1: maxTimerFolha =  $peFolha * DIO_{min}$ ;
2: timer = rand( $1/2 * DIO_{min}, DIO_{min}$ ]; ▷ reseta timer
3: count = 0; ▷ conta o número total de descendentes recebidos por mensagens
    $DAO_{MHCL}$ 
4: while NÃO- $DAO_{MHCL}$ -DE-PAI do
5: ▷ enquanto não recebeu uma fatia do endereço IPv6
6:   if NÃO-RAIZ and ESTOUROU-TIMER then
7:     if definiuPai and ( $count < 1$ ) then
8:       envia  $DAO_{MHCL}$  para pai;
9: ▷ dispara a agregação
10:    end if
11:    if MUDOU-COUNT then
12:      envia  $DAO_{MHCL}$  para pai;
13:      reseta timer;
14:    else
15:      if  $timer < maxTimerFolha$  then
16:        timer *= 2;
17:      end if
18:    end if
19:  end if
20: end while

```

Algorithm 5 MHCL-Bottom-up: Timer de Agregação (Raiz)

```

1: definiuDescendentes = FALSE;
2: maxTimerRaiz =  $peRaiz * DIO_{min}$ ;
3: timer = rand( $1/2 * DIO_{min}, DIO_{min}$ ]; ▷ reseta timer
4: count = 0; ▷ conta o número total de descendentes recebidos por mensagens
    $DAO_{MHCL}$ 
5: while NÃO definiuDescendentes do
6:   if É-RAIZ and ESTOUROU-TIMER then
7:     if MUDOU-COUNT then
8:       reseta timer;
9:     else
10:      if  $timer < maxTimerRaiz$  then
11:        timer* = 2;
12:      else
13:        definiuDescendentes = TRUE;
14:      end if
15:    end if
16:  end if
17: end while

```

de cada faixa. Dessa forma, o encaminhamento de mensagens pode ser realizado em tempo linear usando uma operação de comparação para cada entrada da tabela (veja Algoritmo 6). A pesquisa binária poderia ser utilizada, mas considerando que o tamanho da tabela de roteamento é limitado pelo número de filhos diretos de um nó, optamos pela simplicidade.

Algorithm 6 Encaminhamento de Mensagens

```

1:  $i = 0$ ;
2: while IPv6-destino  $>$  finalFaixaIP[i] do
3:      $i++$ 
4: end while
5: encaminha para filho[i]
```

4.5 Análise de Complexidade

A análise de complexidade de tempo ($Tempo(MHCL)$) e mensagens ($Mensagem(MHCL)$) é realizada considerando o modelo de comunicação síncrono com troca de mensagens ponto a ponto, ou seja, todos os nós começam a executar o algoritmo simultaneamente e que o tempo é dividido em voltas síncronas, tal que, quando uma mensagem é enviada por um nó v para seu vizinho u em um intervalo de tempo t , ela deve chegar em u antes de $t + 1$ (veja Seção 2.2). Note que o MHCL requer que uma topologia acíclica (uma árvore T) tenha sido construída pela rede antes de começar o endereçamento, ou seja, todo nó deve obrigatoriamente saber quem é seu pai preferencial ($definiuPai = TRUE$).

A abordagem Top-down apresenta duas fases: (1) uma rotina de comunicação de um salto, rodando em paralelo em todos os nós, na qual cada nó informa a seu pai preferencial sobre sua decisão. Ao ser informado, o pai deve enviar uma confirmação e, no final dessa rotina, cada nó sabe quantos filhos possui; (2) um *broadcast* da raiz para todos os nós na árvore T com informação de alocação de endereços, cada mensagem sendo confirmada pelo receptor da faixa de endereços. A complexidade de tempo da rotina (1) é 2, e a complexidade de mensagem é $2(n - 1)$. A complexidade de tempo da rotina (2) é $profundidade(T)$ e a complexidade de mensagem é $2(n - 1)$. A abordagem Bottom-up também possui duas fases: (1) agregação do número de descendentes, por meio de uma rotina *convergecast*, na qual nós folhas começam enviando mensagens para seus pais e os pais agregam a informação recebida e encaminham para cima até que a raiz é alcançada; (2) a raiz realiza um *broadcast* de informações sobre a alocação de endereços para baixo por toda a árvore T , cada mensagem sendo confirmada. A

complexidade de cada fase é $\text{profundidade}(T)$ e a complexidade de mensagem é $2(n - 1)$.

A complexidade total do MHCL no modelo síncrono com mensagens ponto a ponto é resumida pelo seguinte teorema:

Teorema 3. *Para qualquer rede de tamanho n com uma árvore geradora T com raiz no nó raiz, $\text{Mensagem}(\text{MHCL}(T, \text{raiz})) = O(n)$ e $\text{Tempo}(\text{MHCL}(T, \text{raiz})) = O(\text{profundidade}(T))$. Essa complexidade de mensagem e tempo é assintoticamente ótima.*

Prova: MHCL é composto de um *broadcast* e um *convergecast* por toda a árvore (na abordagem Bottom-up). Na operação *broadcast*, uma mensagem (com informação sobre a alocação de endereços) deve ser enviada para cada nó por seu respectivo pai preferencial, necessitando de $\Omega(n)$ mensagens. Além disso, a mensagem enviada pela raiz deve chegar a cada nó a uma distância de $\text{profundidade}(T)$ saltos, o que precisa de $\Omega(\text{profundidade}(T))$ intervalos de tempo. Da mesma forma, na operação *convergecast*, cada nó deve enviar uma mensagem para seu pai, depois de ter recebido uma mensagem de seus filhos, o que precisa de $\Omega(n)$ mensagens. Além disso, uma mensagem enviada por cada folha deve alcançar a raiz, em uma distância $\leq \text{profundidade}(T)$, o que requer $\Omega(\text{profundidade}(T))$ intervalos de tempo.

Note que, na realidade, os pressupostos de sincronia e entrega de mensagens ponto a ponto não se mantêm em uma rede 6LoWPAN. O momento em que cada nó se junta à rede, ou seja, atribui $\text{definiuPai} = \text{TRUE}$ varia de nó a nó, de tal forma que nós mais próximos à raiz tendem a começar a executar o protocolo de alocação de endereços antes do que nós mais distantes da raiz. Além disso, as colisões e falhas de nós e de conexões podem causar atrasos e impedir que mensagens sejam entregues. O desempenho do MHCL em um modelo assíncrono com colisões e nós transitórios e falhas de conexões é analisado através de emulações no Capítulo 5.

Capítulo 5

Resultados Experimentais

O protocolo MHCL apresenta características compatíveis com as necessidades de uma rede sem fio de baixa potência. Preparado para possuir um comportamento robusto em caso de falhas e com um algoritmo adaptado para as restrições da rede, ele se mostra uma boa alternativa para o endereçamento de nós em redes IPv6 *multihop*, auxiliando no roteamento e solucionando problemas em protocolos já existentes. A fim de comprovar esses resultados, emulações foram realizadas. O algoritmo do MHCL foi desenvolvido no sistema operacional Contiki [8], interno ao protocolo RPL. Os resultados apresentam uma comparação entre o protocolo RPL com sua implementação padrão, utilizando o endereçamento IPv6 gerado a partir de um identificador da interface e o RPL com o endereçamento do MHCL. Na Seção 5.1 são apresentadas as configurações das emulações. Em seguida, a inicialização da rede é avaliada para os algoritmos emulados (Seção 5.2), em função do tempo e da quantidade de mensagens enviadas. Além disso, a taxa de sucesso de endereçamento do MHCL é avaliada, na Seção 5.3. Por fim, são avaliadas as taxas de sucesso de entrega das mensagens de dados (algoritmo de aplicação), na Seção 5.4.

5.1 Configurações da Emulação

O MHCL foi implementado como uma sub-rotina do protocolo RPL no sistema operacional Contiki [8], substituindo o mecanismo de alocação de endereços padrão do IPv6, no qual o endereço de *host* de um nó é derivado de seu endereço MAC. Os resultados experimentais comparam a performance do RPL utilizando o MHCL Top-down e MHCL Bottom-up contra a alocação de endereços padrão (referenciada nos gráficos apenas como RPL). Para realizar os experimentos, foi utilizado o Cooja [10], um simulador 6LoWPAN fornecido pelo Contiki, com nó do tipo *SkyMote* com 10KB de RAM

e faixa de transmissão e interferência de 50 metros. Foram emuladas duas topologias: regular (com n nós distribuídos em um *grid* regular com 35m de distância entre cada nó) e uniforme (com nós uniformemente distribuídos aleatoriamente sobre uma área de lado $(\sqrt{n} - 2) * 35\text{m}$), com $n \in \{9, 25, 49, 81, 121, 169\}$. Também foram emulados dois tipos de falhas: no nó (TX) e na conexão (RX). As falhas são implementadas pelo simulador a nível de pacote. Em falhas do tipo TX, em $1 - TX\%$ de transmissões de pacotes, nenhum dispositivo recebe o pacote enviado, ou seja, é como se fosse uma falha no nó-transmissor. Em falhas do tipo RX, em $1 - RX\%$ de transmissões de pacotes, apenas o nó destino não recebe o pacote, ou seja, é como se a falha ocorresse no link. Nas emulações foram utilizados os valores de 90%, 95% e 100% para TX e RX, de maneira que quando uma das variáveis está em 90% ou 95%, a outra obrigatoriamente está em 100%.

Na camada de aplicação, nós utilizamos como base um exemplo disponível no Cooja (rpl-udp), que utiliza o protocolo UDP na camada de transporte, ou seja, não implementa retransmissão de pacotes, e RPL na camada de rede. Fizemos algumas modificações na aplicação, de forma que cada nó envie uma mensagem para a raiz e ela responda cada uma dessas mensagens. Dessa forma, a aplicação envia $n - 1$ mensagens ascendentes e $n - 1$ descendentes. Essas mensagens começam a ser trocadas após 180 segundos de emulação. A emulação de cada um dos algoritmos termina quando a última mensagem de aplicação é respondida, ou quando o tempo para esse evento ser completado se encerra, em caso de falhas.

A Tabela 5.1 apresenta os parâmetros dos algoritmos apresentados no Capítulo 4, que são usados para estimar o tempo de estabilização da rede da seguinte forma: cada temporizador usado no protocolo MHCL começa com um valor mínimo $I = 2^{DIO_{min}}$ ms., baseado no algoritmo *Trickle*, e um valor máximo, igual a $pe * 2^{DIO_{min}}$ ms., onde pe é o parâmetro de estabilização de cada rotina executada (veja os algoritmos apresentados no Capítulo 4).

Parâmetro	peFilho	pePai	peFolha	peRaiz	DIO_{min}
Valor	2	4	4	8	6

Tabela 5.1: Valor dos parâmetros de estabilização nos algoritmos

5.2 Inicialização da Rede

Para avaliar a inicialização da rede três parâmetros foram avaliados. O primeiro deles, o tempo de inicialização, leva em consideração quanto tempo a rede precisa para se

estruturar e obter as informações necessárias para realizar o roteamento. Os dois outros aspectos avaliados são a quantidade de mensagens do tipo DAO e DIO trocadas durante essa fase de inicialização. Como o tempo de inicialização é diferente entre os algoritmos, as mensagens são contadas durante os primeiros 180 segundos de emulação, para todos os casos. Os resultados obtidos nas emulações são apresentados a seguir. Na Seção 5.3 as duas alternativas de endereçamento do MHCL são comparadas de acordo com o sucesso no endereçamento dos nós na rede.

5.2.1 Tempo de Inicialização

Para avaliar o algoritmo MHCL, o tempo de inicialização foi definido como o tempo necessário para todos os nós serem endereçados, com exceção dos nós não endereçados devido à falhas. No protocolo RPL padrão, foi medido o tempo necessário para a raiz ter conhecimento de todas as $n - 1$ rotas (não necessariamente armazenar todas as rotas na tabela de roteamento descendente, pois a raiz pode não possuir memória suficiente para isso), ou seja, o tempo necessário para a raiz conhecer o caminho para todos os destinos existentes na rede.

As Figuras 5.1, 5.2, 5.3, 5.4 e 5.5 apresentam os gráficos com o tempo de inicialização de cada protocolo nos cenários com e sem falhas para as topologias regular e uniforme, respectivamente. Note que estamos lidando com algoritmos assíncronos e a inicialização de cada nó é aleatória, o que pode afetar o tempo de inicialização da rede, como é possível observar em alguns pontos a grande variação (intervalos de confiança muito grandes). Mesmo assim, é possível observar um crescimento linear em função da altura do DAG, como foi apresentado na análise de complexidade na Seção 4.5. Considerando um intervalo de confiança de 95%, podemos afirmar que na topologia regular, para 169 nós (com altura 24), o algoritmo Top-down se mostra mais rápido que os demais. O RPL em sua versão padrão e o Bottom-up são estatisticamente iguais quanto ao tempo de inicialização. Para a topologia uniforme, o RPL se mostra mais rápido. Foram realizados cálculos do Teste-t [18] para comparar os algoritmos MHCL Top-down e Bottom-up na topologia uniforme com 169 nós (e altura 10). A Tabela 5.2 apresenta o intervalo de confiança obtido e, como ele possui 0, podemos concluir que os algoritmos MHCL Top-down e Bottom-up apresentam o mesmo tempo de inicialização para a topologia uniforme com 169 nós.

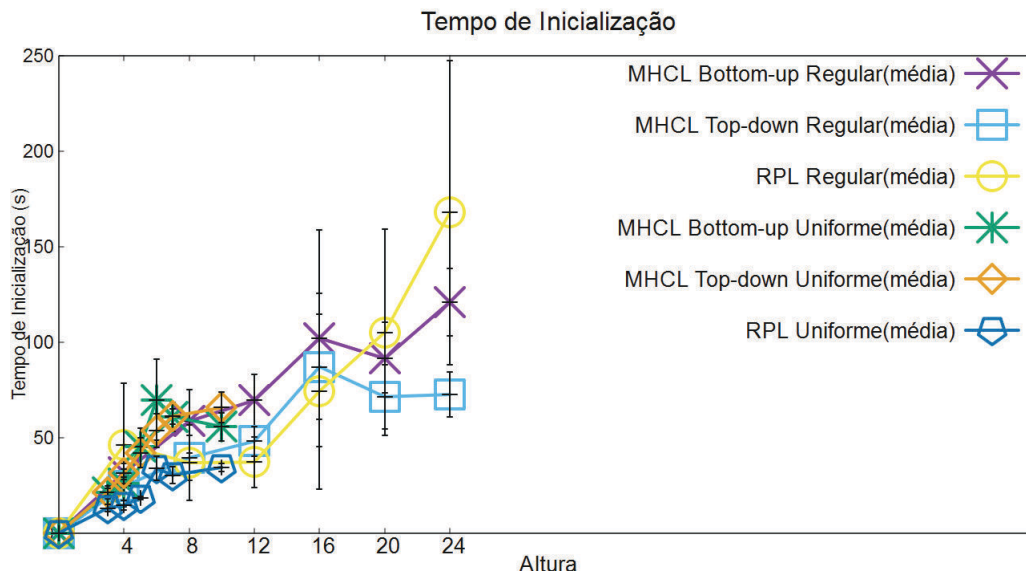


Figura 5.1: Tempo de inicialização em função da altura do DAG.

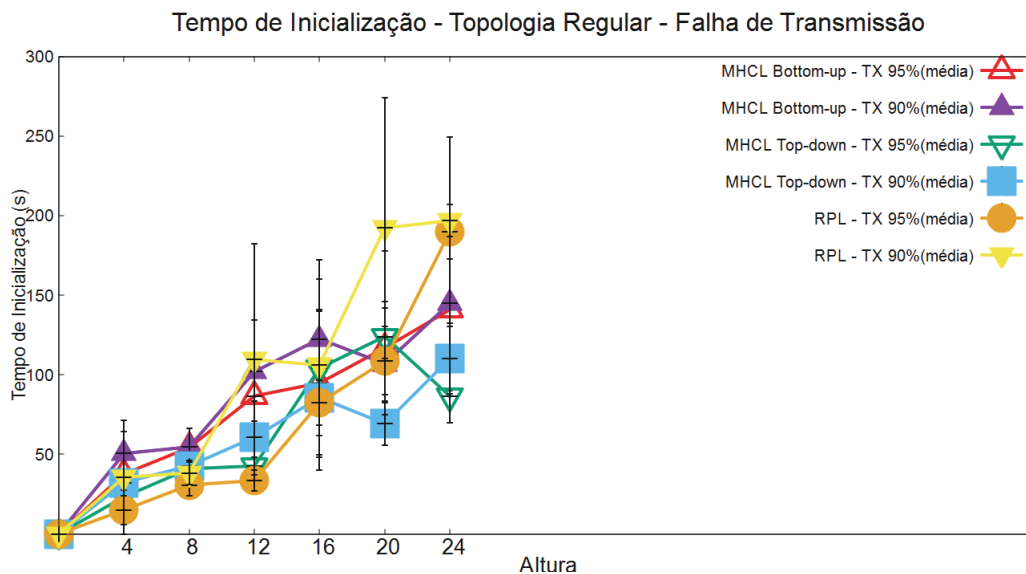


Figura 5.2: Tempo de inicialização em função da altura do DAG - Topologia Regular - Falha de Transmissão.

5.2.2 Número de Mensagens

Para avaliar a quantidade de mensagens enviadas por cada protocolo, foi definido um intervalo de 180 segundos. Durante esse intervalo, os números de mensagens DAO e DIO foram contados separadamente. No caso dos algoritmos que contam com retransmissões, mensagens de confirmação também foram contabilizadas (*DIOACK* é considerada *DIO*, por exemplo). As Figuras 5.6, 5.7, 5.8, 5.9, e 5.10 apresentam os resultados obtidos para as mensagens do tipo *DIO*. Mesmo o mecanismo de descoberta

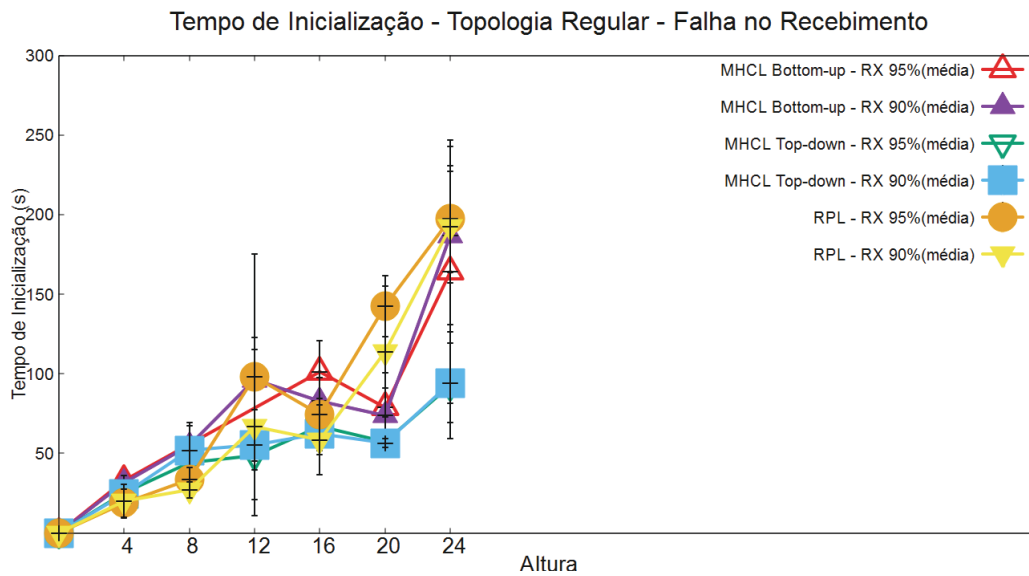


Figura 5.3: Tempo de inicialização em função da altura do DAG - Topologia Regular - Falha no Recebimento.

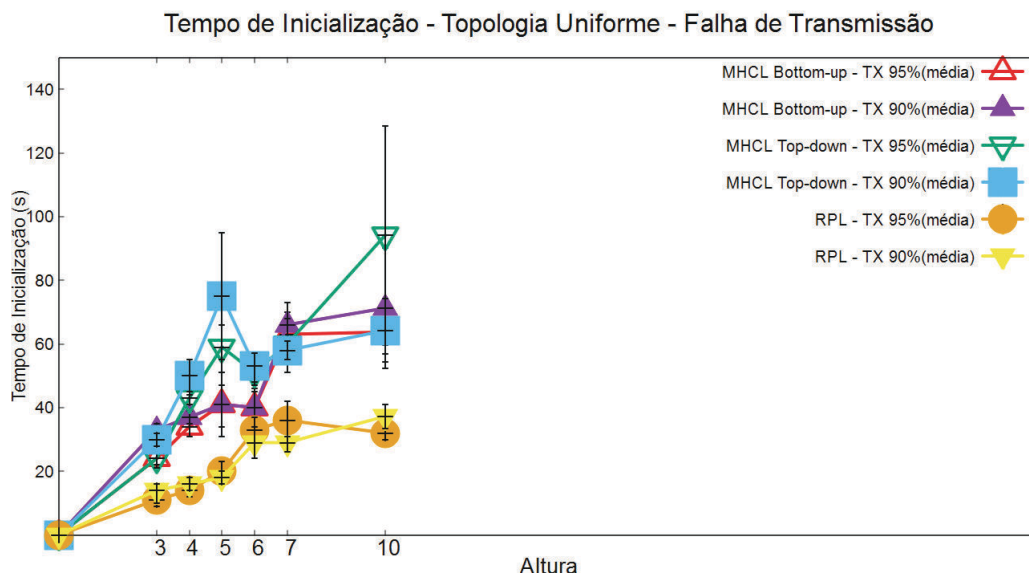


Figura 5.4: Tempo de inicialização em função da altura do DAG - Topologia Uniforme - Falha de Transmissão.

e manutenção do DAG não sendo alterado da implementação do RPL para a rotina do MHCL, é possível observar que, com 95% de confiança para 169 nós, o RPL envia mais mensagens do tipo DIO que o MHCL e a estratégia Top-down envia mais que a Bottom-up na topologia regular. Como mensagens desse tipo são utilizadas para a descoberta, formação e manutenção da topologia, é difícil determinar qual comportamento requereu um maior número de DIO sendo transmitidos no RPL em comparação com o MHCL. Além disso, é possível observar que esses resultados se mantêm em ambas as

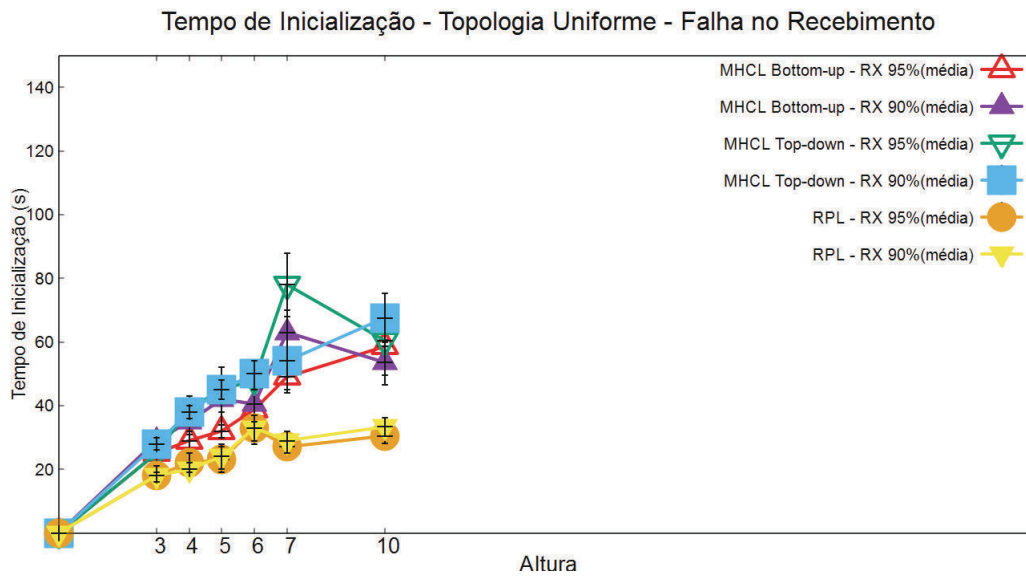


Figura 5.5: Tempo de inicialização em função da altura do DAG - Topologia Uniforme - Falha no Recebimento.

topologias e para todos os valores de TX e RX. Na topologia uniforme essa diferença é menor e, em alguns casos, os algoritmos enviam a mesma quantidade de mensagens do tipo *DIO* (uma vez que existe a sobreposição dos intervalos de confiança e a média de uma estratégia está contida no intervalo de confiança de outra).

Entretanto, como o mecanismo de descoberta de rotas, que utiliza mensagens DAO, é diferente no RPL e MHCL, o número de mensagens desse tipo enviadas tem uma diferença significativa. Isso pode ser visto nas figuras 5.11, 5.12, 5.13, 5.14, e 5.15. Com 95% de confiança podemos afirmar que o MHCL envia menos mensagens do tipo DAO do que o RPL, em todos os cenários emulados. Isso porque no MHCL existe um algoritmo de coleta de informações sobre a topologia que, uma vez encerrado, não envia mais mensagens DAO. O RPL realiza a notificação de rotas durante o funcionamento da rede, o que faz com que mensagens DAO sejam transmitidas continuamente para notificação de novas rotas. Também é possível observar que o algoritmo Top-down envia mais mensagens DAO que o Bottom-up. Isso pode ser explicado pelo fato de o algoritmo Top-down requerer o envio de confirmações (*DAOACK*) que são contabilizados, enquanto o Bottom-up não realiza o envio desse tipo de mensagem. Além disso, pode ser observado que, apesar de colisões e falhas nos nós e conexões, o número de mensagens DIO e DAO cresce linearmente com o número de nós, correspondendo com a complexidade de mensagens apresentada na Seção 4.5.

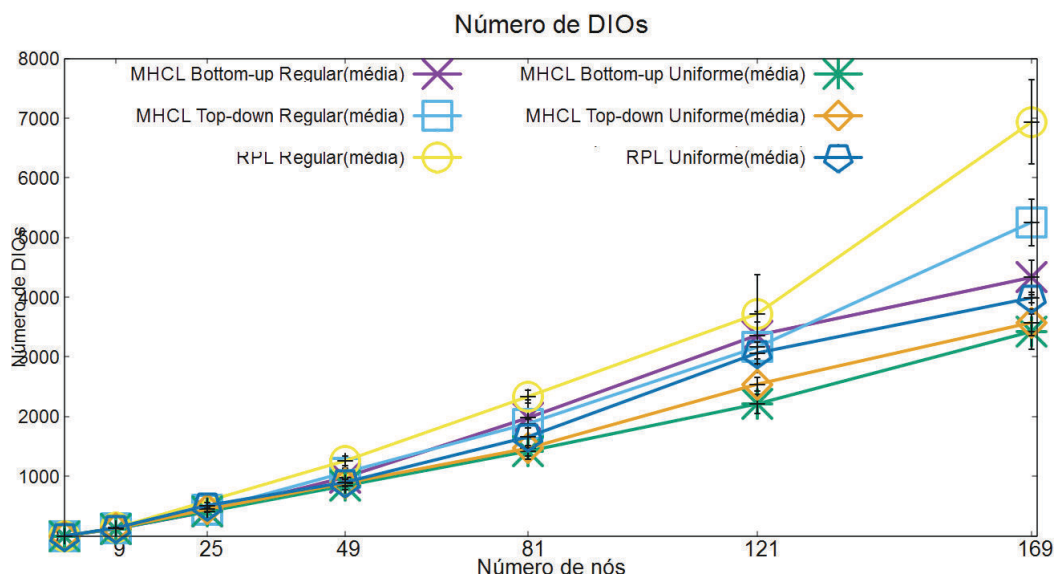


Figura 5.6: Número de DIOs em função do número de nós.

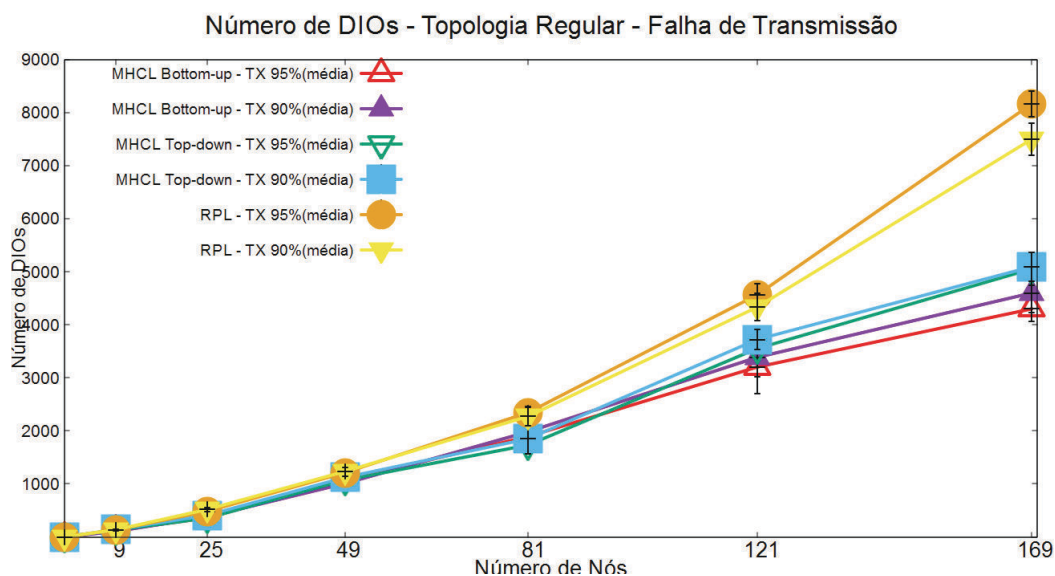


Figura 5.7: Número de DIOs em função do número de nós - Topologia Regular - Falha de Transmissão.

5.3 Sucesso de Endereçamento

O endereçamento dos nós pelo MHCL é feito durante a inicialização da rede. Como a rede pode apresentar colisões e falhas nos nós e nos links, algumas mensagens de endereçamento (do tipo DIO_{MHCL}) podem ser perdidas, mesmo usando mensagens de confirmação (do tipo $DIOACK_{MHCL}$), e realizando tentativas de retransmissão (nas emulações apresentadas, até 2 vezes). As Figuras 5.16, 5.17, 5.18, 5.19 e 5.20 apresentam os gráficos que comparam a taxa de sucesso de endereçamento de nós das

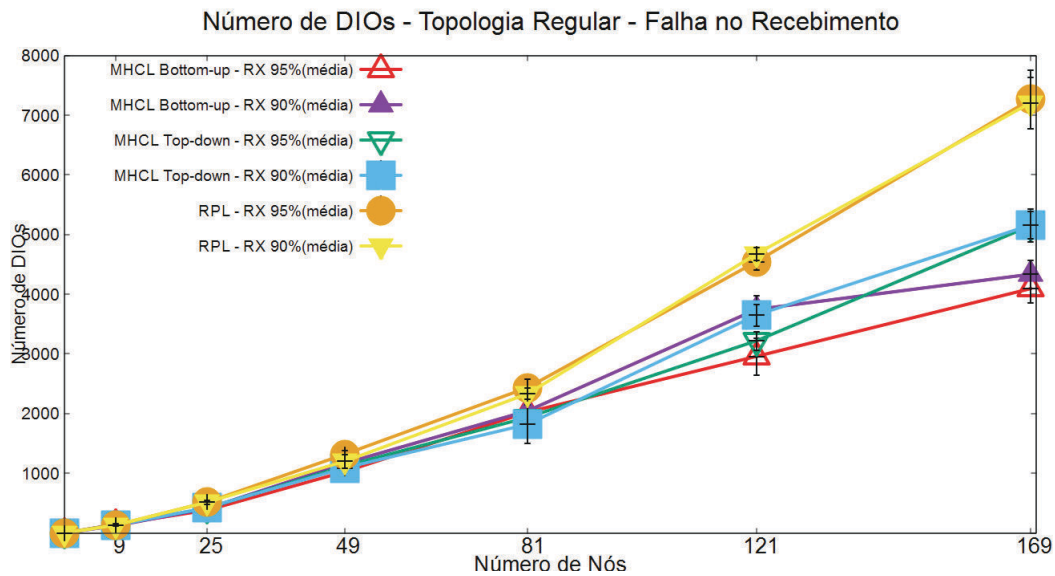


Figura 5.8: Número de DIOs em função do número de nós - Topologia Regular - Falha no Recebimento.

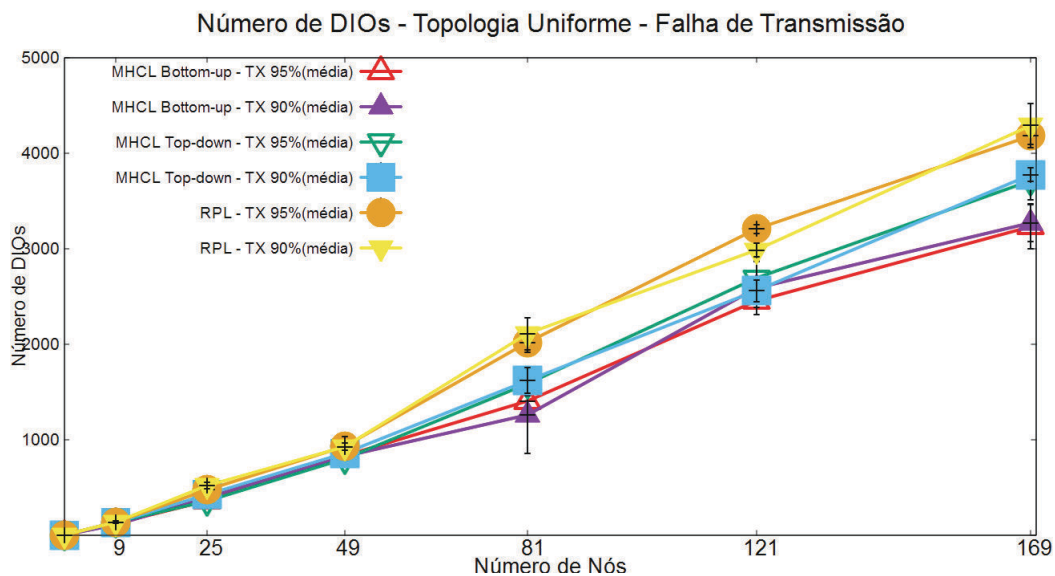


Figura 5.9: Número de DIOs em função do número de nós - Topologia Uniforme - Falha de Transmissão.

estratégias Bottom-up e Top-down. Ou seja, qual a porcentagem de nós na rede que recebe seu endereço IPv6 hierárquico correto. Podemos observar que, apesar de colisões e falhas intermitentes de nós e links, o sucesso de endereçamento se mantém entre 70% e 100% em ambas as estratégias. Como é possível observar, em média, a estratégia Top-down possui uma taxa de endereçamento superior à estratégia Bottom-up. Entretanto, quando existe a sobreposição dos intervalos de confiança e ambos contêm a média da outra estratégia na topologia regular, podemos afirmar que, estatisticamente, o

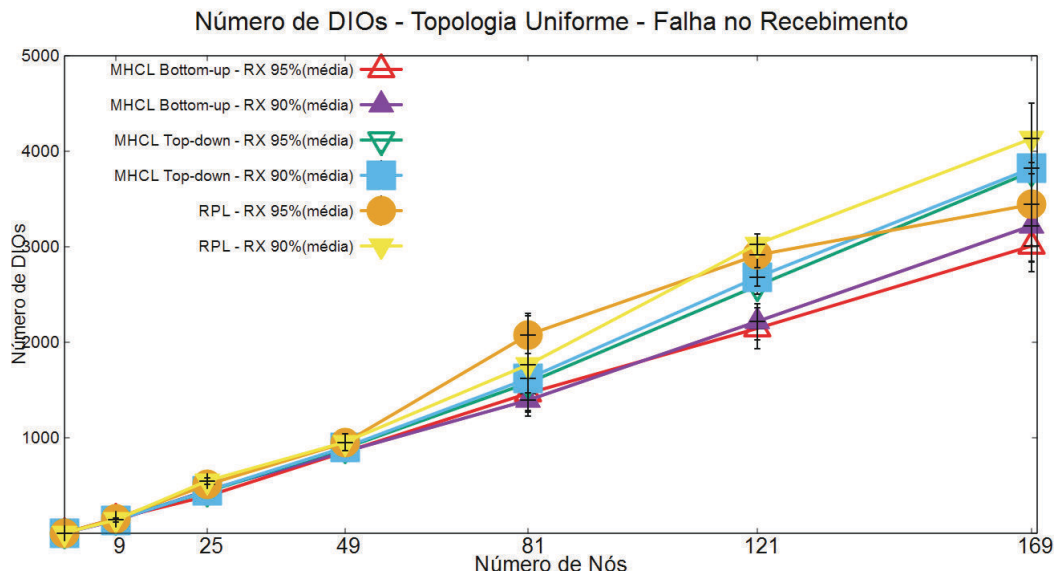


Figura 5.10: Número de DIOs em função do número de nós - Topologia Uniforme - Falha no Recebimento.

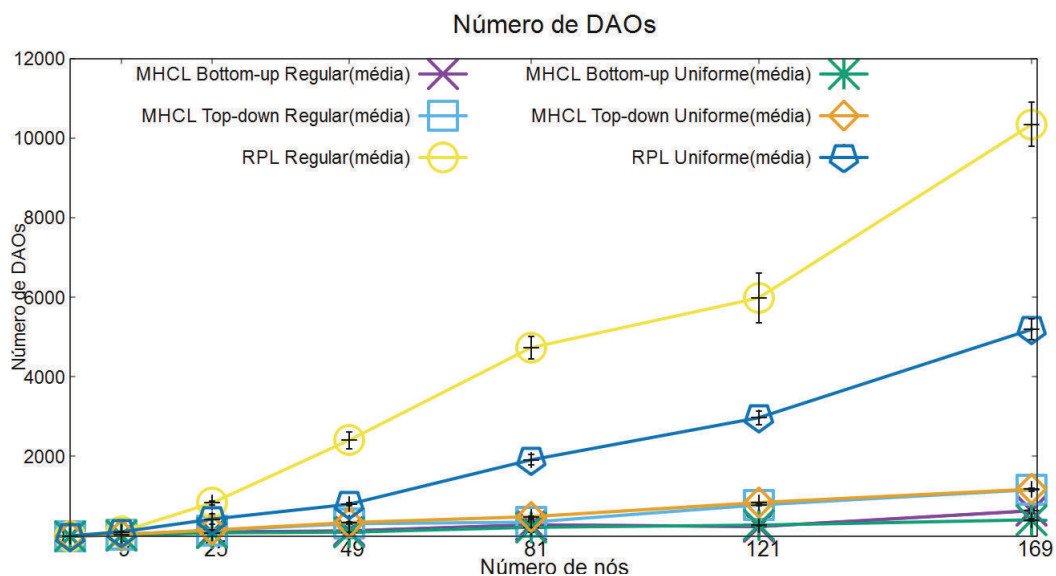


Figura 5.11: Número de DAOs em função do número de nós.

endereçamento de ambas as estratégias apresentam uma mesma taxa de sucesso. Já quando existe uma pequena sobreposição dos intervalos de confiança, na topologia uniforme, o teste-t mostra que a estratégia Top-down possui, estatisticamente, um sucesso de endereçamento igual a da estratégia Bottom-up, para 169 nós, como pode ser visto na Tabela 5.2.

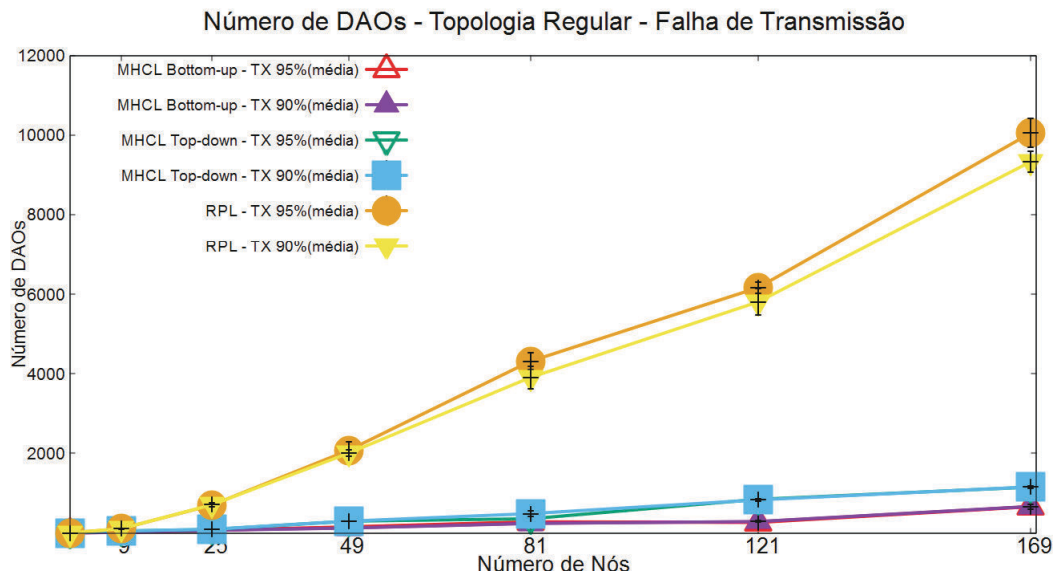


Figura 5.12: Número de DAOs em função do número de nós - Topologia Regular - Falha de Transmissão.

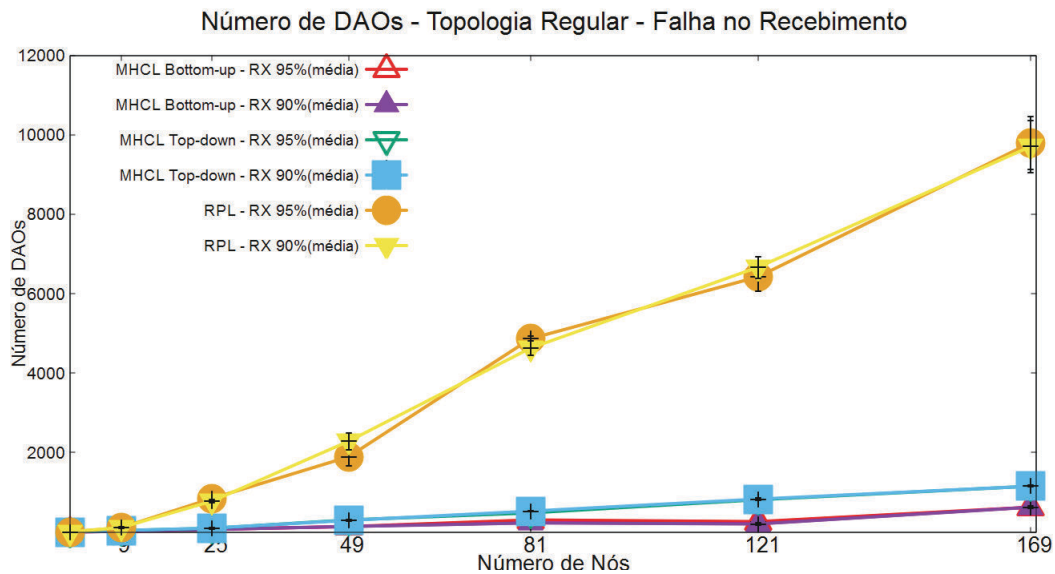


Figura 5.13: Número de DAOs em função do número de nós - Topologia Regular - Falha no Recebimento.

5.4 Sucesso de Entrega de Mensagens de Aplicação

Por fim, a taxa de sucesso de entrega de mensagens de aplicação ascendentes e descendentes foi calculada. Note que a camada de aplicação emulada não realiza confirmações e retransmissões de mensagens, portanto, uma colisão ou falha em um link ou nó pode causar perdas definitivas de mensagens. Para o caso de mensagens ascendentes (Figuras 5.21, 5.22, 5.23, 5.24 e 5.25), como o algoritmo de encaminhamento não foi alterado,

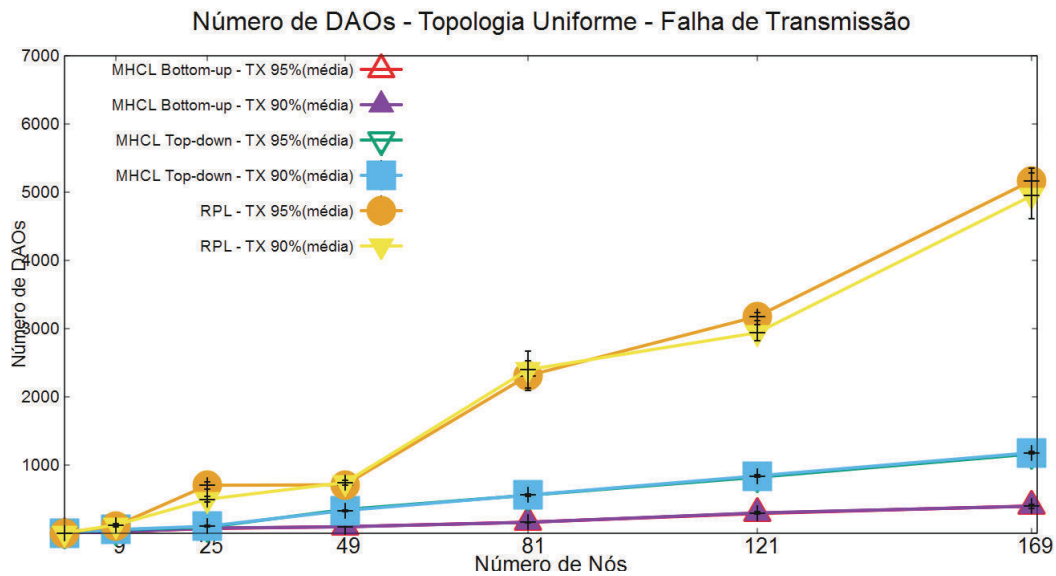


Figura 5.14: Número de DAOs em função do número de nós - Topologia Uniforme - Falha de Transmissão.

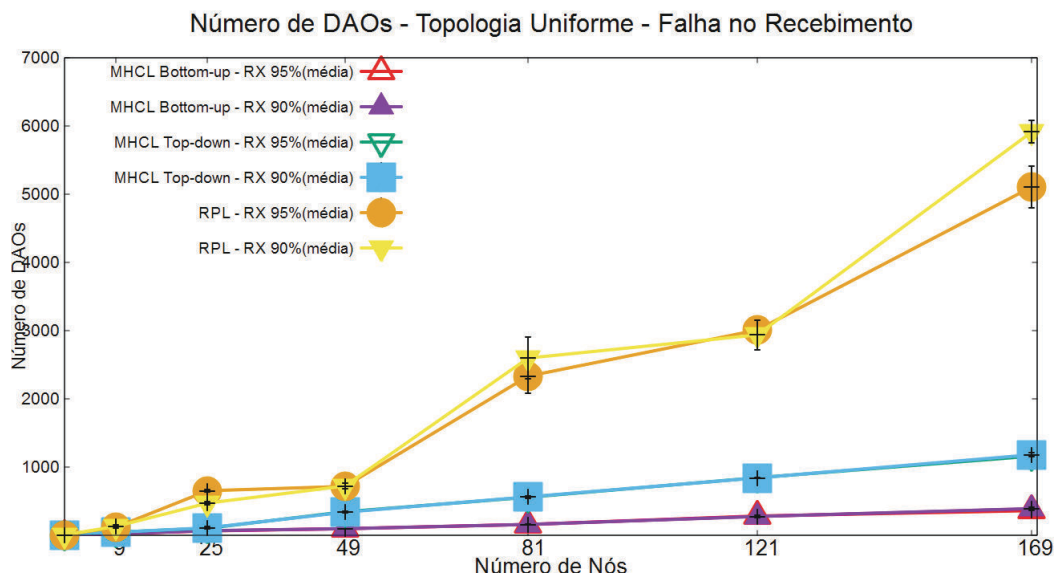


Figura 5.15: Número de DAOs em função do número de nós - Topologia Uniforme - Falha no Recebimento.

as estratégias apresentam valores próximos. Entretanto, como durante o algoritmo de aplicação o RPL ainda envia mensagens do tipo DAO, essas mensagens disputam o meio com as mensagens ascendentes da aplicação, aumentando a chance de colisões. No MHCL isso não ocorre. Com isso, apesar de ser possível observar uma diferença entre as médias, estatisticamente todas as estratégias apresentam a mesma taxa de sucesso de entrega de mensagens ascendentes, uma vez que existe sobreposição dos intervalos de confiança e, para o caso onde as médias não estão contidas no intervalo

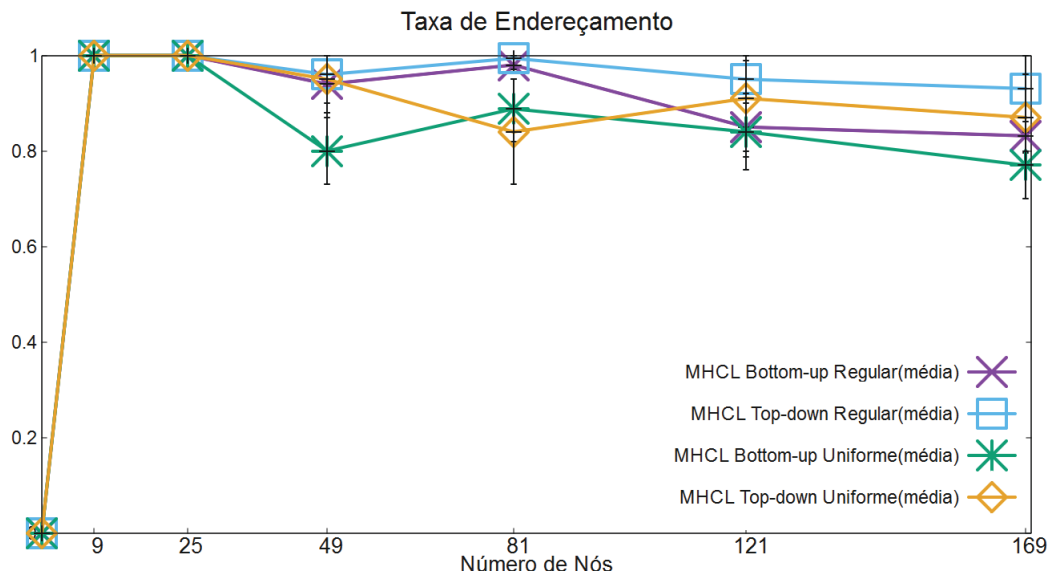


Figura 5.16: Sucesso de endereçamento em função do número de nós.

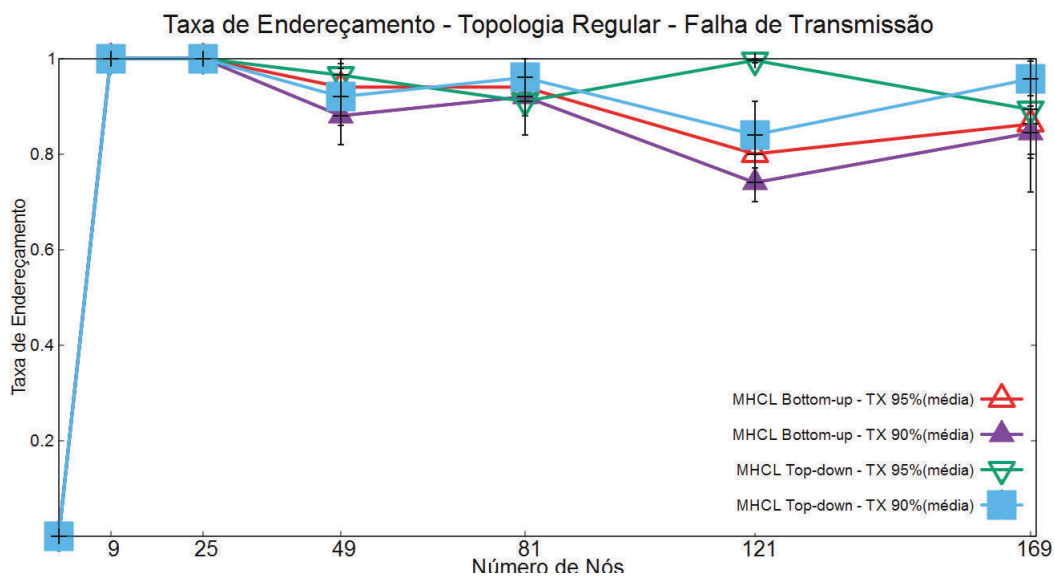


Figura 5.17: Sucesso de endereçamento em função do número de nós - Topologia Regular - Falha de Transmissão.

de confiança (Bottom-up e Top-down na topologia regular) o teste-t (veja Tabela 5.2) confirma essa igualdade.

Para o roteamento descendente (Figuras 5.26, 5.27, 5.28, 5.29 e 5.30) a diferença de desempenho entre MHCL e RPL é gritante. Como é possível observar, enquanto o RPL, para 169 nós, apresenta um sucesso de entrega de 15% em média, o MHCL chega a 80%. Com 95% de confiança, podemos afirmar que o MHCL possui uma taxa de entrega de mensagens descendentes superior ao RPL, e as estratégias bottom-up e top-down do MHCL são iguais, em relação a essa taxa. Isso ocorre pois, no RPL, os nós não têm

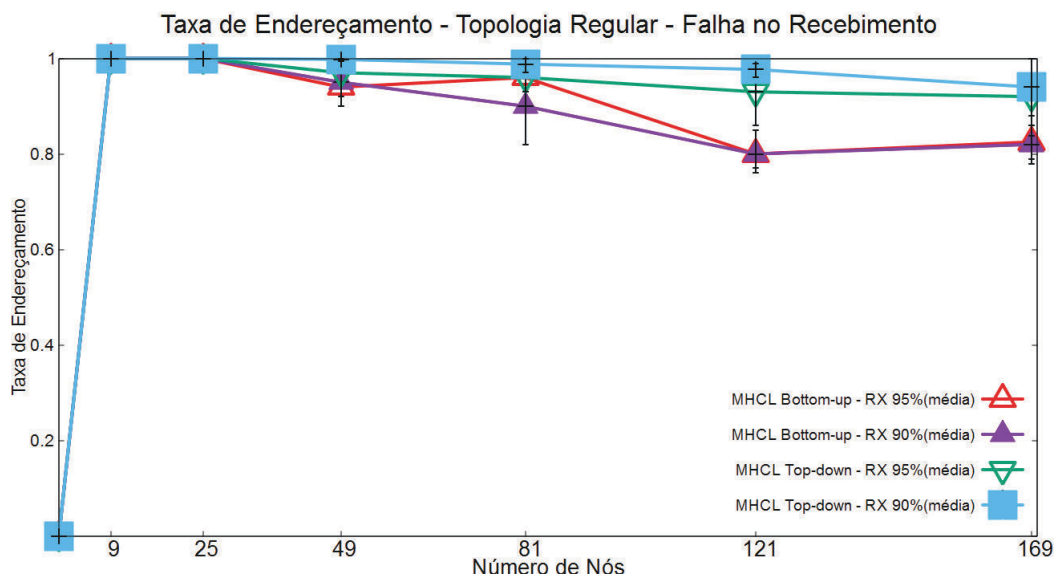


Figura 5.18: Sucesso de endereçamento em função do número de nós - Topologia Regular - Falha no Recebimento.

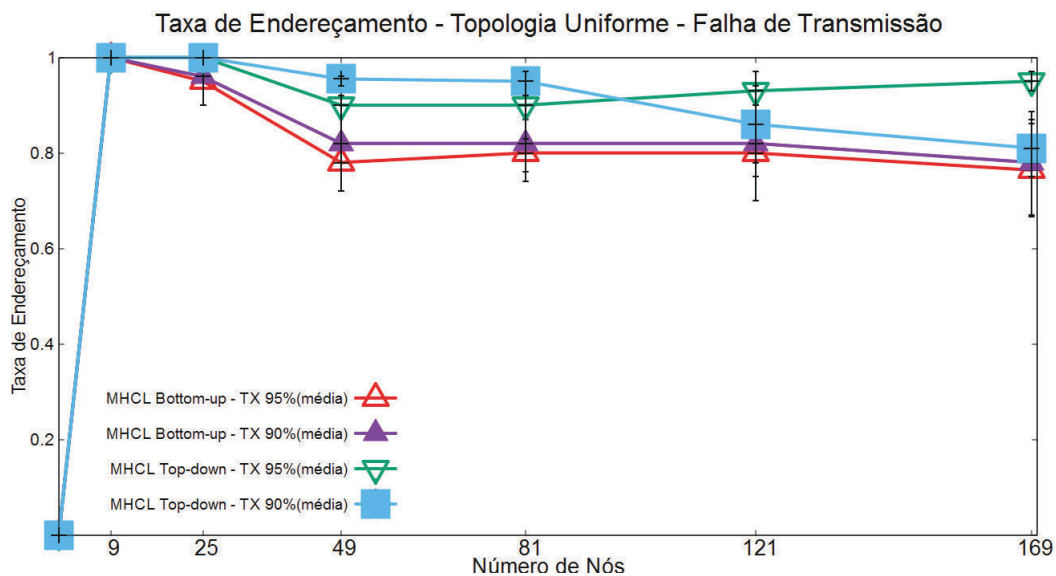


Figura 5.19: Sucesso de endereçamento em função do número de nós - Topologia Uniforme - Falha de Transmissão.

memória suficiente para armazenar as tabelas de roteamento necessárias para endereçar todos os seus descendentes, impossibilitando assim o roteamento de todas as mensagens, levando à um baixo sucesso no roteamento descendente. O MHCL apresenta uma taxa de entrega de mensagens mais de $3\times$ superior a do RPL. Como foi exposto na Figura 5.16, a taxa de endereçamento do MHCL é próxima de 90%. Portanto, as perdas de mensagens descendentes ocorrem devido a colisões e falhas em nós e links, somando-se o fato de não haver confirmações e retransmissões de mensagens na camada de aplicação

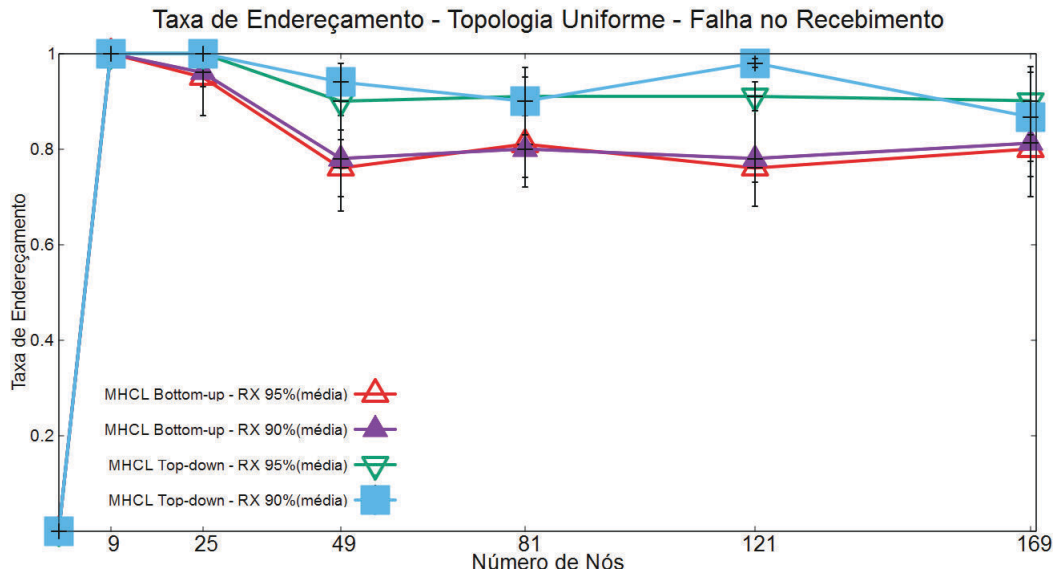


Figura 5.20: Sucesso de endereçamento em função do número de nós - Topologia Uniforme - Falha no Recebimento.

emulada. Note que, quando as falhas são do tipo TX, mais mensagens são perdidas do que quando as falhas são do tipo RX. Segundo implementação do simulador, em falhas do tipo TX a falha está na transmissão de pacotes, sendo que em $1 - TX\%$ de transmissões, nenhum dispositivo recebe o pacote enviado. Em falhas do tipo RX, em $1 - RX\%$ de transmissões de pacotes, apenas o nó destino não recebe o pacote, ou seja, é como se a falha ocorresse no *link*.

Quando comparamos o sucesso na entrega de mensagens, incluindo mensagens na fase de endereçamento, entre as topologias regular e uniforme, podemos observar que a uniforme apresenta um resultado um pouco melhor do que a regular. Isso pode ser explicado pelo fato de que a altura do DAG construído pelo RPL é menor na topologia uniforme, devido a uma maior variabilidade de distância entre os nós. Por exemplo, quando $n = 169$, na topologia regular, $altura(DAG) = 24$ mas, na topologia uniforme, $altura(DAG) = 10$, em média. Dado que as probabilidades de falhas e colisões se acumulam ao longo dos vários saltos do caminho seguido por cada mensagem, a perda geral de mensagem é maior na rede regular.

Parâmetro	Tempo de Inicialização	Sucesso de Endereçamento	Roteamento Ascendente
Topologia	Uniforme	Uniforme	Regular
Algoritmos	(Top-down x Bottom-up)	(Top-down x Bottom-up)	(Top-down x Bottom-up)
Teste-t	(-0.04 , 20)	(-0.01 , 0.21)	(-0.276 , 0.076)

Tabela 5.2: Teste-t para os valores que apresentaram sobreposição de intervalo de confiança mas a média de um não estava inclusa no intervalo de outro.

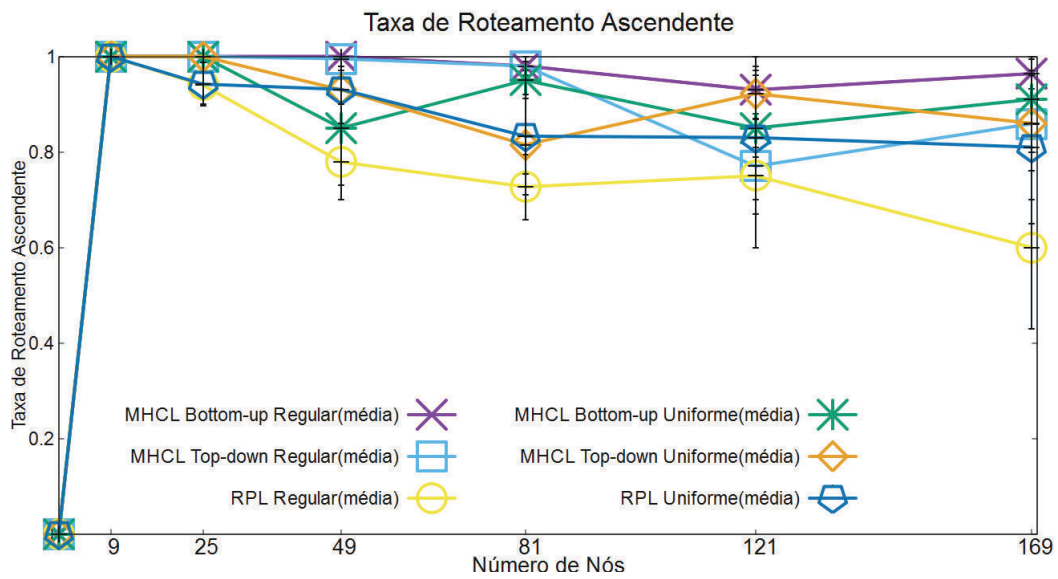


Figura 5.21: Sucesso de entrega de mensagens de aplicação ascendentes em função do número de nós.

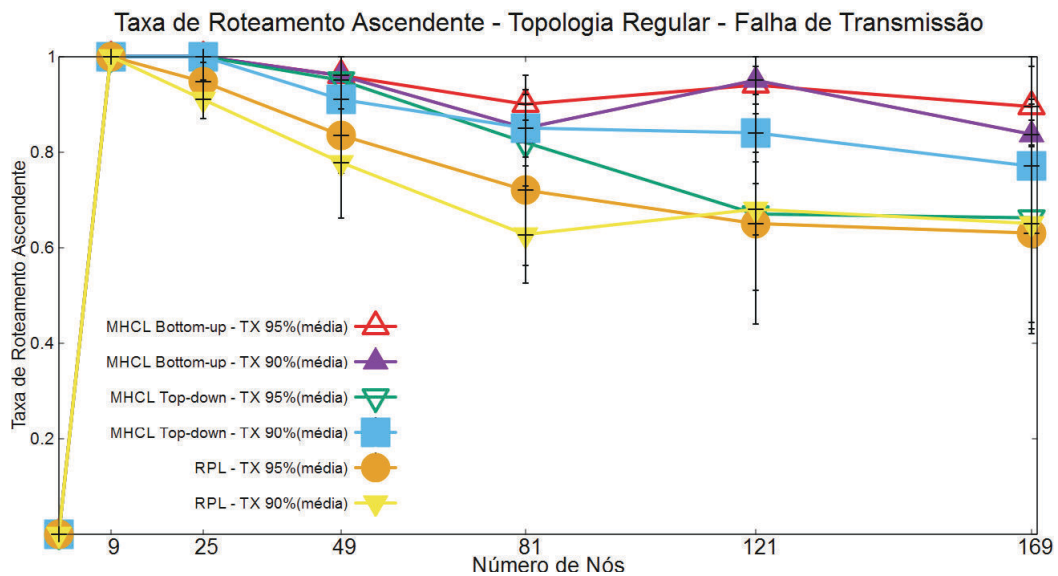


Figura 5.22: Sucesso de entrega de mensagens de aplicação ascendentes em função do número de nós - Topologia Regular - Falha de Transmissão.

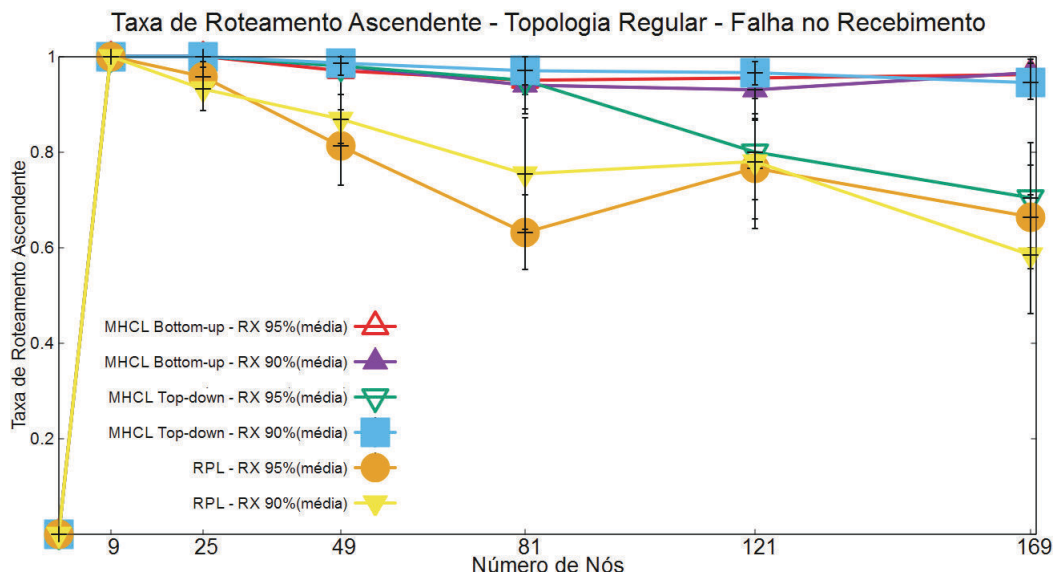


Figura 5.23: Sucesso de entrega de mensagens de aplicação ascendentes em função do número de nós - Topologia Regular - Falha no Recebimento.

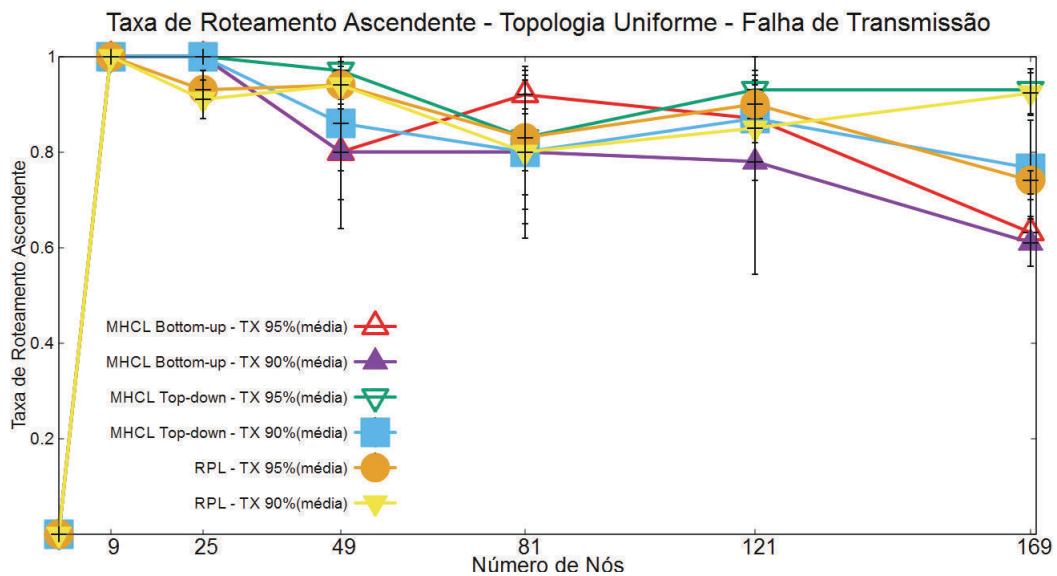


Figura 5.24: Sucesso de entrega de mensagens de aplicação ascendentes em função do número de nós - Topologia Uniforme - Falha de Transmissão.

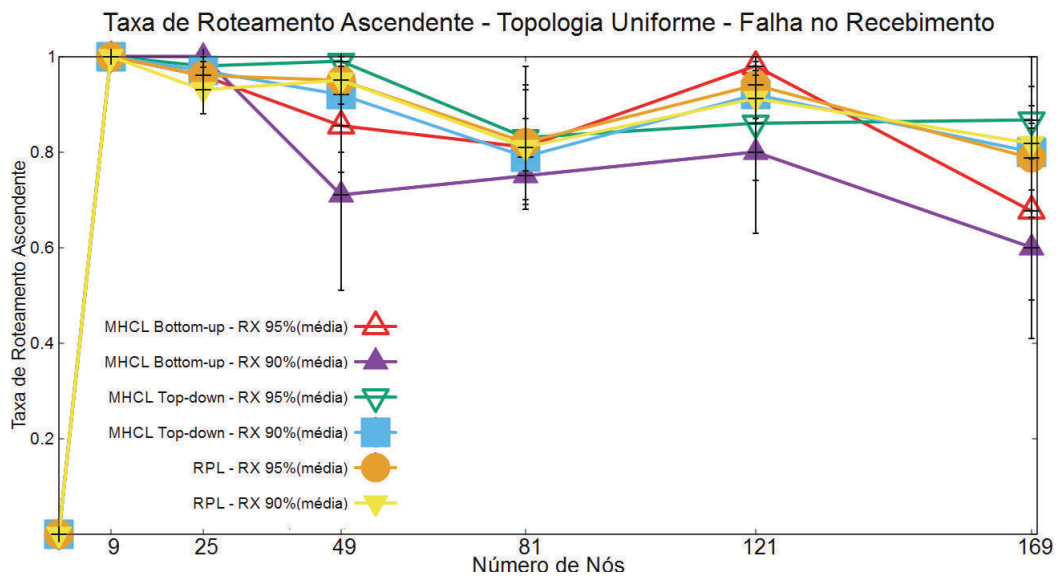


Figura 5.25: Sucesso de entrega de mensagens de aplicação ascendentes em função do número de nós - Topologia Uniforme - Falha no Recebimento.

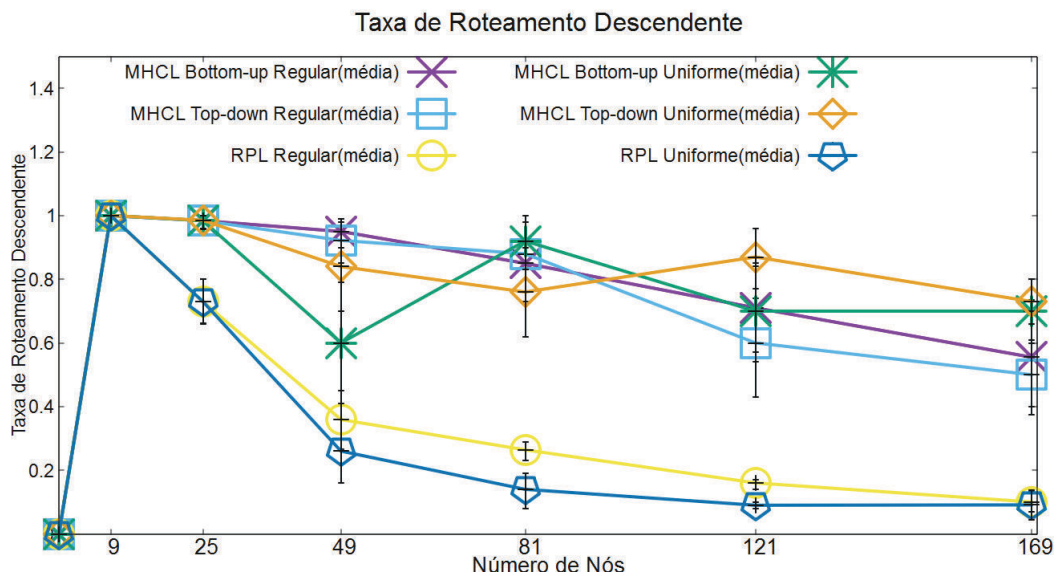


Figura 5.26: Sucesso de entrega de mensagens de aplicação descendentes em função do número de nós.

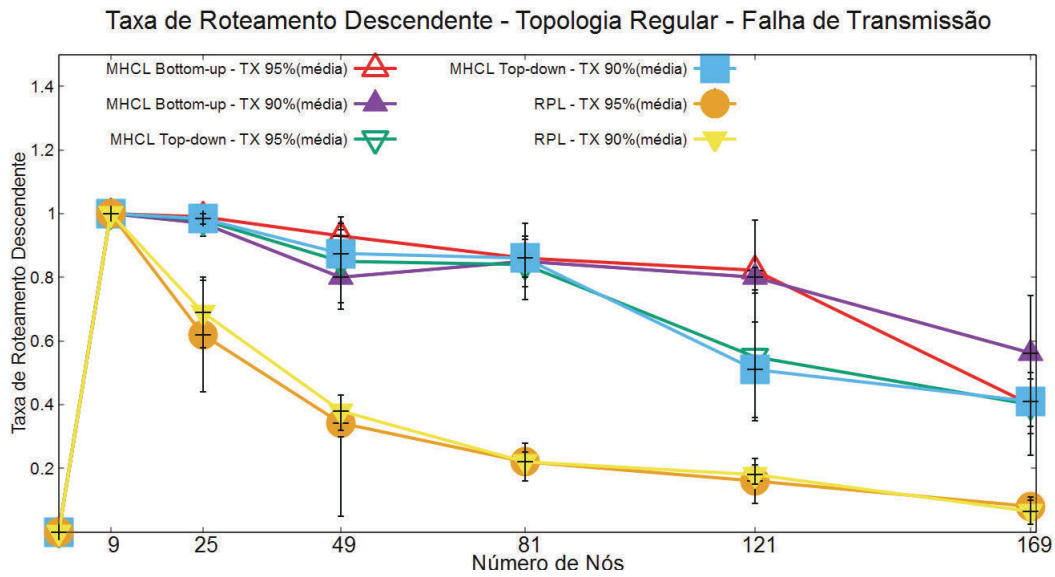


Figura 5.27: Sucesso de entrega de mensagens de aplicação descendentes em função do número de nós - Topologia Regular - Falha de Transmissão.

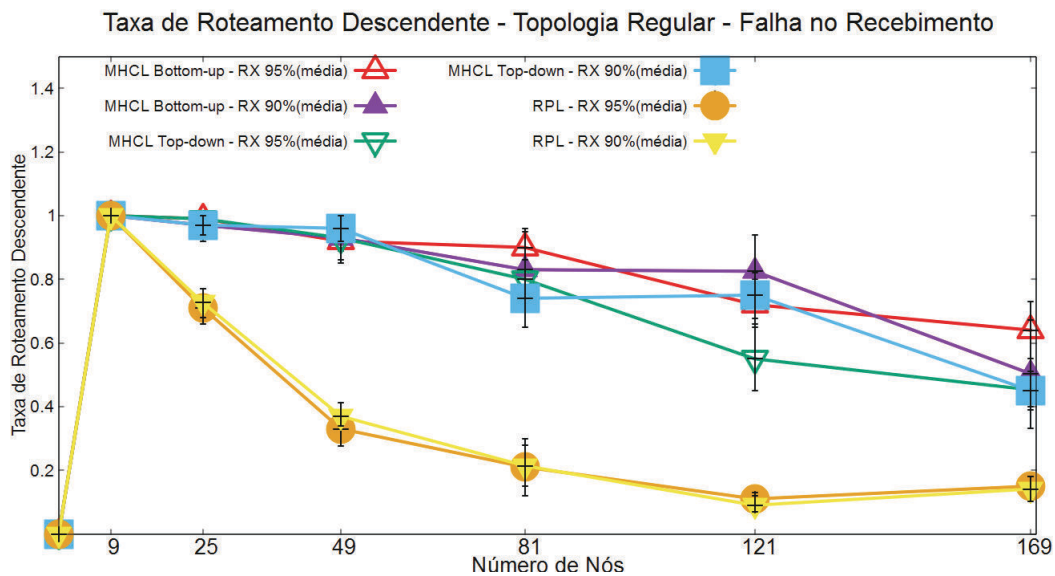


Figura 5.28: Sucesso de entrega de mensagens de aplicação descendentes em função do número de nós - Topologia Regular - Falha no Recebimento.

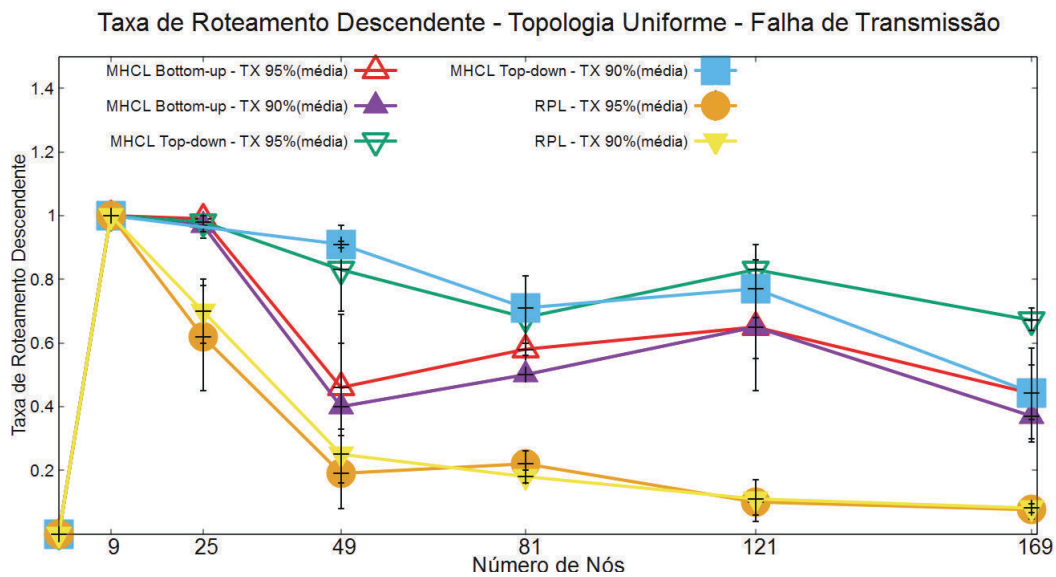


Figura 5.29: Sucesso de entrega de mensagens de aplicação descendentes em função do número de nós - Topologia Uniforme - Falha de Transmissão.

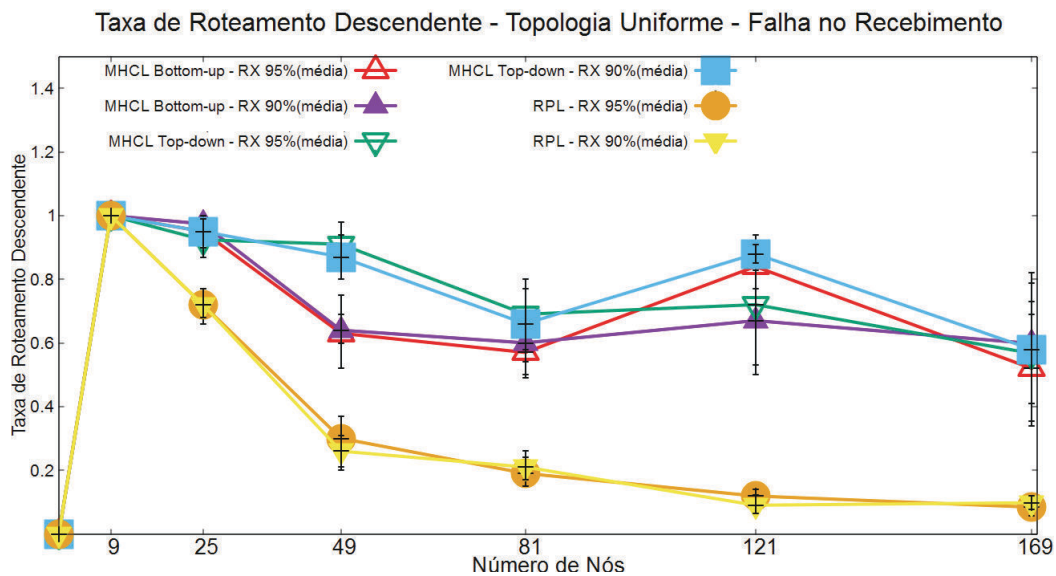


Figura 5.30: Sucesso de entrega de mensagens de aplicação descendentes em função do número de nós - Topologia Uniforme - Falha no Recebimento.

Capítulo 6

Conclusões e Trabalhos Futuros

6.1 Conclusões

O tráfego de dados de cima para baixo não é tipicamente o principal alvo das redes sem fio *multihop* de baixa potência. Entretanto, ele é ativado por alguns esquemas de roteamento populares como uma função de exceção, o que significa que as rotas são otimizadas para o tráfego de baixo para cima e mensagens descendentes podem seguir por rotas mais longas ou não serem entregues devido a restrições de memória. No protocolo RPL, por exemplo, é mantida uma topologia de rede direcionada e acíclica, chamada DAG (*Directed Acyclic Graph*), em que cada nó mantém na memória uma pequena lista de nós pais, ordenados pela qualidade da conexão, para os quais ele encaminha os dados em direção ascendente à raiz. Se um nó quer atuar não somente como uma fonte de dados, mas como destino, ele envia uma mensagem DAO (*Destination Advertisement Object*) para cima através do DAG, e os nós intermediários (se eles operam no assim chamado modo *storing*) adicionam uma entrada para este destino na tabela de roteamento criada especificamente para rotas descendentes. O tamanho de cada tabela deste tipo é potencialmente $O(n)$, onde n é o tamanho da sub-árvore com raiz em cada nó, ou seja, o número total de descendentes do nó. Dado que a capacidade de memória pode ser altamente restringida em redes sem fio de baixa potência, muitas vezes nem todas as rotas podem ser armazenadas e, conseqüentemente, pacotes serão descartados. Isso resulta na perda de um número elevado de mensagens e um alto consumo de memória. A fim de reduzir o tamanho da tabela de roteamento, seria desejável agregar vários endereços de nós de destinos próximos em uma única entrada da tabela. Entretanto, porque os endereços IPv6 têm os seus últimos 64 bits derivados do endereço de interface único de cada nó, eles são basicamente aleatórios e não contêm informações sobre a topologia da rede.

Neste trabalho foi proposto o MHCL—uma estratégia de alocação de endereços IPv6 para redes sem fio *multi-hop* de baixa potência. O MHCL foi implementado como uma rotina do protocolo RPL no sistema operacional *Contiki*. Duas estratégias de endereçamento foram desenvolvidas e testadas. A estratégia Top-down foca na eficiência, com o propósito de ser mais rápida e necessitar de um menor processamento de informações da topologia. A estratégia Bottom-up, por sua vez, utiliza uma função de agregação para contabilizar o número de nós na rede com a informação de quantos filhos e descendentes cada nó possui, para um endereçamento mais justo. Por meio de análise teórica e emulações, foi possível observar que o MHCL, com as duas estratégias, apresenta características favoráveis à dinâmica da rede e suas restrições.

As principais características do MHCL são: (1) Eficiência de memória: ao utilizar a topologia acíclica da rede para realizar um particionamento hierárquico do espaço de endereçamento disponível para um roteador de borda, os endereços de nós pertencentes a uma mesma subárvore são agrupados em uma única entrada na tabela de roteamento de um nó, tornando o tamanho da tabela $O(k)$, onde k é o número de filhos diretos do nó; isso se contrapõe ao tamanho $O(n)$ das tabelas de roteamento do RPL, onde n é o número total de descendentes de cada nó; (2) Eficiência temporal: as principais rotinas do MHCL são baseadas em temporizadores que rapidamente se adaptam à dinâmica da rede, possibilitando uma rápida fase de configuração de nós (ou seja, distribuição de endereços IPv6 aos mesmos); ao compararmos com o RPL, o tempo de inicialização da rede foi estatisticamente equivalente; (3) Eficiência em número de mensagens: o número de mensagens enviadas é adaptado à dinâmica da rede, através dos mesmos temporizadores acima; fazendo uma comparação com RPL, o número de mensagem do tipo DIO foi estatisticamente o mesmo e o de *DAOs* foi significativamente menor; (4) Eficiência de endereçamento: o MHCL consegue endereçar perto de 100% dos nós, mesmo em cenários com falha; (5) Eficiência na entrega de mensagens de aplicação: o roteamento descendente do MHCL se mostrou eficaz, entregando até 3× mais mensagens do que o RPL.

6.2 Trabalhos Futuros

Como trabalhos futuros, pretende-se analisar a robustez do MHCL a mudanças na topologia da rede que se estendam além de falhas intermitentes de nós e *links* individuais. Por meio de mecanismos de *local broadcast* pretende-se fazer com que uma mensagem enviada a um nó que se moveu na topologia consiga chegar a seu destino, por meio de poucos saltos. Além disso, com um monitoramento constante, utilizando mensagens

DIO, um nó pai pode ser capaz de definir se um filho se desconectou ou se moveu na topologia. A partir disso, ele pode adicionar seu filho à uma lista de filhos desconectados e, sempre que alguma mensagem for encaminhada para o endereço desse filho, ele deve avisar a seus vizinhos. Dessa forma, é esperado que com poucas retransmissões (uma vez que a topologia de grafo UDG (*Unit disk graph*) é utilizada na rede) uma mensagem chegue a seu destino, mesmo que ele tenha se movido na topologia. Note que o endereço desse nó que se moveu não é alterado e, com sua migração, ele não reflete mais sua posição na topologia. Por isso, é interessante analisar se tais migrações ocorrem com alta frequência, o que pode influenciar na eficiência do protocolo.

A análise teórica será estendida para modelos computacionais mais realistas, tais como o Modelo Físico de Interferência (SINR) , que representa de forma mais realista a propagação de sinal de rádio, interferência e colisões entre as mensagens, e modelos mais gerais de computação distribuídas, tais como Modelo Assíncrono e o Modelo com Falhas Intermitentes, como apresentado em [28]. Além disso, os algoritmos serão implementados em dispositivos físicos reais que rodam o sistema operacional Contiki, como o *OpenMote* [Ope]. Por fim, pretende-se implementar uma aplicação que utilize esses nós sensores, como, por exemplo, uma aplicação de *Smart Buildings* que controle o consumo de energia em um prédio através de medidores de corrente elétrica acoplados às tomadas.

Referências Bibliográficas

- [con] Contiki: The open source os for the internet of things. <http://www.contiki-os.org/>.
- [iet] The internet engineering task force (ietf).
- [Ope] Openmote, open hardware for the internet of things.
- [4] Boleng, J. (2000). Efficient network layer addressing for mobile ad hoc networks. Em *in Proc. of International Conference on Wireless Networks (ICWN02), Las Vegas*, pp. 271--277.
- [5] Clausen, T.; Herberg, U. & Philipp, M. (2011). A critical evaluation of the ipv6 routing protocol for low power and lossy networks (rpl). Em *Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on*, pp. 365--372.
- [6] Clausen, T.; Yi, J.; Herberg, U. & Igarashi, Y. (2013). Observations of rpl: Ipv6 routing protocol for low power and lossy networks. draft-clausen-lln-rpl-experiences-05.
- [7] da Silva Santos, E. R.; Vieira, M. A. & Vieira, L. F. (2013). Routing ipv6 over wireless networks with low-memory devices. Em *Personal Indoor and Mobile Radio Communications (PIMRC), 2013 IEEE 24th International Symposium on*, pp. 2398--2402.
- [8] Dunkels, A.; Gronvall, B. & Voigt, T. (2004). Contiki - a lightweight and flexible operating system for tiny networked sensors. Em *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, LCN '04*, pp. 455--462, Washington, DC, USA. IEEE Computer Society.
- [9] Erciyes, K. (2013). *Distributed Graph Algorithms for Computer Networks*. Springer Publishing Company, Incorporated.

- [10] Eriksson, J.; Österlind, F.; Finne, N.; Tsiftes, N.; Dunkels, A.; Voigt, T.; Sauter, R. & Marrón, P. J. (2009). Cooja/mspsim: Interoperability testing for wireless sensor networks. Em *Proceedings of the 2Nd International Conference on Simulation Tools and Techniques*, Simutools '09, pp. 27:1--27:7, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- [11] Gnawali, O.; Fonseca, R.; Jamieson, K.; Moss, D. & Levis, P. (2009). Collection tree protocol. Em *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pp. 1--14, New York, NY, USA. ACM.
- [12] Goyal, M.; Baccelli, E.; Philipp, M.; Brant, A. & Martocci, J. (2013). Reactive discovery of point-to-point routes in low power and lossy networks. IETF Internet Draft, draft-ietfroll-p2p-rpl-17.
- [13] Hinden, R. & Deering, S. (2006). Ip version 6 addressing architecture. Network Working Group, Request for Comments: 4291.
- [14] Hui, J. & Culler, D. (2010). Ipv6 in low-power wireless networks. *Proceedings of the IEEE*, 98(11):1865–1878.
- [IEEE] IEEE. Standard group mac addresses: A tutorial guide.
- [16] IEEE (1997). Guidelines for 64-bit global identifier (eui-64) registration authority.
- [17] J. Hui, E. (2009). Compression format for ipv6 datagrams in 6lowpan networks. tools.ietf.org/html/draft-ietf-6lowpan-hc-06.
- [18] Jain, R. (1991). *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling*. Wiley professional computing. Wiley.
- [19] Ko, J.; Dawson-Haggerty, S.; Gnawali, O.; Culler, D. & Terzis, A. (2011). Evaluating the Performance of RPL and 6LoWPAN in TinyOS. Em *Proceedings of Extending the Internet to Low power and Lossy Networks (IPSN 2011)*.
- [20] Ko, J.; Jeong, J.; Park, J.; Jun, J. A. & Kim, N. (2012). Towards full rpl interoperability: Addressing the case with downwards routing interoperability. Em *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, pp. 353--354, New York, NY, USA. ACM.

- [21] Kuhn, F.; Locher, T. & Wattenhofer, R. (2008). Distributed selection: A missing piece of data aggregation. *Commun. ACM*, 51(9):93--99.
- [22] Lamport, L. & Lynch, N. (1989). Chapter on distributed computing.
- [23] Levis, P.; Brewer, E.; Culler, D.; Gay, D.; Madden, S.; Patel, N.; Polastre, J.; Shenker, S.; Szewczyk, R. & Woo, A. (2008). The emergence of a networking primitive in wireless sensor networks. *Commun. ACM*, 51(7):99--106.
- [24] Levis, P.; Madden, S.; Polastre, J.; Szewczyk, R.; Whitehouse, K.; Woo, A.; Gay, D.; Hill, J.; Welsh, M.; Brewer, E. & Culler, D. (2005). TinyOS: An Operating System for Sensor Networks. Em Weber, W.; Rabaey, J. & Aarts, E., editores, *Ambient Intelligence*, capítulo 7, pp. 115--148. Springer Berlin Heidelberg, Berlin/Heidelberg.
- [25] Levis, P.; Patel, N.; Culler, D. & Shenker, S. (2004). Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. Em *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, NSDI'04, pp. 2--2, Berkeley, CA, USA. USENIX Association.
- [26] Liu, Y.; He, Y.; Li, M.; Wang, J.; Liu, K. & Li, X.-Y. (2013). Does wireless sensor network scale? a measurement study on greenorbs. *IEEE Trans. Parallel Distrib. Syst.*, 24(10):1983--1993.
- [27] P. Levis, T. Clausen, J. H. O. G. J. K. (2011). The trickle algorithm. <https://tools.ietf.org/html/draft-ietf-roll-trickle-08>.
- [28] Peleg, D. (2000). *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.
- [29] Ruiz-Sanchez, M. A.; Biersack, E. W. & Dabbous, W. (2001). Survey and taxonomy of ip address lookup algorithms. *Netw. Mag. of Global Internetwkg.*, 15(2):8-23.
- [30] Sun, Y. & Belding-Royer, E. M. (2004). A study of dynamic addressing techniques in mobile ad hoc networks: Research articles. *Wirel. Commun. Mob. Comput.*, 4(3):315--329.
- [31] Tolle, G.; Polastre, J.; Szewczyk, R.; Culler, D.; Turner, N.; Tu, K.; Burgess, S.; Dawson, T.; Buonadonna, P.; Gay, D. & Hong, W. (2005). A microscope in the redwoods. Em *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, SenSys '05, pp. 51--63, New York, NY, USA. ACM.

- [32] Tsvetko, T. (2011). Rpl: Ipv6 routing protocol for low power and lossy networks. Em *Seminar SensorKnoten: Betrieb, Netze und Anwendungen SS*.
- [33] Upadhyayula, S.; Annamalai, V. & Gupta, S. (2003). A low-latency and energy-efficient algorithm for convergecast in wireless sensor networks. Em *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, volume 6, pp. 3525–3530 vol.6.
- [34] Vasseur, J. (2008). Terminology in low power and lossy networks. disponível em <http://tools.ietf.org/html/draft-vasseur-roll-terminology-01>.
- [35] Vasseur, J.; Agarwal, N.; Hui, J.; Shelby, Z.; Bertrand, P. & Chauvenet, C. (2011). Rpl: The ip routing protocol designed for low power and lossy networks. Internet Protocol for Smart Objects (IPSO) Alliance.
- [36] Vasseur, J.-P. & Dunkels, A. (2010). *Interconnecting Smart Objects with IP: The Next Internet*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- [37] Vicaire, P.; He, T.; Cao, Q.; Yan, T.; Zhou, G.; Gu, L.; Luo, L.; Stoleru, R.; Stankovic, J. A. & Abdelzaher, T. F. (2009). Achieving long-term surveillance in vigilnet. *ACM Trans. Sen. Netw.*, 5(1):9:1--9:39.
- [38] Werner-Allen, G.; Lorincz, K.; Johnson, J.; Lees, J. & Welsh, M. (2006). Fidelity and yield in a volcano monitoring sensor network. Em *Proceedings of the 7th Symposium on Operating Systems Design and Implementation, OSDI '06*, pp. 381--396, Berkeley, CA, USA. USENIX Association.
- [39] Winter, T.; Thubert, P.; Brandt, A.; Hui, J.; Kelsey, R.; Levis, P.; Pister, K.; Struik, R.; Vasseur, J. & Alexander, R. (2012). RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550 (Proposed Standard).
- [40] Xie, W.; Goyal, M.; Hosseini, H.; Martocci, J.; Bashir, Y.; Baccelli, E. & Durrezi, A. (2010). A performance analysis of point-to-point routing along a directed acyclic graph in low power and lossy networks. Em *Network-Based Information Systems (NBIS), 2010 13th International Conference on*, pp. 111–116.
- [41] Yong, L.; Ping, Z. & JiaXiong, L. (2009). Dynamic address allocation protocols for mobile ad hoc networks based on genetic algorithm. Em *Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on*, pp. 1–4.

- [42] Zhang, T. & Li, X. (2014). Evaluating and analyzing the performance of rpl in contiki. Em *Proceedings of the First International Workshop on Mobile Sensing, Computing and Communication*, MSCC '14, pp. 19--24, New York, NY, USA. ACM.
- [43] Zhao, J.; Govindan, R. & Estrin, D. (2003). Computing aggregates for monitoring wireless sensor networks.