Índice

Lista de figuras

Lista de tabelas

Acrónimos

6LoWPAN	IPv6 Over Low power Wireless Personal Area Networks
ARP	Address Resolution Protocol
CIRD	Classless Inter-Domain Routing
DAD	Duplicated Address Detection
DAG	Directed Acyclic Graph
DAO	Destination Advertisement Object
DIO	DODAG Information Object
DIS	DODAG Information Object
DODAGs	Destination Oriented DAGs
EEROM	Electrically Erasable Programmable Read-Only Memory
EGP	Exterior Gateway Protocol
FFD	Full Function Devices
IANA	Internet Assigned Numbers Authority
ID	Identification
IEEE	Institute of Electrical and Electronics Engineers
IGMP	Internet Group Management Protocol
IGP	Interior Gateway Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6

LR-WPAN	Low Rate – Wireless Personal Area Network
MAC	Media Access Control
MCU	Multipoint Control Unit
MPDU	MAC Protocol Data Unit
MPDU	MAC Servisse Data Unit
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NUD	Neighbor Unreachability Detection
OSPF	Open Shortest Path First
PHR	Physical Header
PHY	Physical Layer
PPDU	Physical Protocol Data Unit
QoS	Quality of Service
RAM	Random-Access Memory
RFC	Request For Comments
RFD	RFD Reduced Function devices
RIP	Routing Information Protocol
RIPng	Routing Information Protocol Next Genaration
ROLL	Routing Over Low Power and Lossy Networks
RPL	Routing Protocol for Low Power and Lossy Networks
RTOS	Real Time Operating System

SHR	Synchronization Header
TCP	Transmission Control Protocol
TTL	Time To Live
UDP	User Datagram Protocol
ND	Neighbor Discovery

1. Introdução

"O descontentamento é o primeiro passo na evolução de um homem ou de uma nação."

Óscar Wilde

1.1 Motivação

Com a possibilidade dos sensores estarem conectados à Internet, todos os pedidos/acessos à informação podem ser realizados por qualquer pessoa no mundo. Esta possibilidade de acessos à informação a qualquer hora, é bastante importante numa variedade de aplicações, como por exemplo, o controlo da luminosidade numa exploração de cogumelos ou mesmo monitorização da temperatura de uma floresta para prevenção de fogos.

Foi esta perspectiva que levou muitos a pensar que a inserção da pilha TCP/IP seria uma boa ideia nas redes de sensores sem fio (RSSF) e, juntamente com o IPv6, tornar-se-ia numa rede global com imensas possibilidades. Contudo a pilha TCP/IP não foi concebida para estes tipos de dispositivos onde a principal preocupação é o cuidado na utilização dos escassos recursos disponíveis. A pilha TCP/IP contém protocolos que elevam o consumo e o tempo de processamento. Além disto, os pacotes são demasiado complexos e de tamanho excessivo comparativamente à memória típica de um dispositivo desta rede. Foi por esta razão que o IETF (*Internet Engineering Task Force*) definiu o grupo 6LoWPAN (IPv6 *Over Low Power Wireless Personal Area Networks*), que definiu uma camada de adaptação entre a camada de rede e a camada MAC, permitindo a ligação entre a rede IPv6 e as RSSF, usando endereços IPv6.

Ao contrário das redes sem fio, os protocolos de encaminhamento das redes com fio (por exemplo RIP "*Routing IP Protocol*" e OSPF "*Open Shortest Path First*"), consomem muita largura de banda devido à constante troca de mensagens, levando a uma pior performance. Outro problema destes protocolos é a problemática dos "terminais escondidos" nas redes sem fio. Para endereçar estes problemas, o grupo de trabalho ROLL (*Routing Over Low Power and Lossy Networks*) do IETF desenvolveu um protocolo de encaminhamento IPv6 designado por RPL (*Routing Protocol for Low power and Lossy network*). Este é constituído por diversas especificações da camada de acesso à rede, com a utilização da norma IEEE 802.15.4 (Institute of Electrical and Electronics Engineers) juntamente com o 6LoWPAN.

No desenvolvimento de aplicações para as RSSF, apercebemo-nos da impossibilidade de reutilização de diversos algoritmos num vasto número de micro controladores existentes (por exemplo MSP e AVR). Em virtude disso, os sistemas operativos para as RSSF, cada vez mais, tornam simples os acessos ao *hardware* bem

como a programação dos aplicativos, permitindo economizar o máximo de energia possível. Os sistemas operativos como o Contiki tornam-se eficientes em termos de consumo de memória, processador e energia.

1.2 Objectivos

A importância deste projecto é fortemente contextualizada nas redes de sensores sem fio (RSSF) com ligação à Internet. O projecto tem como finalidade a implementação de uma rede de sensores sem fio experimental com a possibilidade de processar a luminosidade de um ambiente e reagir mediante comandos enviados a partir da Internet via IPv6. Sendo possível ser incorporada a medição de outras grandezas, tais como: temperatura e humidade.

Para a comunicação dentro da RSSF foi usada a norma IEEE 802.15.4 e o protocolo de encaminhamento RPL. De forma a interligar-se à rede IPv6, será utilizada a camada de adaptação 6LoWPAN. Não se pretende com este projecto, desenvolver de raiz um protocolo de comunicação ou de encaminhamento, mas sim a utilização dos já existentes, 6LoWPAN e RPL.

Este projecto foi realizado em quatro fases:

- Estudo dos protocolos de rede e sistemas operativos aplicados a redes de sensores;
- Realização de testes nas várias soluções existentes no sistema operativo Contiki, permitindo a escolha da melhor solução para o *hardware* existente;
- Implementação de uma solução capaz de monitorizar um ambiente e enviar dados (estado do led, bem como o ligar/desligar, valor da luminosidade) para uma aplicação de modo a que o utilizador os possa consultar;
- Teste da solução e conclusões.

1.3 Organização da dissertação

Esta dissertação encontra-se dividida em 9 capítulos. Além deste capítulo, os restantes capítulos são descritos a seguir:

- **Rede de sensores sem fio (RSSF)** – Introduz definições e conceitos necessários neste tipo de rede;
- **IPv6** – Descreve o protocolo de comunicação IPv6;
- **6LoWPAN** – Descreve o protocolo de comunicação que implementa o IPv6 em redes de baixa potência e baixas taxas de transmissão;
- **RPL** – Apresenta o protocolo de encaminhamento utilizado em redes de baixa potência e de baixas taxas de transmissão;
- **Sistema operativo Contiki –** Introduz definições e conceitos dos sistemas operativos usados em redes de baixa potência e baixas taxas de transmissão, contendo também uma descrição do sistema operativo Contiki;
- **Implementação e testes –** Apresenta o material, configurações e testes necessários para a construção do projecto final;
- **Projecto Final –** Apresenta o projecto final de forma global.
- **Conclusão e trabalho futuro –** Apresenta as conclusões de trabalho futuro do projecto.

2. Redes de Sensores Sem Fio

"Num mundo em constante transformação nenhum conhecimento é definitivamente adquirido. Por isso, qualquer que seja a idade, o meio social, a profissão, cada um deve esforçar-se por aumentar ou renovar o seu saber."

Bernard Roux

2.1 Redes de sensores sem fio

Uma rede de sensores sem fio ou proveniente do inglês *"Wireless Sensor Network*" é uma rede de dispositivos autónomos distribuídos espacialmente que utilizam sensores para monitorar as condições físicas ou ambientais, tais como: temperatura, pressão, movimento e som. Os dados são encaminhados via rádio pelos dispositivos autónomos que têm recursos limitados.

Estes dispositivos são conhecidos como nós sensores ("*sensor nodes*") e estão ligados sem fio (*wireless)* a um *gateway* central (coordenador) que contém as condições necessárias para analisar, processar e apresentar ou enviar os dados para outros computadores/dispositivos.

Uma das principais vantagens das RSSF é a mobilidade inerente a estes sistemas, que permite a sua instalação, em lugares remotos e de difícil acesso. Este tipo de sistemas apresenta ainda outras vantagens, tais como:

- Baixo preço e tamanho reduzido dos dispositivos;
- Flexibilidade e a inteligência distribuída por vários sensores;
- Disponibilidade.

Em contrapartida, uma das principais desvantagens é a reduzida autonomia devido ao uso de baterias. Actualmente, deparamo-nos com um mundo de novas tecnologias, onde a nano tecnologia impera na necessidade presente. Apesar dos componentes serem cada vez mais de tamanho muito reduzido a sua fonte de energia encontrar-se limitada ao tamanho e capacidade. Verificamos que, para a bateria de um dispositivo ter durabilidade energética, tem de ter obrigatoriamente um determinado tamanho para poder armazenar a energia necessária em virtude da resolução do problema. Visto não haver possibilidade de grandes quantidades de armazenamento energético, a solução para este problema passa por reduzir o consumo dos recursos/processamento.

As RSSF têm uma vasta aplicabilidade. Graças à diversidade de sensores, estes podem ser usados numa grande variedade de aplicações tais como:

- **Industriais** – Monitorização da radiação, temperatura, humidade, composição do gás, monitoramento de pressão, ou qualquer variável inerente ao lugar de produção ou processamento de difícil acesso;

- **Militares** – Monitorização e acompanhamento da posição do inimigo para medidas de segurança e controle;

- **Ambientais** – Mudanças no ambiente, monitorização de um prédio, floresta, mar, entre outros;

- **Engenharias** – Monitorização do stress e tensão de um material;

- **Medicinas** – Monitorização do batimento cardíaco do paciente, pressão arterial, entre outros;

- **Tráfegos rodoviários** – Monitorização e verificação do melhor caminho e disponibilidade do mesmo;

- **Tráfegos Aéreos** – Monitorização do espaço aéreo de forma a verificar os voos de partida, chegada entre outras aplicações.

2.2 Requisitos

Muitas das aplicações vistas anteriormente, partilham características em comum em protocolos e programas, mesmo sendo usadas em diversos ambientes operacionais. É importante definir metas para cada uma das RSSF, uma generalização dos requisitos para cada nó de sensor não é prático, nem a reutilização do código. Cada nó de sensor deve ser implementado com base nas suas características, limitações e ambiente de destino para que no final se obtenha uma RSSF, optimizada, rápida, funcional e adaptada às necessidades [1]. Para isso é importante ter em conta algumas preocupações básicas na projecção de uma RSSF, tais como:

- **Auto configuração** - Os nós sensores podem ser colocados manualmente num local sendo os dados transmitidos através de um caminho pré-determinado ou, em locais/objectos em movimento (ex.: animais, veículos), onde existe a necessidade de se autoconfigurarem. O objectivo é adaptação a qualquer tipo de ambiente em movimento.

Isto significa que os nós sensores a qualquer momento estão dispostos aleatoriamente e nos casos onde o movimento é elevado, têm como consequência o grande consumo de energia, devido ao processamento constante para determinar o melhor caminho até ao *gateway*. Por este motivo, a eficiência energética depende da posição do *gateway*;

- **Escala** - Quando se compara outras redes sem fio com a RSSF, esta exige mais nós sensores para cobrir uma mesma área. Devido ao seu alcance reduzido, em grandes áreas ocorrem problemas associados com a cobertura do sinal, enquanto que em pequenas áreas é importante limitar o número de nós, uma vez que pode ocorrer replicação de informação o que pode levar ao comprometimento de entrega dos dados;

- **Mobilidade** - Nas RSSF a mobilidade tem um grande impacto sobre o esquema da topologia de rede. Os nós sensores podem ser estacionários, mas, para tirar o máximo partido do potencial das RSSF, os nós sensores devem ser móveis. Ao analisar a mobilidade existem algumas preocupações a ter em conta. Após a colocação inicial de um nó de sensor, este pode mudar a sua localização, falhar ou acabar a bateria do mesmo, provocando uma lacuna na topologia da rede, neste caso eles devem ser auto configuráveis;

- **Programação** - Ao lidar com tão pouca memória, existe uma necessidade de programar cada dispositivo para uma finalidade específica. Não há espaço para grande codificação/programação ou processamento. O código deve ser directo e limpo de modo a que não sacrifique o desempenho e eficiência do sistema;

- **Energia** - Os dispositivos de uma RSSF possuem dispositivos de alimentação limitada, que pode causar transtornos durante o serviço. As baterias são geralmente de pequeno porte e facilmente esgotam a energia. Características como a recepção, processamento, transmissão de dados e até mesmo em estado inactivo (*idle*), consomem energia sendo importante controlar o sistema tendo em conta as características acima mencionadas.

2.3 Nós sensores

Os nós sensores ou do inglês "*sensor nodes*" são normalmente dispositivos pequenos e autónomos de baixa potência, compostos tipicamente por memória, processador, sensores, módulo rádio e um dispositivo energético para alimentar todo o sistema como mostrado na Figura 1 e apresentados de seguida:

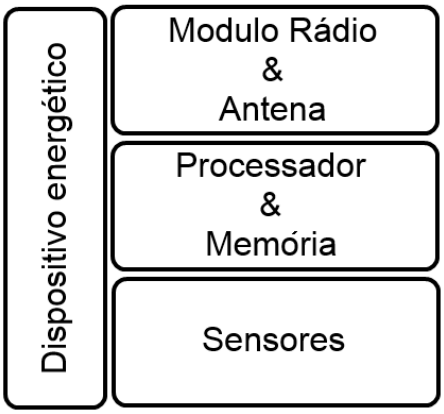


Figura 1 - Típico esquema de um nó de sensor [1]

- **Memória** – A memória pode estar incluída no chip de memória *flash* ou RAM do microcontrolador. Esta armazena as instruções executadas pelo processador, dados captados pelos próprios sensores ou de outros dispositivos;

- **Processador** – O processador juntamente com a memória formam a unidade de processamento *"microcontroler unit*" (MCU). O processador tipicamente com baixo poder de computação pode usar três modos diferentes de processamento:

a) ***Idle*** – Este modo é usado para ficar à espera de dados de outros sensores;

b) ***Sleep*** – É o mais utilizado de forma a ajudar na poupança de energia;

c) ***Active*** – Sempre que é necessário processar dados, sejam eles captados pelos sensores, enviados para outros nós sensores ou recebidos de outros nós sensores.

- **Sensores** – O sensor é um material ou um dispositivo passivo que altera as suas propriedades de acordo com as suas características. Dependendo da aplicabilidade, um dispositivo poderá conter diversos sensores. Um único dispositivo pode ser usado para captar, por exemplo, luminosidade, temperatura e humidade;

- **Modulo rádio** – É responsável pela comunicação sem fio "*wireless"* de curta distância, tipicamente até 100 metros e com baixa velocidade, entre 10 a 100 kbps. A comunicação pode ser feita por infravermelhos, laser, ou o mais usual, frequência de rádio com uma antena para um aumento na distância de comunicação;

- **Dispositivo energético** – Este dispositivo é usado para alimentar todo o sistema. No dispositivo, tudo consome energia: a captação de dados, a comunicação, o processamento, entre outras operações. No entanto é na comunicação que existe o maior consumo de energia. Os dispositivos energéticos mais comuns são as baterias recarregáveis, painéis solares ou condensadores, dependendo da aplicação. Devido à capacidade limitada de armazenamento de energia, muitos dos nós sensores são equipados com equipamentos de captação de energia, tais como a energia solar ou eólica por exemplo. Deste modo os nós sensores podem operar sem qualquer manutenção durante meses ou mesmo anos.

2.4 Rede de sensores

O termo rede de sensores proveniente do inglês "*Sensor Network*", refere-se aos inúmeros nós sensores com capacidade para captação e processamento de dados e o mais importante na rede de sensores, o envio dos dados via rede sem fios [1].

Na actualidade, as redes de sensores crescem a olhos vistos devido à facilidade de implementação dos nós sensores de baixo custo. Cada vez mais existem esquemas de ligação para criação de nós sensores de livre acesso, o que possibilita que qualquer pessoa em casa consiga desenvolver a sua própria rede de sensores.

Independentemente da funcionalidade e aplicabilidade dos nós sensores, os dados são encaminhados pela rede sem fios, de nó em nó até um destino, (sendo o mais comum, o coordenador), para que no final o utilizador consiga ter acesso à informação. A transmissão de dados é feita através de protocolos não proprietários ou proprietários que são normalmente desenvolvidos tendo em conta a norma standard IEEE 802.15.4, que especifica a camada física (PHY [*physical*]) e a camada de acesso ao meio (MAC [*Media Access Control]*) para redes pessoais sem fio e de baixo consumo (LR-WPAN "*Low Rate – Wireless Personal Area Network*") [1] [2].

Os protocolos de comunicação, como o IPv6 sobre redes pessoais de baixa potência e baixa taxa de transmissão (6LoWPAN), ZigBee, DASH7 ou WirelessHART, apresentados a seguir, conjugados com o padrão IEEE 802.15.4, completam o conjunto de protocolos destinados para as RSSF, sendo uma enorme ajuda na criação de uma grande rede escalável.

- **6LoWPAN** – É um protocolo "*open source*" definido pelo IETF, que define mecanismos de encapsulamento e compressão do cabeçalho que permitem que pacotes IPv6 sejam enviados e recebidos via redes IEEE802.15.4;

- **ZigBee** – É um protocolo proprietário desenhado pela ZigBee Alliance, um grupo de empresas que mantém e publica a norma ZigBee. Uma rede ZigBee fornece uma rede que tem pelo menos um nó de sensor central, que actua como coordenador da topologia de rede estrela ou árvore;

- **DASH7** – Mantido pela DASH7 Alliance, é um protocolo *open source* e alternativo ao ZigBee, sendo superior no alcance e de baixo consumo de energia.

- **WirelessHART** – Foi o primeiro protocolo de comunicação sem fio *standard* especialmente projectado para aplicações industriais como medições no controlo de aplicações [3];

- **EnOcean** – EnOcean é uma tecnologia proprietária de colheita de energia usada em edifícios e instalações industriais. Baseia-se na operação de colheita energéticamente eficiente a partir do ambiente de operação. A colheita da energia é obtida através da pressão de uma tecla e de todas as operações que geram energia suficiente para transmitir sinais sem fios.

2.5 Topologia

A topologia de uma rede sem fios, estrela, árvore ou malha, apresentada de seguida e representada na Figura 2, específica a forma como os nós estão organizados, por forma a fornecer intercomunicação entre os dispositivos, a forma como o coordenador actua na rede e a configuração das rotas usadas para a transmissão da informação.

- **Estrela** – A topologia Estrela, proveniente do inglês "*star"*, é uma topologia simples para cobrir uma área pequena e com uma rede de pequena escala, devido à capacidade de alcance dos dispositivos. Na topologia em estrela todas as informações de todos os dispositivos *end-devices* convergem para um coordenador único que está no centro da topologia. O coordenador controla toda a rede e é responsável por iniciar e manter todos os dispositivos da topologia estrela. Não sendo necessário grandes quantidades de memória e processamento;

- **Malha** - A topologia em malha, proveniente do inglês "*mesh"*, é usada em redes de grande escala. É uma rede de encaminhamento complexo e rotas flexíveis. Para este efeito, é necessário grandes quantidades de memória e processamento. Estes recursos são usados para calcular os percursos para a entrega de dados. No entanto, nesta abordagem, todos os dispositivos estão livres para se comunicarem, o coordenador não tem um papel central, mas é responsável por iniciar e manter todos os dispositivos da rede. Caso um dispositivo perca a comunicação, a rede automaticamente redirecciona os dados por outro caminho diferente, tornando-a mais confiável;

- **Árvore** – A topologia árvore, proveniente do inglês *"tree"*, é uma topologia utilizada em redes de médio porte. É uma simples rede de encaminhamento que não recorre a rotas alternativas. É semelhante à topologia de malha, mas esta topologia apresenta uma estrutura hierárquica em que o coordenador está na parte superior ou na parte inferior e a informação é encaminhada até ele. Isto pode causar problemas quando um dos dispositivos perde comunicação, levando a que os outros dispositivos percam também a ligação até ao coordenador.

O coordenador é único em todas as topologias apresentadas, este poderá ligar-se a outros coordenadores, formando uma grande e única rede.

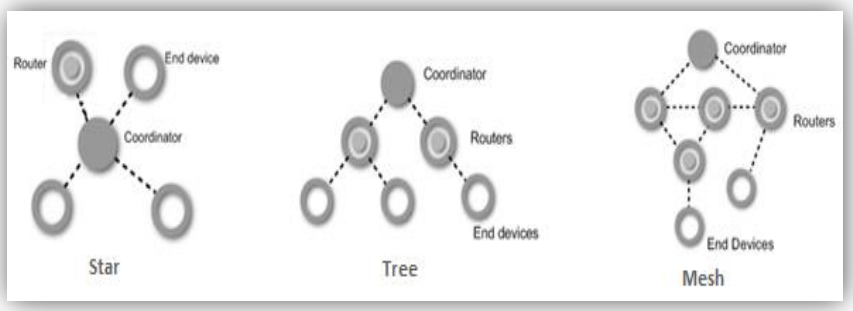


Figura 2 - Topologias típicas de rede [3]

2.6 IEEE 802.15.4

O IEEE 802.15.4 é uma norma padrão para redes pessoais sem fio de baixa velocidade de transferência e com perdas (*Low Power and Lossy Networks* [LLS]), mantido pelo IEEE *Task* 802.15 Grupo 4. Esta norma específica a camada física (PHY) e a camada MAC para dispositivos tipicamente pequenos, de baixo custo e baixa potência, que operam em velocidades de dados de 20-250 kbps. Neste tipo de rede, a LR-WPAN (*Low-Rate Wireless Personal Area Networks*) existem dois tipos de nós:

- **RFD** – *Reduced Function devices*, são dispositivos de funções reduzidas e capacidades limitadas, usados em aplicações muito simples, que não exigem a necessidade de grande processamento ou envio de grande quantidade de dados. Apenas funcionam como nós finais e só se podem comunicar com FFDs descritos abaixo;

- **FFD** - *Full Function Devices*, são dispositivos com maior capacidade do que os RFDs. Eles têm a capacidade de trabalhar como coordenador ou *router, gateway* para comunicação com outros FFD ou RFD.

IEEE 802.15.4 permite organizar a rede com múltiplas topologias. Seja qual for a topologia, deve conter pelo menos um FFD actuando como coordenador. Uma rede *peer-to-peer* define que cada dispositivo pode comunicar directamente com todos os outros dispositivos e permite criar redes mais complexas, como malha e árvore, representado na Figura 2. No entanto, esta não permite topologia em estrela [2].

Dentro das topologias encontram-se 3 tipos de equipamentos/nós sensores:

- **Coordenador** – O coordenador funciona como *gateway*, também conhecido como *sink*. É único na rede e do tipo FFD. Ele é responsável por coordenar a entrada e saída de dispositivos do tipo *router* e/ou *end-devices* entre outras características dependendo do tipo de rede. Como é um dispositivo que processa muitos dados e necessita estar constantemente ligado, é essencial que esteja ligado a um equipamento de energia confiável e o processador esteja sempre activo;

- **Routers** – O *router* é do tipo FFD e contém mecanismos para encaminhar os dados até ao coordenador. Para além das características anteriormente mencionadas, os *routers* podem ter as mesmas características que os *end-devices;*

- **End-devices** – Estes dispositivos podem ser do tipo FFD ou RFD dependendo da sua aplicabilidade. São os últimos da rede e comunicam através dos *routers* para enviar os dados para o coordenador. Dependendo da rede, podem ou não comunicar entre si. Numa rede ZigBee, estes dispositivos não podem comunicar entre si, já na rede 6LoWPAN os mesmos comunicam entre si. Estes dispositivos podem usar o processador nos três modos, dependendo da sua aplicabilidade.

	Coordenador	**Router**	***End-device***
***Full Function Devices* (FFD)**	Sim	Sim	Sim
***Reduced Function Devices* (RFD)**	Não	Não	Sim

Tabela 1 - Diferenças entre FFD e RFD

2.6.1 A *frame* do protocolo 802.15.4

A *frame* 802.15.4 é definida em quatro tipos diferentes:

a) ***Beacon frame*** – Para sincronização dos dispositivos e da rede pessoal (PAN);

b) ***Data frame*** – É usada para a transmissão de informação;

c) ***Acnowledgment frame*** – É opcional;

d) MAC command *frame* – Faz o controlo de transferências entre o MAC destino e origem.

Define ainda uma estrutura *superframe* delimitada por duas *frames beacon*. As *superframes* são enviadas pelo coordenador e são divididas, opcionalmente, em activo e inactivo, possibilitando que o coordenador entre em modo de *power-save*.

A *frame* 802.15.4, ilustrada na Figura 3, é formada da seguinte forma. Quando a camada superior passa o seu *payload* para a camada de ligação lógica, este especifica-o como MSDU (*MAC Service Data Unit*). Nesta camada é adicionado o cabeçalho MAC (MHR, *"MAC Header"*) no início (prefixo) e no final o MFR (*MAC Footer*) que contém o *Frame Check Sequence* (FCS).

Depois de formada a *frame* MAC esta é especificada como MPDU (*MAC Protocol Data Unit*) e é enviada para a camada física como PSDU (*Phisical Service Data Unit*). Na camada física, ao PSDU é adicionado no início o SHR (*Synchronization header*) e o PHR (*Physical Header*). O resultado final é um PPDU (*Physical Protocol Data Unit*) pronto para ser transmitido pela rede 802.15.4, sendo por definição, o pacote PPDU que contém um tamanho fixo de 127 bytes, onde 25 bytes são condicionalmente utilizados, restando apenas 102 bytes para o MSDU.

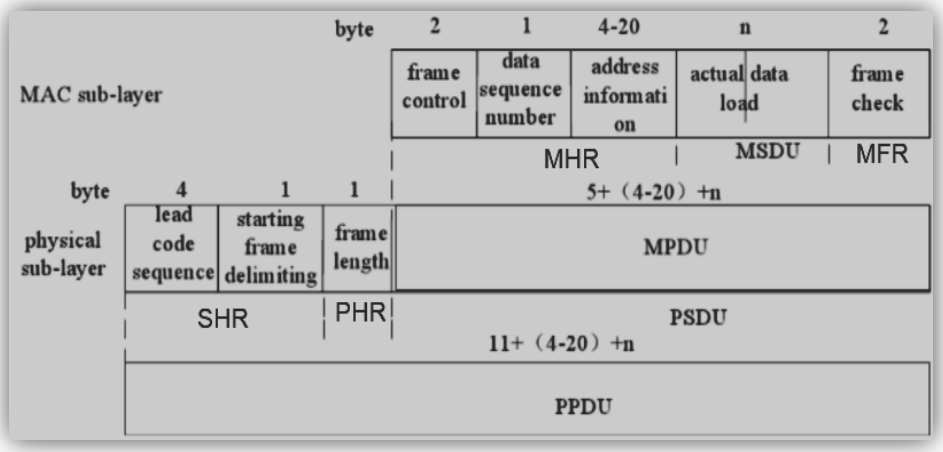


Figura 3 - Formato de uma *frame* 802.15.4 [4]

2.7 Vantagens e desvantagens das RSSF

Hoje é possível comprar os nós sensores a baixo custo ou dispositivos necessários para a construção de um nó de sensor de fabrico artesanal a muito baixo custo, de forma a criar uma RSSF.

As RSSF são frequentemente implementadas em áreas sem qualquer tipo de vigilância/segurança, estão susceptíveis a roubos, danificação e deste modo vulneráveis a uma variedade de potenciais ataques. Além disso, os sensores deste tipo de rede, sofrem várias limitações em particular, em termos de capacidade de processamento, memória e capacidade da bateria. Com estas limitações, é impossível ter a mesma implementação em redes sem fio tradicionais. Para evitar estes problemas, é necessário implementar estratégias de consumo e de poupança energética, sendo que isso significa reduzir o consumo dos recursos, mas de forma a garantir a confidencialidade e fiabilidade necessárias para a comunicação.

2.8 Sumário

Uma típica rede de sensores sem fio (RSSF) é formada por um conjunto de dispositivos conhecidos como nós sensores, que possuem capacidade de processamento, armazenamento e memória limitada. A comunicação entre os nós sensores é estabelecida pelas interfaces de rede sem fio, que opera em redes de baixa potência e baixas taxas de transmissão (IEEE 802.15.4). As RSSF podem ser usadas numa uma vasta série de aplicações, tais como: ambiental, militar e industrial, entre outras, sendo que em todas elas a principal preocupação é o consumo de energia. Devido à facilidade de implementação e proliferação, bem como ao custo reduzido dos dispositivos deste tipo de redes, têm vindo a existir uma maior preocupação na construção de um protocolo de comunicação padronizado (o 6LoWPAN – *IPv6 Over Low Power Personal Area Networks*), face aos protocolos já existentes, tais como o ZigBee, DASH7, WirelessHart. Todas as aplicações das RSSF têm certos objectivos que podem ser analisados na Figura 4.

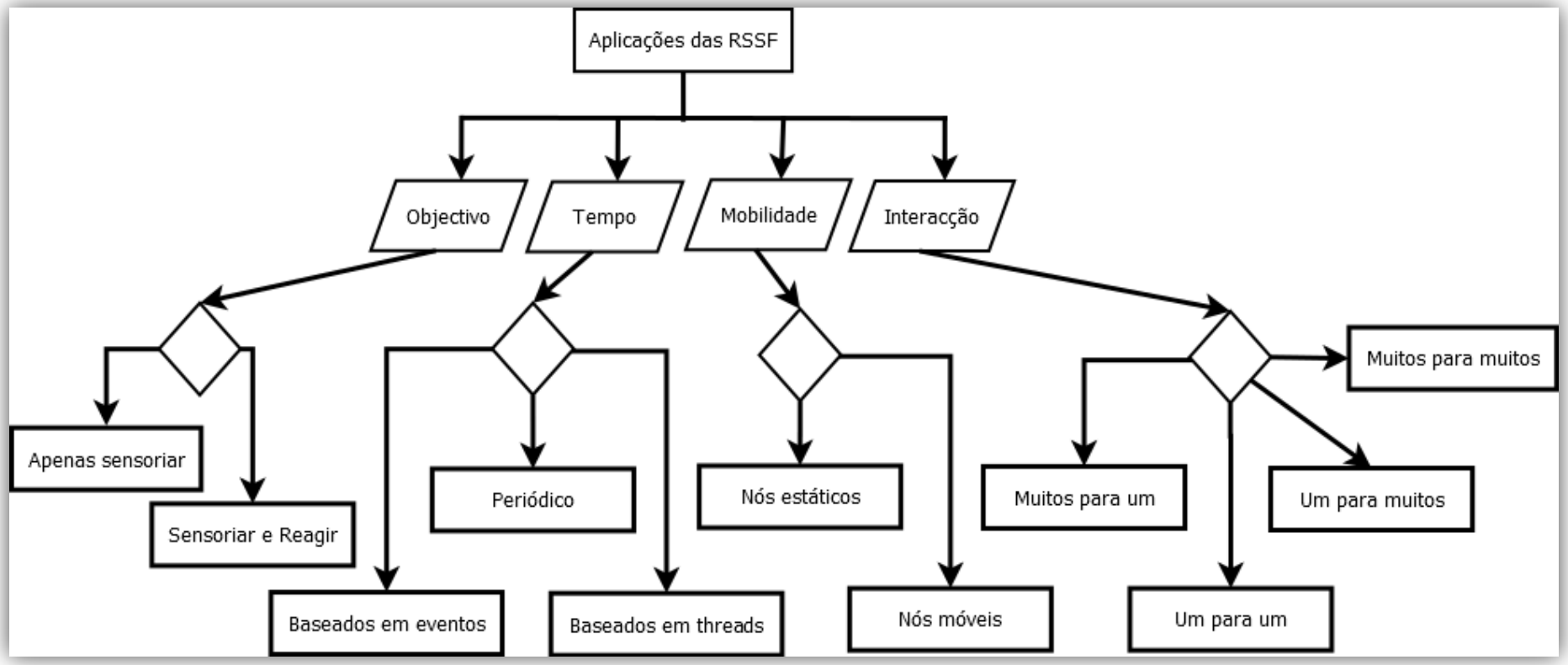


Figura 4 – Esquema de programação das RSSF

3. IPv6

"A tecnologia mais necessária para evolução do nosso mundo, se chama: simplicidade."

Di Medeiros

3.1 IPv6

Desenvolvido pelo IETF, o IPv6 é uma nova versão do IPv4 (*Internet Protocol*). O IPv4 é um protocolo de rede adoptado para comunicações na Internet. A estrutura do cabeçalho do IPv4 utiliza endereços apenas de 32 bits o que limita a atribuição de endereços livres tendo surgido a necessidade de criação de um novo protocolo com um maior espaço de endereçamento (IPv6), as razões são apresentadas de seguida:

- Cada vez mais existem novos utilizadores e dispositivos na Internet, no entanto 32 bits apenas fornecem uma quantidade inferior de 4,3 biliões de endereços, isto significa que não existe um endereço IP para cada pessoa ou dispositivo que usa a Internet;
- Grandes tabelas de encaminhamento devido ao crescimento de utilizadores e dispositivos;
- Nem mesmo técnicas como NAT (*Network Address Translation*) ajudaram, apenas adiaram o inevitável, o esgotamento de endereços IPv4.

O potencial problema com endereços IP tornou-se realidade em Fevereiro de 2011, quando a IANA (*Internet Assigned Numbers Authority*) abordou os últimos endereços e declarou o esgotamento do IPv4, como mostra a Figura 5 [5] [6].

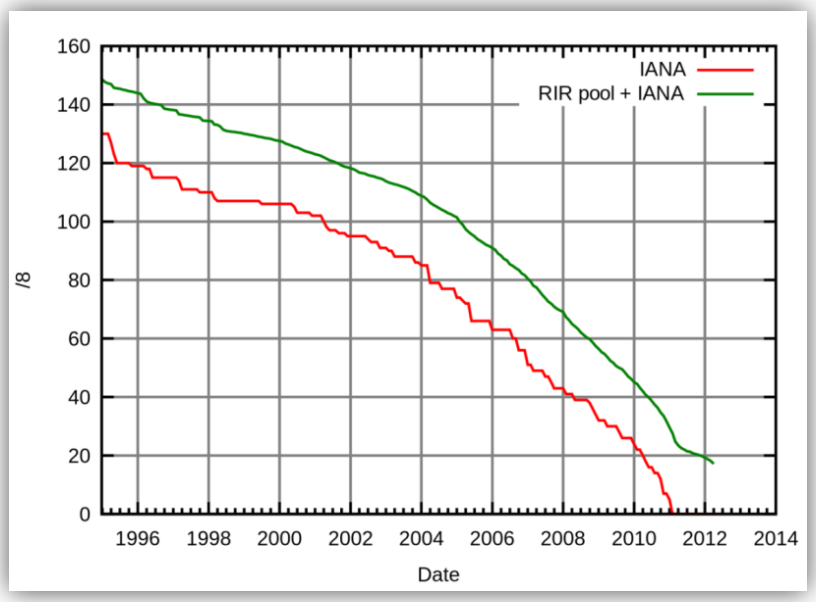


Figura 5 - Esgotamento de endereços IPv4 desde 1995 [5]

Na década de 90, antecipando este problema, o IETF iniciou o desenvolvimento do IPv6. Com endereços de 128 bits, o IPv6 fornece endereços IP quase ilimitados, para além de outras vantagens, como a configuração automática (*stateless*), descrita na secção 3.5, e melhor QoS (*Quality of Service*). Apesar de tudo, o IPv6 não substituirá o IPv4, pelo menos a curto e médio prazo, uma vez que a maioria das redes encontram-se implementadas em IPv4 e incapazes de comunicar directamente com *hosts*/dispositivos IPv6, resultando numa coexistência mútua por vários anos.

	***Internet Protocol Version* 4 (IPv4)**	***Internet Protocol Version* 6 (IPv6)**
Tamanho do endereço	32 bits	128 bits
Formato do endereço	173.194.34.248	2A00:1450:4007:802:0000:0000:1017
Notação do endereço	Números decimais separados por pontos "."	Números hexadecimais separados por dois pontos ":"
Número de endereços	2^2	2^{128}
Notação do prefixo	192.168.1.1/24	aaaa::f:f:1/64
Tabelas de encaminhamento	Grandes	Pequenas
Cabeçalho	Complexo	Simples
Duração do IP	Se o endereço IP é manual, este é permanente e se for automático depende do DHCP.	Normalmente os IP são duráveis (estáticos) para um e transientes (dinâmicos) para outros.
***Classless Inter-Domain Routing* (CIDR)**	Variável	Fixos, 64 bits para identificação da rede e 64 para identificação do host.
Endereços de broadcast	Existe	Não existe.

Tabela 2 – Diferenças IPv4 do IPv6

No IPv6, o espaço de endereçamento de 2^{128} é suficientemente grande para não ser necessário o uso de NAT. Para resolver o esgotamento dos endereços IP, a cada interface de rede pode ser atribuído um endereço IPv6 global único, possibilitando que um equipamento possa comunicar via Internet. Algumas diferenças entre as duas versões são apresentadas na Tabela 2. É importante notar que o IPv6 não é apenas um

IPv4 com um maior espaço de endereçamento. Existem outras diferenças significativas, principalmente no que se refere à configuração de IP´s e redes, QoS e segurança.

3.2 Endereçamento

A arquitectura de endereçamento é definida no RFC4291. A principal alteração é o comprimento do endereço. Para ter acesso a todos os nós, o IPv6 usa identificação única de cada nó. A cada interface de rede é atribuído um endereço de 128 bits que representa a identificação exclusiva da rede e da interface, o qual não permite que duas interfaces tenham o mesmo endereço.

3.2.1 Notação

Existem três formas convencionais para representar endereços IPv6:

a) Ao contrário dos endereços IPv4 onde os 32 bits são divididos em grupos de 8 bits cada um, separados por "." ou o mais usual, escritos com dígitos decimais, como pode ser visto na Figura 6. No endereço IPv6 representado na Figura 7, os 128 bits são divididos em 8 grupos de 16 bits e separados por ":", representados por números hexadecimais (dígitos 0 a F). O IETF decidiu-se por esta convenção porque é mais fácil converter de hexadecimal para binário do que decimal para binário, uma vez que cada dígito hexadecimal representa quatro dígitos binários [7];

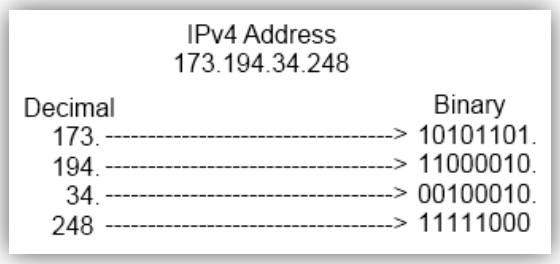


Figura 6 - Representação do endereço IPv4

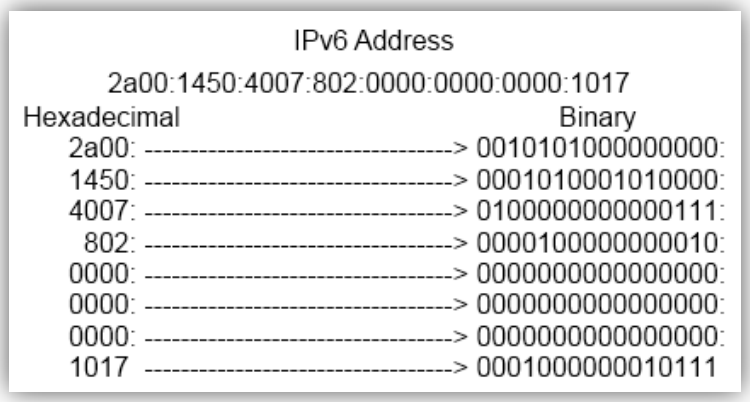


Figura 7 - Representação do endereço IPv6

b) Ao invés de escrever um endereço IPv6 com longas sequências de grupos de 16 bits de zeros a escrita pode ser simplificada usando apenas uma vez dois pontos duplos "::". Além disso, para simplificar as letras podem ser escritas em maiúsculas e minúsculas. Em acréscimo, qualquer grupo de quatro dígitos hexadecimais de zeros dentro de um endereço IPv6 pode ser reduzido para um único zero ou totalmente omitido. Portanto, os seguintes endereços IPv6 representados na Figura 8 são semelhantes e igualmente válidos;

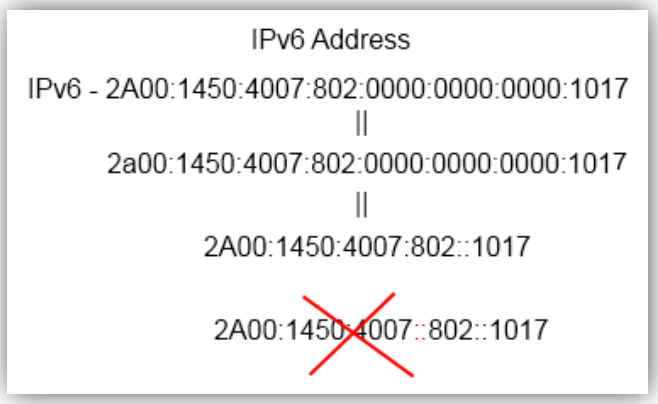


Figura 8 - Representação de diferentes formas de endereços IPv6

c) Outro modo de representar, tal como demonstra a Figura 9, é incorporar um endereço IPv4 num endereço IPv6, usando os últimos 32 bits. Este tipo de endereço é usado para representar os endereços IPv4 de uma interface usando endereços IPv6 [8];

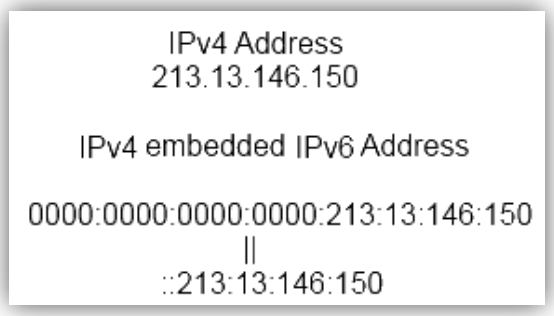


Figura 9 - Endereço IPv4 dentro do endereço IPv6 [9]

Tal como o IPv4, o IPv6 usa *subnetting* para subdividir uma única classe de rede em várias redes. O IPv6 usa um número, como "prefixo" para identificar o número de bits significativos de uma rede. Nesta representação, como mostra a Figura 10, permite a redução do tamanho da tabela de encaminhamento e aceleração do encaminhamento dos pacotes, ajudando também a uma melhor representação da topologia de rede, do fornecedor de serviços de Internet (ISP), a posição geográfica e o mais importante, das sub-redes;

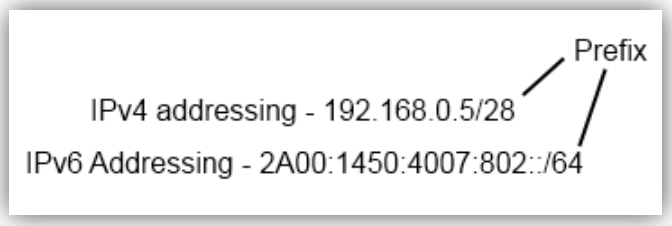


Figura 10 - Representação do prefixo IPv4 e IPv6 [9]

3.2.2 Tipos

Tal como os endereços IPv4, os 128 bits do endereço IPv6 são atribuídos a uma única interface, possibilitando uma comunicação por *unicast*. No entanto, o IPv6 vem com outras mudanças na comunicação. O IPv6 não suporta *broadcast* para envio de pacotes para toda a rede, a mesma funcionalidade é implementada no IPv6 por comunicação com *multicast*. Outra mudança importante é o aparecimento de endereços anycast. Estes tipos de endereços podem ser consultados na Tabela 3.

Tipo de endereço	Prefixo em binário	Notação IPv6
Não especificado	00...0 (128 bits)	::/128
Loopback	00...1 (128 bits)	::1/128
Multicast	11111111	FF00::/8
Link-local unicast	1111111010	FE80::/10
Site-local unicast	1111111011	FEC0::/10
Global unicast	Todos os outros endereços	

Tabela 3 - Tipos de endereços IPv6

a) *Unicast* - No IPv6 existem endereços privados e públicos. Um endereço *unicast* identifica uma interface única dentro do âmbito do tipo de endereço. O que significa que, se um pacote é encaminhado para um endereço *unicast* é entregue apenas na interface identificada por esse endereço, como demonstrado na Figura 11;

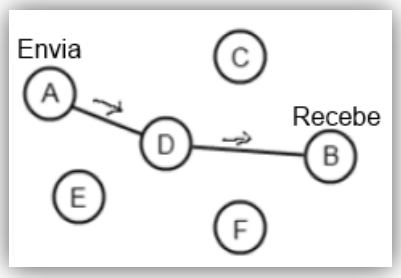


Figura 11 - Comunicação Unicast

Existem cinco tipos de endereços, são eles:

b) *Global Unicast* – Os endereços Global *Unicast* são o equivalente aos endereços públicos IPv4, sendo endereços globais. A estrutura do endereço, representada na Figura 12, contém um prefixo de rede estruturado (n+m=64 bits), que permite a agregação de várias redes, pequenas tabelas de encaminhamento e um identificador da interface. O campo global routing prefix de n bits (48 bits = 3 bits + 45 bits) contém a identificação do tipo de endereço que neste caso é sempre 001 (3 bits), juntamente com a identificação do ISP (45 bits). Já o campo subnet ID de m bits (16 bits) identifica a rede subnet e por último a interface ID de 64 bits representa a identificação do host;

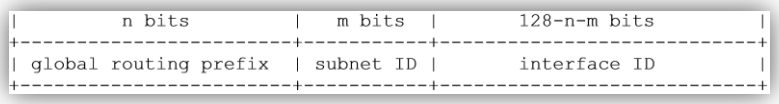


Figura 12 - Estrutura do endereço Global Unicast [8]

c) *Link-local* – Este tipo de endereço privado é automaticamente configurado pelo próprio host, este é usado para a comunicação entre vizinhos dentro da mesma ligação (Link) e para o processo de descoberta de vizinhos (*Neighbors Discovery*). Os endereços Link-local nunca são encaminhados por routers, já que pertencem apenas dentro da rede interna. Começam sempre por FE80, sendo o seu prefixo FE80::/64 como podemos analisar na Figura 13.

Figura 13 - Estrutura do endereço Link-local [8]

d) Site-local – Este tipo de endereço local foi originalmente projectado para serem usados para dentro de uma rede local, sem a necessidade de um prefixo global. No entanto, este tipo de endereço encontra-se descontinuado [10], mas apesar disso continua-se a falar ainda muito e por essa razão o mesmo é explicado no parágrafo seguinte e visualizado na Figura 14.

Os endereços Site-local podem ser comparados como os endereços privados do IPv4, ex: 10.0.0.0/8. São usados em redes (empresariais) onde não existe a necessidade de um endereço global, por exemplo: redes privadas que não têm acesso directo. Tal como no Link-local, os pacotes com endereços de destino Site-local não são encaminhados pelos routers para fora da rede privada. Nos endereços Site-local os 10 primeiros bits são fixos, os restantes identificam a sub-rede e a interface do host, FEC0::/10.

Figura 14 - Estrutura do endereço Site-local [8]

e) ***Special*** – Existem dois tipos de endereços especiais. São eles:

- ***Unspecified*** – Este endereço nunca deve ser utilizado em nenhum nó ou encaminhado. Tal como o nome indica, endereço não especificado ou ausência de endereço, sendo representado por 0:0:0:0:0:0:0:0 ou :: [8];

- ***Loopback*** – Tal como no IPv4, o endereço *Looback* (IPv4 -127.0.0.1 ; IPv6 - 0:0:0:0:0:0:0:1 ou ::1) é usado pelo nó para enviar pacotes IPv6 para ele próprio e tal como os endereços *Unspecified*, os pacotes com o endereço *Loopback* nunca deverão ser enviados para a rede ou encaminhados por um router [8].

f) Compatible – Este tipo de endereço já foi descontinuado [8]*;*

g) ***Multicast*** **-** O IPv6 utiliza *multicast* como *broadcast* uma vez que o modo de funcionamento é semelhante. Os endereços *multicast* são usados para identificar um grupo de interfaces, e cada interface pode pertencer, a qualquer momento, a um ou mais grupos. Um pacote enviado para um endereço *multicast* é entregue a todas as interfaces pertencentes ao grupo *multicast*, como se pode observar na Figura 15;

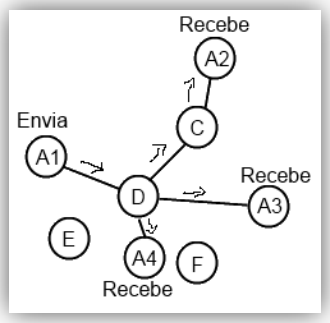


Figura 15 - Comunicação Multicast

No IPv4 o scope é definido pelo campo TTL e tem 8 bits, já no IPv6 o campo identificado por scope tem 4 bits. Na Tabela 4 o campo de scope indica a rede IPv6 onde os pacotes se destinam no tráfego *multicast*. Por exemplo, se um router recebe um pacote com o campo scope superior a 14 "E" em hexadecimal inclusive, o router não irá encaminhar o pacote.

O último campo de 112 bits no endereço *multicast* é o ID do grupo, utilizado para identificar o grupo *multicast*. No entanto, de FF00:: para FF0F:: os endereços *multicast* não podem ser atribuídos a nenhum grupo *multicast,* porque eles são reservados [11]. A estrutura do endereço pode ser visualizada na Figura 16.

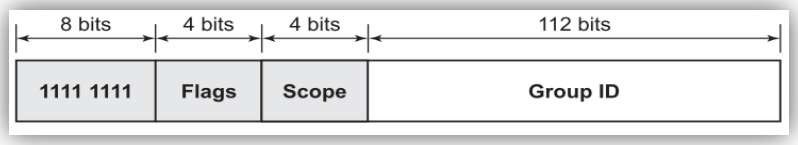


Figura 16 - Estrutura de um endereço IPv6 multicast [12]

Valor do *Scope* (Hex)	Valor do *Scope* (Bin)	*Scope*
0	0000	Reservado
1	0001	Interface local
2	0010	Link local
3	0011	Reservado
4	0100	Admin-local
5	0101	Site-local
8	1000	Organization-local
E	1110	Global
F	1111	Reservado
6,7,9,A, B,C,D	0110,0111,1001,1010, 1011,1100,1101	Não atribuido

Tabela 4 - Valores do campo scope para o endereço multicast [8]

h) *Anycast* – É um novo tipo de endereço e é utilizado para identificar um grupo de interfaces. No entanto, os pacotes endereçados com endereço *anycast* são transmitidos apenas às interfaces do grupo que está mais próximo do remetente original, ou seja utilizando o exemplo da Figura 17, apesar de A2 e A3 pertencerem ao grupo, o *host* A1 verifica que na tabela de encaminhamento o A4 se encontra mais perto e comunica apenas com este. Para alcançar esta comunicação a infra-estrutura de encaminhamento deve estar ciente das interfaces que têm anycast atribuído e a sua distância em termos de métricas de encaminhamento, desta forma, reduzindo a quantidade de tráfego de dados e encaminhamento na rede.

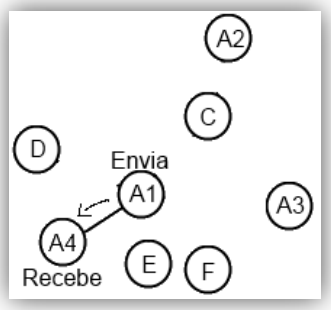


Figura 17 - Representação de uma comunicação Anycast

3.3 Cabeçalho

Um pacote contém um cabeçalho que tem a informação necessária para movê-lo através da Internet, sendo seguido pelos dados. O cabeçalho IPv6 baseia-se no antigo cabeçalho IPv4, no entanto, apenas os campos necessários foram mantidos e foram expandidos para acomodar os endereços que têm maiores dimensões. Como apresentado na Figura 18, os campos obsoletos ou desnecessários foram completamente removidos.

O cabeçalho IPv6 tem apenas 40 bytes, em que 32 são usados para endereços IPv6 e os restantes 8 bytes são usados por 6 campos. Para fazer o cabeçalho IPv6 mais eficiente e flexível para o futuro, os campos menos comuns tornaram-se extensões do cabeçalho, e são colocados após o cabeçalho IPv6 básico. Desta forma, os pacotes passam rapidamente através dos routers e apenas aqueles com extensão, são profundamente analisados. Cada cabeçalho contém um campo, que descreve o cabeçalho seguinte, podendo ser encadeado, formando um pacote, como representa a Figura 19.

O cabeçalho possibilita ter seis extensões, cada uma com um novo cabeçalho para identificar o cabeçalho seguinte. São elas:

- *Hop-by-Hop option;*
- *Destination option;*
- *Routing;*
- *Fragment;*
- *Authentication;*
- *Encapsulation Security Payload;*

Cada extensão fornece suporte para a segurança, gestão da rede, encaminhamento e/ou fragmentação. Quando é usado mais do que uma extensão no mesmo pacote, deve seguir a ordem como representa na Tabela 5, podendo ser usada apenas uma vez. No entanto, a extensão *"Destination option*" pode ocorrer duas vezes, uma vez antes de um cabeçalho de encaminhamento e uma vez antes do cabeçalho da camada superior [12] [11].

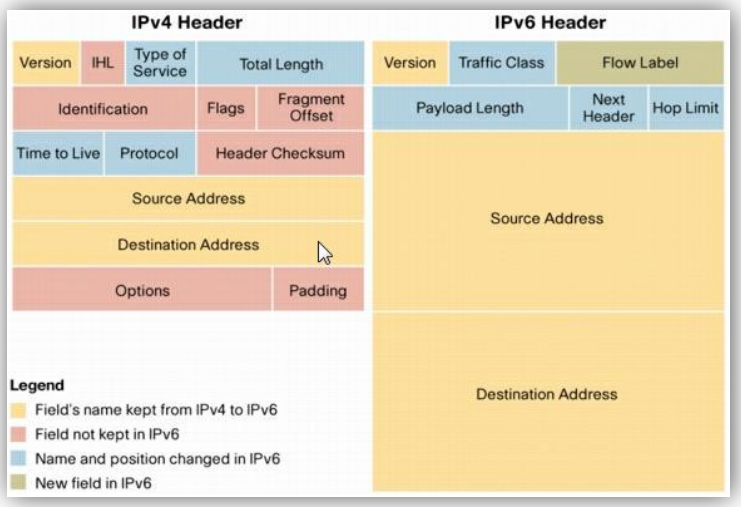


Figura 18 - Comparação do cabeçalho IPv4 e IPv6 [13]

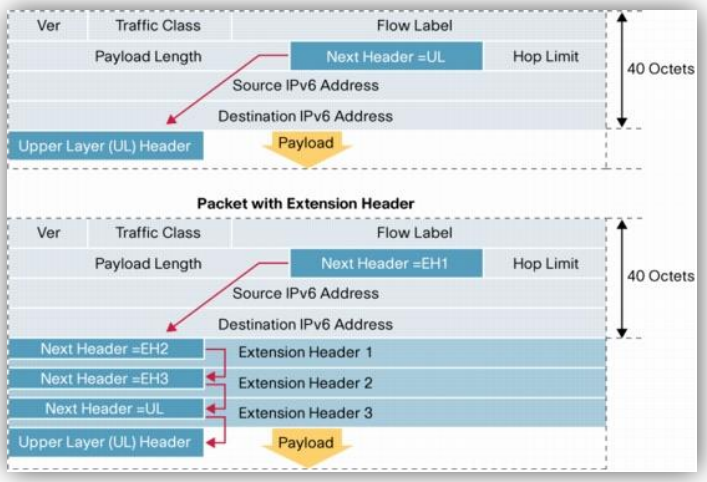


Figura 19 - Localização das extensões do cabeçalho IPv6 [13]

Order	Header Type	Next Header Code
1	Basic IPv6 Header	-
2	Hop-by-Hop Options	0
3	Destination Options (with Routing Options)	60
4	Routing Header	43
5	Fragment Header	44
6	Authentication Header	51
7	Encapsulation Security Payload Header	50
8	Destination Options	60
9	Mobility Header	135
	No next header	59
Upper Layer	TCP	6
Upper Layer	UDP	17
Upper Layer	ICMPv6	58

Tabela 5 - Tipos de extensões do cabeçalho IPv6 e a sua ordem recomendada [13]

3.4 *Internet Control Message Protocol* (ICMP)

O protocolo ICMP, definido no RFC4443, é uma ferramenta administrativa utilizada em diagnósticos e testes. Tal como no IPv4, o IPv6 baseou-se no ICMP, dando origem ao ICMP versão 6 (ICMPv6), com o cabeçalho representado na Figura 20, em todos os nós. Isto significa que se encontra dentro do IPv6 e não num protocolo à parte como no IPv4. O ICMPv6 contém as seguintes diferenças em relação ao ICMPv4:

- A transmissão de informação sobre os grupos *multicast*, é uma função similar ao IGMP *(Internet Group Management Protocol*) do IPv4;

- O *Address Resolution Protocol* (ARP), utilizado com o IPv4, passou a ser incorporado no ICMPv6;

- O IPv6 inclui um novo protocolo chamado *Neighbor Discovery, baseado no ICMP*, que será descrito no ponto 3.6.

As mensagens encontram-se divididas em três tipos:

- **Mensagens de erro** "*Error messages*", com o bit mais significativo do código definido a "0";

- **Mensagens informativas** "*Informative messages*", com o bit mais significativo a "1";

- **Próximo cabeçalho "***Next Header***"**, é resto das mensagens como *Echo Request, Echo Reply, Time Exceeded, Packet Too Big*, entre outras, foram mantidas. As mensagens ICMPv6 são encapsuladas dentro do pacote IPv6, podendo ser identificado na Figura 18 [14].

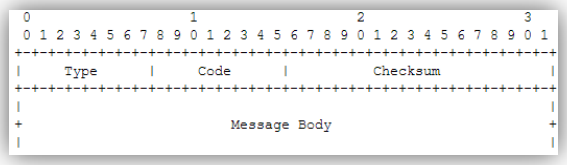


Figura 20 - Formato geral da mensagem ICMPv6 [14]

Existem dois campos muito importantes representados na Figura 20, que serão explicados a seguir:

- **Type** – Este campo de 8 bits identifica o tipo/propósito da mensagem, por exemplo se o campo estiver a 3 (00000011) significa que o pacote não pode ser entregue;

- **Code** – Com 8 bits, fornece mais informações sobre a mensagem, sendo esta informação dependente do tipo de mensagem [14] [15].

3.5 Configuração automática

Visto que os endereços IPv6 são mais difíceis de memorizar do que os endereços IPv4, o objectivo da configuração automática é eliminar o número de vezes que o utilizador tem que configurar manualmente o endereço IP. Em vez disso, assume que cada interface pode fornecer um endereço único.

Existem dois tipos de configuração automática:

- **Configuração *Stateless*** de endereços e outras configurações baseadas na recepção de mensagens "*Router Advertisement*" ou informações locais como o prefixo da rede e o endereço MAC para criar um endereço único da própria interface. Este tipo de configuração é melhor para pequenas redes, uma vez que é necessária uma configuração mínima em cada router e nenhuma configuração manual nos host [16];

- **Configuração *Statefull*** de endereços baseia-se na utilização de um protocolo de configuração de endereços, como o DHCPv6, para obter endereços entre outras configurações. Tal como no DHCPv4, um servidor mantém-se informado sobre os endereços que são atribuídos, o que abriga à utilização de uma base de dados para entrada e saída de endereços [16].

De qualquer forma, o melhor é usar ambos, usar a configuração automática *stateless* no *host* de modo a obter o seu próprio endereço e configuração automática *statefull* para obter outras informações, como por exemplo os endereços dos servidores de DNS, permitindo uma redução de envio e recepção de mensagens na rede e por consequência também no processamento.

3.6 *Neighbor discovery*

O serviço/protocolo *Neighbor Discovery* (ND) permite que cada nó IPv6, obtenha informações importantes como o endereço da interface que esteja no mesmo segmento local da rede. Esta opera em três princípios fundamentais [1]:

- Manutenção redundante das ligações para comunicação;
- Antecipação do movimento com base nas alterações da qualidade das ligações;
- Fazer de uso da informação recebida dos nós envolventes para uma melhor eficiência energética na descoberta de vizinhos.

O ND é um substituto do protocolo ARP do IPv4, inclui mensagens ICMP *Router Discovery e Redirect functions,* é incorporado no ICMPv6 e tem as seguintes funções [17]:

- Determinar prefixos da rede e outras configurações;
- Localizar endereços duplicados usando *Duplicated Address Detection* (DAD);
- Determinar, na camada 2 (ligação de dados) do TCP/IP, se o endereço de cada nó se encontrar na mesma rede;
- Encontrar routers nas suas imediações (vizinhos) que tenham possibilidade de encaminhar pacotes;

- Manter controlo dos vizinhos alcançáveis e inalcançáveis, usando *o Neighbor Unreachability Detection* (NUD);

- Detectar mudanças de qualquer endereço.

O modo de funcionamento do *Neighbor Discovery* está representado na Figura 21. Quando um dispositivo entra na rede, envia um pedido "Router Solicitation", de forma a dizer que está pronto para receber informações sobre as rotas, nós activos e também pontos de comunicação. Recebe "*Router Advertisement"* de dispositivos mais próximos e logo de seguida usa "*Neighbor Solicitation"* e "*Neighbor advertisement*" para se auto configurar, possibilitando a comunicação entre todos os dispositivos.

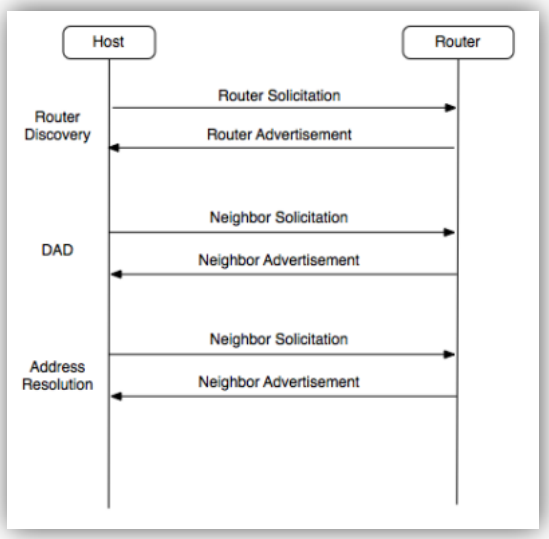


Figura 21 - Procedimento do Neighbor Discovery

3.7 Encaminhamento

Encaminhamento, ou em inglês *routing*, é um processo de encaminhamento de pacotes entre redes. Tal como o IPv4, o encaminhamento no IPv6 tem dois tipos de configuração:

- **Estático** – É o mais utilizado em redes pequenas, a configuração do endereço da rede de origem e do endereço de destino é manual;

- **Dinâmico** – É usado em redes médias e grandes, com a ajuda de protocolos de encaminhamento tais como RIPng [18] e OSPFv3 [19] para redes internas (IGP – *Interior gateway protocol*) e BGP [20] para as redes externas (EGP – *Exterior Gateway Protocol*).

3.8 Sumário

O protocolo IPv6, foi projectado para ser o sucessor do protocolo IPv4, que devido às suas limitações, como por exemplo, o tamanho do endereçamento (que levou à exaustão de endereços), limitação ao nível de segurança, mobilidade e encaminhamento. O IPv6 introduz 128 bits para o tamanho dos endereços, simplicidade no cabeçalho, uma camada de segurança, configuração automática da rede, permitindo que cada dispositivo de rede procure e reclame o seu próprio endereço de rede, entre outras características. Outra característica muito usual nas RSSF, por forma a não serem trocadas muitas informações, é o serviço ND. Este permite facilmente que cada nó crie o seu próprio endereço através do prefixo da rede, juntamente com o endereço MAC. O ND é usado apenas para troca de informações da rede como por exemplo, rotas e características da rede. Já a configuração automática *statefull* não se enquadra neste tipo de rede, devido à mobilidade dos nós e à grande carga de informação existente sempre que um nó entra na rede.

4. 6LoWPAN

"Ainda bem que chegamos a um paradoxo. Agora, há esperança de conseguirmos algum progresso."

Niels Bohr

4.1 6LoWPAN

O IPv6 suporta o endereçamento de muitos dispositivos devido à grande quantidade de endereços que possibilita, no entanto o seu tamanho de 128 bits é um grande problema nas redes que utilizam o protocolo IEEE 802.15.4. Devido aos problemas relativos aos nós dos sensores, anteriormente referidos, como por exemplo a baixa capacidade de memória, por forma a suportar o IPv6 em redes de baixa potência (*low power*), tem-se vindo a desenvolver o 6LoWPAN.

O acrónimo 6LoWPAN significa IPv6 *Over Low Power Wireless Personal Network*, e é o nome do grupo de desenvolvimento da IETF, que cria e mantêm as especificações que nos permitem usar IPv6 nas redes IEEE 802.15.4. Este grupo trabalha para definir os RFC *(Request For Comments*) do 6LoWPAN. Cada RFC, define *standards* de métodos, comportamentos, pesquisas ou inovações, capazes de definir a compressão, encapsulação e fragmentação do cabeçalho dos pacotes IPv6 em *frames* IEEE 802.15.4, permitindo que os mesmos sejam enviados e recebidos nestas redes. Os mais importantes são:

- **RFC4919** – Define uma visão geral, suposições, declarações do problema, e metas do 6LoWPAN;

- **RFC4944 ou draft-ietf-6lowpan-format** – Define como é feita a transmissão dos pacotes IPv6 nas redes 802.15.4;

- **RFC6282 ou draft-ietf-6lowpan-hc** – Define o formato da compressão dos pacotes IPv6 nas redes 802.15.4;

- **RFC6775 ou draft-ietf-6lowpan-nd** – Define optimizações do *Neighbor Discovery* para o 6LoWPAW;

- **RFC6550 ou draft-ietf-roll-rpl-19** – Define os protocolos de encaminhamentos dentro do IPv6 nas redes 802.15.4.

4.2 Arquitectura

O 6LoWPAN baseia-se na ideia de que a Internet é inteiramente construída em IP (*IP enable*). Significa que cada dispositivo *(Host) Low Power* deverá ter um IP tornando-se também uma parte no mundo da Internet ou "*Internet of Things*". A "*Internet of Things*" é criada através da conexão de várias ilhas de redes de dispositivos *low power*, em que cada ilha é uma rede independente da Internet, ou seja, os pacotes transmitidos na mesma rede não são transmitidos para redes diferentes, mas sim encaminhados através de um coordenador (*edge router*) até à rede pretendida [21].

Cada ilha *LoWPAN* partilha o mesmo prefixo (*prefix*) do endereço IPv6 (os primeiros 64 bits do endereço IPv6). A cada ilha é limitada a 64000 dispositivos, devido ao limite imposto pelo endereçamento utilizado pelo IEEE 802.15.4, que usa 16 bits de endereços para cada dispositivo na rede, obtendo uma identificação IPv6 única. [22]

Existem três tipos de redes 6LoWPAN e podem ser definidas como na Figura 22:

- **Ad-Hoc** – São redes que não estão conectadas à Internet e que funcionam sem qualquer tipo de infra-estrutura;

- **Simple** – São redes simples, interligadas a outras redes ou mesmo à Internet através de um coordenador;

- **Extended** – São múltiplas redes com arquitecturas simples interligadas por múltiplos coordenadores e ligadas a um *backbone* ou mesmo à Internet.

É importante referir que o coordenador tem um papel crucial no encaminhamento da informação, já que é o responsável pela conexão das redes, controlo de fluxo, fragmentação/desfragmentação bem como na compressão/descompressão dos pacotes IPv6 e registo dos endereços através do *Neighbor Discovery*. Nos casos em que a rede *LoWPAN* está ligada a uma rede IPv4, o coordenador possibilita também a interligação entre as duas redes, como se pode observar na Figura 22.

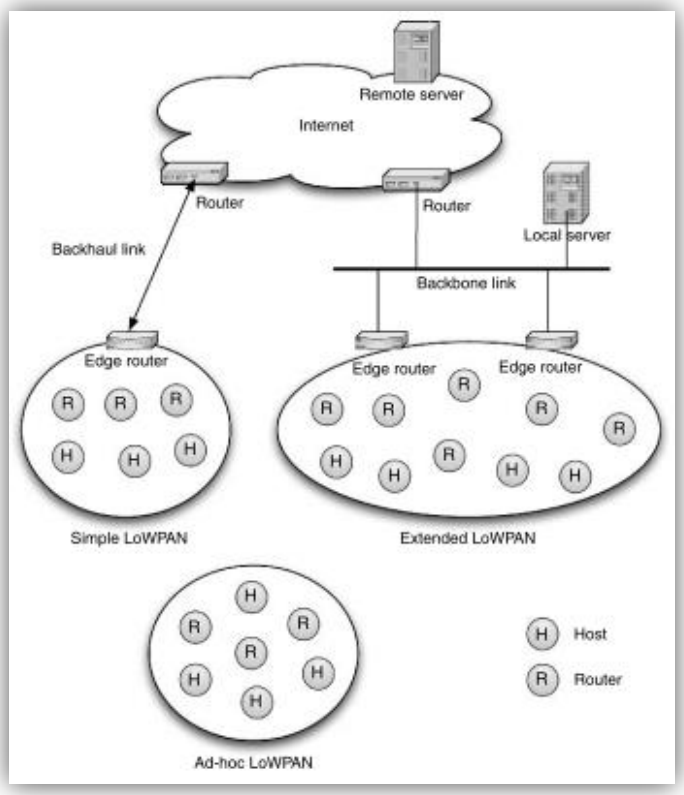


Figura 22 - Topologias da rede 6LoWPAN [21]

4.3 Motivação e problemas

Actualmente as RSSF são implementadas usando tecnologias proprietárias, dificultando a interoperabilidade entre diferentes redes e acessos à Internet. O protocolo IP associado às redes RSSF permite superar este problema, nomeadamente:

- O protocolo IP é livre e encontra-se definido pela IETF;
- A rede IP encontra-se implementada e é bem conhecida, o que permite uma ligação ambígua entre redes;
- Os dispositivos de baixa potência com conectividade IP, podem ser interligados à rede ou à Internet.

No entanto, apesar das vantagens do protocolo IP, existem vários problemas associados, tais como:

- O mínimo MTU (*Maximum Transmission Unit*) requerido na camada de ligação para transporte de pacotes IPv6 é de 1280 octetos [11]. Uma vez que o tamanho máximo de uma *frame* IEEE 802.15.4 é de 127 octetos, onde dos 127 octetos, 25 octetos são usados

para o cabeçalho MAC e, caso seja accionada segurança, podem ser adicionados mais 21 octetos, inclusive. Restam apenas 81 octetos disponíveis para a transmissão de pacotes IPv6. No entanto o pacote IPv6, sempre que possível, deve ser enviado em apenas uma *frame* IEEE 802.15.4, de modo a evitar fragmentação excessiva. A solução para este problema foi a criação de uma camada de adaptação (6LoWPAN) que se encontra descrito na Secção 4.4;

- Os mecanismos de segurança do IPv6 (IPsec) são demasiado complexos para serem usados nos dispositivos de baixa potência [23];

- As soluções actuais de encaminhamento, como RIP (*Routing Information Protocol)*, OSPF (*Open Shortest Path First*) entre outros são muito agressivas devido à constante troca de informação, especialmente nas redes *mesh*, sendo estas as mais utilizadas nas redes de baixa potência. Contudo já existe solução para este problema, com o protocolo RPL que se encontra descrito na secção 5.1;

- O Protocolo IP assume que os dispositivos encontram-se sempre conectados, no entanto, nas redes de baixa potência, os dispositivos por norma permanecem inactivos durante longos períodos de tempo, por forma a ter baixos consumos de energia.

4.4 Camada de adaptação

A Figura 23 mostra uma comparação entre a pilha IP e a pilha 6LoWPAN. Apesar de ter uma estrutura semelhante à pilha IP, o 6LoWPAN apresenta as seguintes diferenças:

- Apenas suporta IPv6, o qual foi adaptado de forma a ser suportado nas redes IEEE 802.15.4, como representado na Figura 24;

- O protocolo TCP (*Transmission Control Protocol*) RFC0793 foi removido devido às suas características complexas (orientado à conexão, controlo de fluxo, entre outros) que levam a um elevado consumo de energia. Todas estas características fazem com que o TCP não lide bem com a perda de pacotes nas redes wireless sendo o UDP (*User Datagram Protocol*) RFC0768 o protocolo mais apropriado;
- Na camada de rede, os dispositivos dentro da rede 6LoWPAN, comunicam usando a camada de adaptação e cabe apenas ao coordenador (*edge router*) a interligação entre as redes IPv6 e LoWPAN como representado na Figura 25.

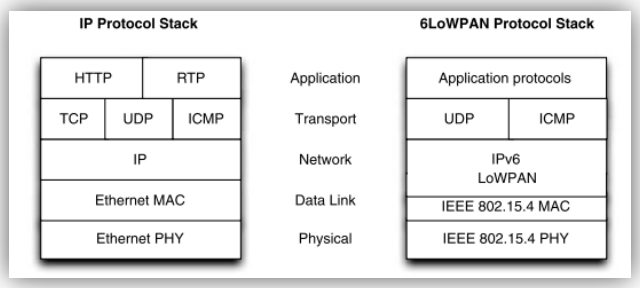


Figura 23 - Pilha IP e 6LoWPAN [21]

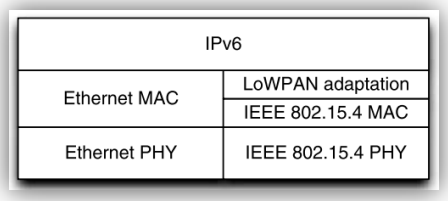


Figura 24 – Camada de adaptação 6LoWPAN [21]

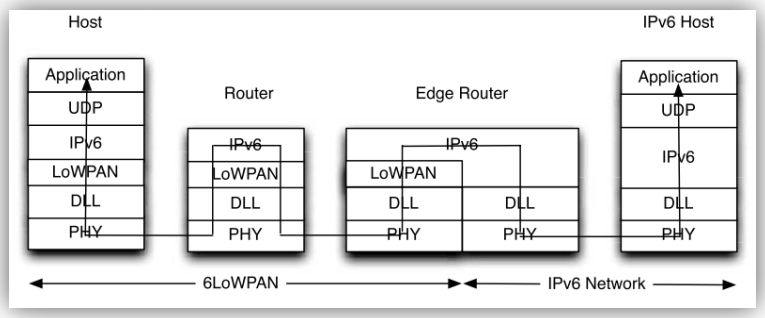


Figura 25 - Encaminhamento de uma rede 6LoWPAN para uma rede IPv6 [21]

A camada de adaptação apresenta três funções essenciais [21] [22] [23]:

- **Compressão do cabeçalho IPv6** – Alguns campos do cabeçalho IPv6 são omitidos do pacote quando é usada a camada de adaptação;

- **Fragmentação/desfragmentação** – Os pacotes IPv6 são fragmentados em múltiplas *frames* 802.15.4 de forma a acomodar o MTU mínimo requerido;

- **Encaminhamento *mesh* na camada de ligação** – De modo a suportar o encaminhamento IPv6 na camada de ligação lógica, a camada de adaptação pode enviar endereços na camada de ligação lógica até ao destino. Em alternativa, o encaminhamento através da camada de rede pode ser feito considerando que cada salto via radio 802.15.4 é um salto IP.

O resultado é um cabeçalho 6LoWPAN que pode apresentar as seguistes formas, ilustradas na Figura 26.

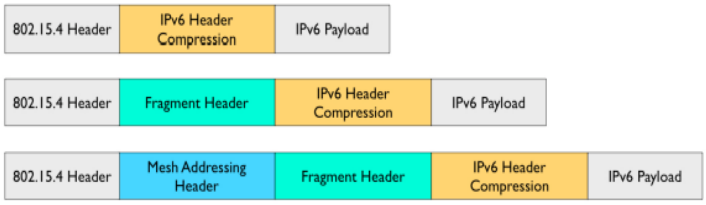


Figura 26 - Possíveis cabeçalhos 6LoWPAN [22]

O formato do pacote básico IEEE 802.15.4, ilustrado na Figura 3 da secção 2.6, não contém um campo de forma a identificar o MPDU transportado pelo pacote, o que significa que não existe qualquer tipo de informação que permita ao destinatário distinguir um pacote 6LoWPAN de qualquer outro tipo de pacotes ou mesmo de diferentes tipos de redes 6LoWPAN. Desta forma, a camada de adaptação deve fornecer um campo de identificação único (cabeçalho), que não é fornecido pelo próprio IEEE 802.15.4 [21].

O cabeçalho 6LoWPAN é identificado pelos primeiros bytes mais significativos, como apresentado Tabela 6, e deverão aparecer na seguinte ordem: cabeçalho *Mesh*, cabeçalho *Broadcast*, cabeçalho *Fragment* e *Dispatch* IPv6 [24]. Os pacotes IPv6 são encapsulados como mostra a Figura 27, e os três primeiros cabeçalhos (*Mesh*, *Broadcast e Fragment*) são adicionados de acordo com os requisitos, sendo o cabeçalho IPv6 comprimido ou descomprimido mediante o valor do campo *Dispatch* [24] [25] .

802.15.4 Hdr	Mesh Hdr	Broadcast Hdr	Fragment Hdr	Dispatch	IPv6 Hdr	Pay-load	FCS

Figura 27 - Encapsulamento do pacote IPv6 [25]

Dispatch	Header Type	
00 xxxxxx	NALP	Not a LoWPAN frame
01 000001	IPv6	Uncompressed IPv6 Addresses
01 000010	LOWPAN_HC1	LOWPAN_HC1 compressed IPv6
01 010000	LOWPAN_BCO	LOWPAN_BC0 Broadcast
01 111111	ESC	Additional Dispatch byte follows
10	Mesh header	Mesh header
11000	Frag1	The first Fragment
11100	FragN	Subsequent Fragments

Tabela 6 - Tipos de cabeçalhos 6LoWPAN [24] [25]

4.4.1 Endereçamento

O endereçamento nas redes 6LoWPAN funciona como em qualquer rede IPv6, permitindo na camada de adaptação usar dois tipos de endereços:

- **Endereços da camada de ligação lógica** – Estes endereços são usados por todos os dispositivos e poderão conter endereços de 16 bits e de 64 bits;
- **Endereços da camada de rede** – Apenas o coordenador contém endereços da camada de rede, uma vez que faz a ligação entre a rede IPv6 e 6LoWPAN.

4.4.2 Compressão do cabeçalho IPv6

De modo a ter uma maior eficiência numa rede 6LoWPAN, todos os pacotes IPv6 devem ser utilizados apenas num pacote IEEE802.15.4, de modo a evitar a fragmentação e a desfragmentação dos pacotes. Para isso são usados dois tipos de compressão, apresentados na Figura 28 [21] [24]:

a) LOWPAN_HC1 – Utilizado para a compressão do cabeçalho IPv6. Este tipo de mecanismo funciona muito bem nas comunicações *unicast*. O cabeçalho é demostrado na Figura 28;

b) LOWPAN_HC2 – Utilizado para a compressão do cabeçalho UDP, este também pode ser usado opcionalmente em pacotes com a compressão HC1.

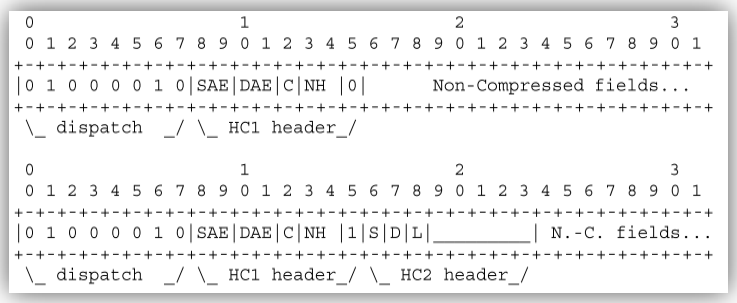


Figura 28 - Compressão HC1 do cabeçalho IPv6, com e sem compressão HC2

Muitos dos campos do cabeçalho IPv6, apresentado na Figura 18 na Secção 3.3, são desnecessários e outros são impossíveis de comprimir [21] [22] [25].

Os campos possíveis de comprimir são:

a) *Traffic Class e flow label* – Estes campos normalmente são zero, no entanto se necessário, o campo C do LOWPAN_HC1 pode estar inactivo (0) e os campos não são enviados, ou activo (1), sendo neste caso incluído nos campos *non-compreessed*;

b) *Payload lenght* – Visto que pode ser obtido ou pela *frame* da camada de ligação lógica ou pelo mecanismo de fragmentação, não existe necessidade de o enviar num cabeçalho comprimido;

c) *Next header* – Este campo é usado para indicar o tipo do próximo cabeçalho (UDP, TCP ou ICMP) e usa 8 bits. De forma a reduzir este campo o LOWPAN_HC1 usa o campo NH de dois bits como mostra a Tabela 7.

Bits NH	*Next header*
00	Enviado no campo *non-compreessed*
01	UDP
10	ICMP
11	TCP

Tabela 7 - Valores do campo NH do LOWPAN_HC1

Os campos impossíveis de comprimir são enviados no campo *non-compressed*, que se encontra no final do HC1 e/ou HC2, são eles adicionados na seguinte ordem:

1) *Version* – Como se trata sempre de uma rede 6LoWPAN, a versão é sempre 6, logo não é necessário enviar;

2) *Hop Limit* – Este campo foi considerado muito difícil de comprimir, por isso é sempre enviado no mesmo campo, sendo sempre o primeiro;

3) Prefixo do endereço de origem (64 bits) - se o bit mais significativo do SAE (*Source address enconding*) for zero;

4) Prefixo do endereço da interface ID de origem (64 bits) – se o bit menos significativo do SAE for zero;

5) Prefixo do endereço de destino (64 bits) - se o bit mais significativo do DAE *(Destination address enconding*) for zero;

6) Prefixo do endereço da interface ID de destino (64 bits) – se o bit menos significativo do DAE for zero;

7) ***Traffic Class*** **(8 bits)** ***e flow label*** **(20 bits)** – se o C for zero;

8) ***Next header*** **(8 bits)** – se o NH for zero;

9) Qualquer campo ***non-compressed*** **do HC2** - se for necessário. Qualquer cabeçalho que venha neste campo ou *payloads*, não são sujeitos à compressão. Para melhor entender o que foi dito anteriormente, a Tabela 8 mostra os valores que os campos SAE e DAE podem ter e as consequências desses valores.

Valores do SAE ou DAE	Prefixo	Interface ID (IID)
00	Enviado	Enviado
01	Enviado	Omitido e derivado da camada de ligação logica ou pelo endereço *mesh*
10	Omitido e assume ser *link-local* (FE80::/64)	Enviado
11	Omitido e assume ser *link-local* (FE80::/64)	Omitido e derivado da camada de ligação logica ou pelo endereço *mesh*

Tabela 8 - Valores do campo SAE e DAE do HC1

De modo a entender o nível desta compressão, a Tabela 9 compara o tamanho dos campos do cabeçalho IPv6 com os campos comprimidos do 6LoWPAN.

Header Field	IPv6 header length	6LoWPAN HC1 length	Explanation
Version	4 bits	------	Assuming communicating with IPv6.
Traffic class	8 bits	1 bit	0 = Not compressed. The field is in full size. 1 = Compressed. The traffic class and flow label are both zero.
Flow label	20 bits		
Payload length	16 bits	------	Can be derived from MAC frame length or adaptation layer datagram size (6LoWPAN fragmentation header).
Next header	8 bits	2 bits	Compressed whenever the packet uses UDP, TCP or Internet Control Message Protocol version 6 (ICMPv6).
Hop limit	8 bits	8 bits	The only field always not compressed.
Source address	128 bits	2 bits	If Both source and destination IPv6 addresses are in link local, their 64-bit network prefix are compressed into a single bit each with a value of one. Another single bit is set to one to indicate that 64-bit interface identifier are elided if the destination can derive them from the corresponding link-layer address in the link-layer frame or mesh addressing header when routing in a mesh.
Destination address	128 bits	2 bits	
HC2 encoding	------	1 bit	Another compression scheme follows a HC1 header.
Total	40 bytes	2 bytes	Fully compressed, the HC1 encoding reduces the IPv6 header to two bytes.

Tabela 9 - Comparação dos campos do cabeçalho IPv6 com os campos comprimidos do 6LoWPAN [26]

4.4.3 Cabeçalho de Fragmentação

Os pacotes IPv6 vêm numa variedade de tamanhos, do mais pequeno, não sendo muito útil, de 40 bytes, para o maior que juntamente com o *payload* poderá conter mais de 4295032830 = $(2^{32} - 1) + (2^{16} - 1)$ = (Cabeçalho IPv6 + *Payload* IPv6) bytes, o que muito possivelmente não acontecerá numa rede 6LoWPAN. Na maioria das redes, é definido um MTU por forma a impor o tamanho máximo do pacote, de modo a que este seja transportado eficientemente. Para que os pacotes IPv6 sejam transportados pela rede IEEE802.15.4 foi definido o tamanho máximo de 127 octetos, uma vez que um pacote IPv6 completo, de 1280 octetos de tamanho máximo, não encaixa numa *frame* 802.15.4 [24].

No cabeçalho de fragmentação mostrado na Figura 29, os primeiros bits mais significativos definem se pertence à primeira fragmentação ou às próximas, como apresentado na Tabela 6. O *datagram_size* define o tamanho total do pacote IPv6 não fragmentado e o *datagram_tag* identifica o pacote, permitindo que todos os fragmentos com o mesmo *datagram_tag* sejam associados e desfragmentados para gerar o pacote. Por fim o *datagram_offset* é o único que apenas é adicionado a partir do segundo fragmento inclusive.

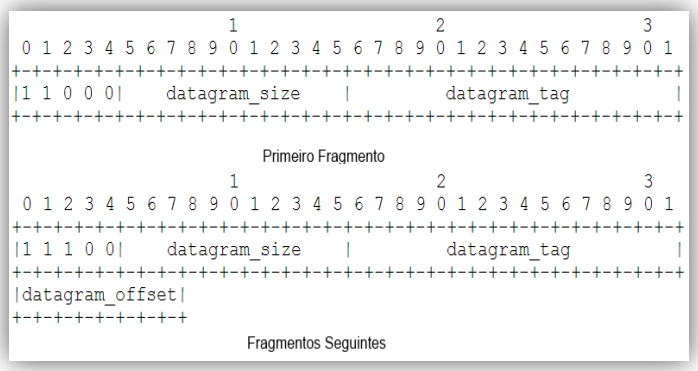


Figura 29 - Cabeçalho de fragmentação [24]

4.4.4 Cabeçalho *mesh*

O cabeçalho *mesh* é usado quando existe necessidade de enviar informação para múltiplos dispositivos que suportem encaminhamento na camada de acesso ao meio. Tal como se pode observar na Figura 30, os primeiros dois bits mais significativos são sempre 10, em consonância com o representado na Tabela 6. Seguem-se os 5 campos:

a) V – Este campo refere-se ao tamanho de bits do endereço de origem. Se for zero, o endereço destino é de 64 bits, se for 1 é de 16 bits;

b) F – Este campo refere-se ao tamanho de bits do endereço de destino. Se for zero, o endereço destino é de 64 bits, se for 1 é de 16 bits;

c) HopsLft – Este campo, também conhecido por *Hop Limited*, é semelhante ao do IPv6, limitando o número de saltos para o encaminhamento. Este é decrementado por cada nó e no caso de chegar a zero, o pacote é descartado;

d) *Originator address* – Este campo refere-se ao IP origem;

e) *Final address* – Este campo refere-se ao IP destino.

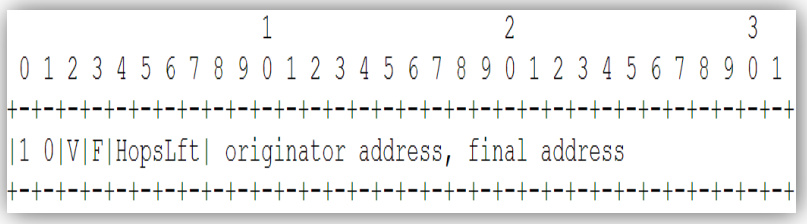


Figura 30 - Cabeçalho mesh [24]

4.5 Sumário

O grupo de trabalho 6LoWPAN da IETF une esforços no sentido que seja possível a comunicação pelo protocolo IPv6 nas redes IEEE 802.15.4. As questões mais pertinentes passam pela fragmentação/desfragmentação e a compressão dos cabeçalhos IPv6. A MTU máximo do IPv6 é de 1280 octetos, enquanto no IEE 802.15.4 é de 127 octetos, tornando-se importante o uso de fragmentação e desfragmentação. Normalmente nas redes LoWPAN o cabeçalho IP e UDP contêm muitos campos desnecessários e outros que também podem ser comprimidos. Portanto é importante e indispensável a compressão dos cabeçalhos IPv6 para o sucesso e futuro do uso de IPv6 nas RSSF. O 6LOWPAN cria uma camada de adaptação entre o IEEE 802.15.4 e o IPv6, com cabeçalhos específicos que podem ser adicionados ou removidos mediante a sua necessidade, permitindo que seja apenas enviado o que realmente é útil.

5. O Protocolo RPL (Routing Protocol for Low Power Lossy Networks)

"Tudo o que chega, chega sempre por alguma razão..."

Fernando Pessoa

5.1 RPL (*Routing Protocol for Low Power Lossy Networks*)

A necessidade de desenvolvimento de uma solução de encaminhamento *standard* levou a IETF a formar um novo grupo, denominado ROLL (*Routing Over Low Power and Lossy Networks*) para estudar os requisitos do encaminhamento no mundo da "*Internet of Things* " em inúmeras variedades de aplicações, tais como redes urbanas, automatização industrial e automatização de casas e edifícios. Também analisa os protocolos de encaminhamento existentes e como desenvolver um protocolo de encaminhamento para as redes LR-WPAN.

O resultado foi a criação do "*Ripple"* ou RPL (*Routing Protocol for Low Power and Lossy Networks*) um protocolo de encaminhamento pró-activo que constrói as suas rotas e topologia em intervalos aleatórios. A topologia é a base do funcionamento do RPL, este usa a topologia como um Grafo Acíclico Direccionado ("*Directed Acyclic Graph*"-DAG) para criar um ou mais destinos orientados (*Destination Oriented DAGs*" (DODAGs), que por sua vez podem estar associados a uma ou mais Instâncias RPL *"RPL Instances*". Cada RPL *Instance* possui uma topologia única, identificada pelo seu próprio ID (RPLInstanceID). Cada nó tem o seu próprio *rank* e para além disso, cada instância RPL é construída através do seu próprio conjunto de diferentes requisitos, das suas DAGs, como se pode observar na Figura 31 e descritos a seguir [27] [28] [29]:

- **Métricas / restrições** – Definem como um nó vai afectar o encaminhamento. É usado para calcular o *rank* de cada nó e decidir o/os nós pais (*parents)* preferidos, ou seja, as métricas/restrições ajudam a determinar o melhor caminho possível sendo as seguintes, as mais usadas:
  - **a)** Contagem de saltos;
  - **b)** Energia;
  - **c)** Carga.

- **Rank** – O *rank* descreve a "distância" lógica de um nó, a partir do nó root (de *rank* igual a zero), dentro de cada DODAG. A atribuição do *rank* é feita na formação da DODAG, em que cada nó selecciona o seu o/os seus nós pais (*parents nodes*) a partir do/dos seus nós vizinhos (*neighbors nodes*). Depois, cada nó calcula o seu próprio *rank* baseando-se nos seus pais, sendo este maior do que o *rank* dos seus pais. É bom referir que o *rank* não é necessariamente uma "distância" lógica ou mesmo uma distância relacionada ao número de saltos, mas sim á métrica escolhida;

- **Função objectiva** – A "*Objective function*" combina as métricas e restrições para calcular o melhor caminho (selecção de pais) e cálculo do *rank*, sendo importante realçar que a existência de varias funções objectivas na rede leva ao recálculo sistemático do "melhor caminho", levando a um maior consumo de energia devido ao processamento. Desta forma, não poderá existir mais do que uma função objectiva na mesma instância RPL. [29] [30] [31]

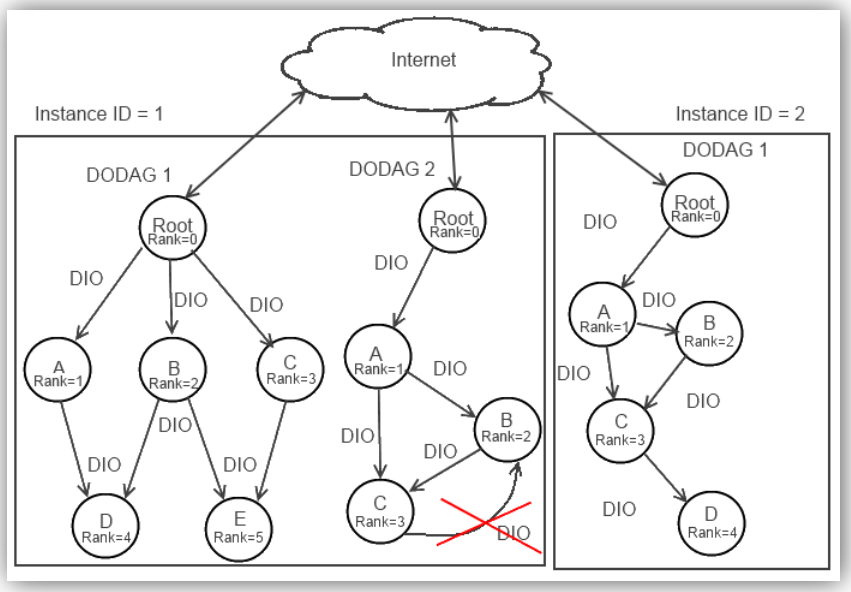


Figura 31 - RPL Instance [27]

5.2 Processo de criação de uma *Destination Oriented DAGs*

Apesar de poderem existir múltiplos nós roots na mesma instância RPL, o processo de criação de cada DODAG é sempre o mesma. Inicia-se no nó root ou, como na maioria dos casos, no *border router*/coordenador, que é programado pelo administrador do sistema como root. O nó root contém os parâmetros de configuração da rede, essa configuração é agrupada num novo pacote de mensagem de controlo ICMPv6 do RPL, de nome DODAG *Information Object* (DIO), o qual é usado para disseminar a informação pelos seus vizinhos na rede, usando *link-local multicasting.*

Apenas os root (coordenadores) podem iniciar o envio de mensagens DIO e ficam com o *rank* zero, uma vez que a distância a ele próprio é zero. Os nós que recebem as mensagens DIO processam as mensagens baseando-se em diversas características (métricas, *Objective functions*, DODAGID, InstanceID, entre outras) para se juntar ou não à respectiva DODAG. Sendo natural que cada nó receba múltiplas

mensagens DIO, cada nó descarta as mensagens DIO dos nós com rank superior ao seu, calcula o seu *rank* através das *Objective functions*, actualiza a mensagem DIO, e envia para todos os seus vizinhos via *broadcast*, como mostra Figura 32.

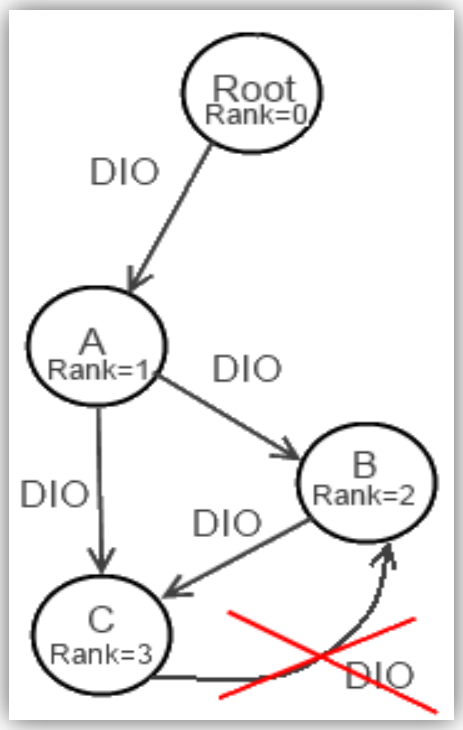


Figura 32 - Processos de criação da DODAG

Cada nó mantém um conjunto de vizinhos, que poderão ser possíveis candidatos para o nó pai *(parent node*) com classificação inferior ou igual, ou seja, vizinhos a partir dos quais recebeu uma mensagem DIO. No final, cada nó selecciona o seu nó pai *(parent node*), que tem um *rank* inferior ao seu, e que serve para encaminhar os dados até ao *root*, como mostra a Figura 32.

Este processo é contínuo, repetido e muito frequente até à formação total da DODAG, sendo depois menos frequente, apenas utilizando-se para manutenção da rede. Durante toda a formação da DODAG, os únicos valores que nunca mudam são a InstanceID e o DODAGID. Este processo encontra-se representado no fluxograma da Figura 33.

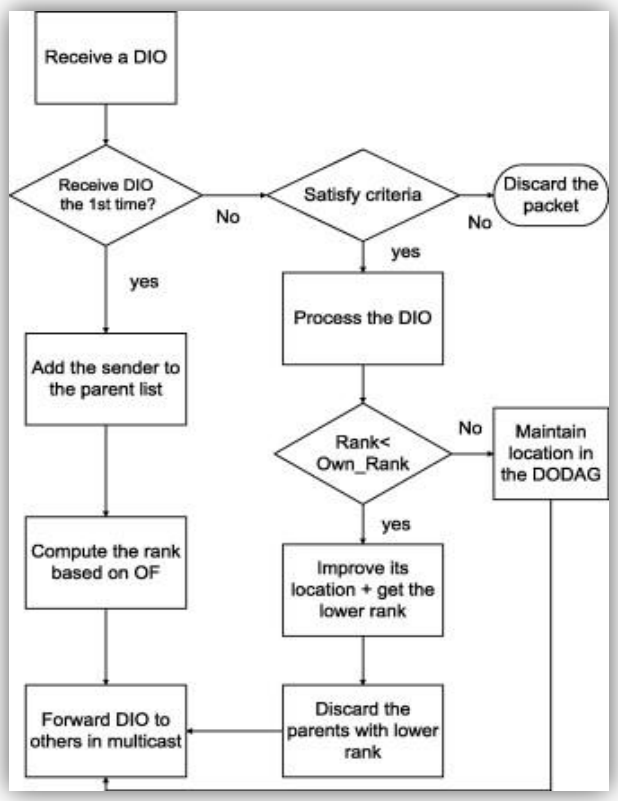


Figura 33 - Processo de criação de uma DODAG [27]

Apesar do *rank* normalmente não ser igual para todos os nós, é importante ter em conta que poderão existir casos de nós com o mesmo *rank*, uma vez que este é calculado pelo *Objective functions*. Este problema é resolvido através do mecanismo de prevenção de *loops* (*Loop Avoidance*), garantindo que não existem ciclos e que o pacote continua a ser encaminhado pela DODAG até o seu destino. [27] [29]

No final do processo, os nós não *root*, têm apenas um pai e uma vez que cada nó apenas comunica com o seu pai no processo de construção da DODAG, consegue-se assegurar uma DODAG livre de ciclos e que qualquer nó num raio de salto múltiplo para o root, consegue chegar ao seu pai. No entanto, o tráfego poderá ser originado fora da rede 6LoWPAN, ou mesmo dentro da rede para nós com *rank* superior (por exemplo *rank*=20) ao root *(rank*=0), sendo necessário que todos os nós conheçam as rotas até ao destino, permitindo encaminhamento para os vários destinos dentro do DODAG.

As mensagens DAO *(Destination Advertisement Object)* permitem anunciar o prefixo/ informação de destino até ao root (até ao cimo da DODAG) por forma a concluir o processo de criação total do DODAG, como mostra a Figura 34.

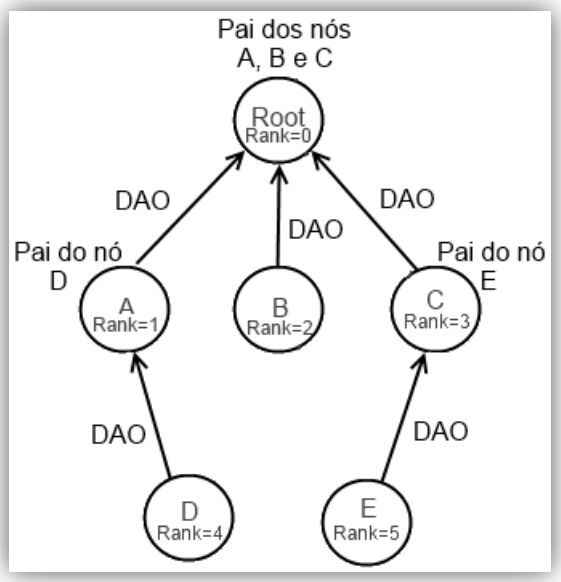


Figura 34 - Modo de envio das mensagens DAO [27]

5.3 Mensagens de controlo

No tópico anterior foram apenas apresentados vagamente dois tipos de mensagens de controlo (DIO e DAO), para a criação da DODAG. No entanto, o RPL faz uso de mais uma mensagem de controlo DIS (DODAG *Information Solicitation*).

Tal como é descrito nas especificações no ICMPv6 [14], as mensagens de controlo RPL são constituídas por um cabeçalho ICMPv6 seguido por uma mensagem base e opções, como mostra a Figura 35. Neste tipo de mensagens, o campo *Type* é sempre 155, pois define as mensagens de controlo RPL (*RPL Control Message*) e o campo *Code* identifica o tipo de mensagem de controlo, podendo ser consultado na Tabela 10. O *checksum* provém das especificações do ICMPv6 e o campo *Option(s)* é definido pelas mensagens de controlo de opções do RPL. Existe também a possibilidade de integrar o campo de segurança como exposto na Figura 36.

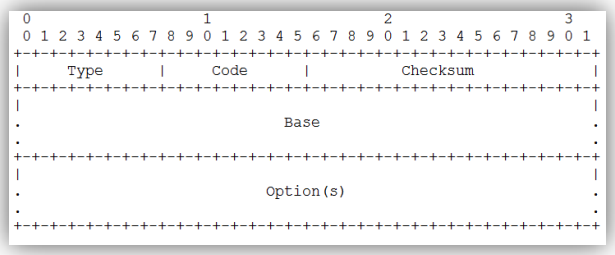


Figura 35 - Controlo de mensagem RPL básico [29]

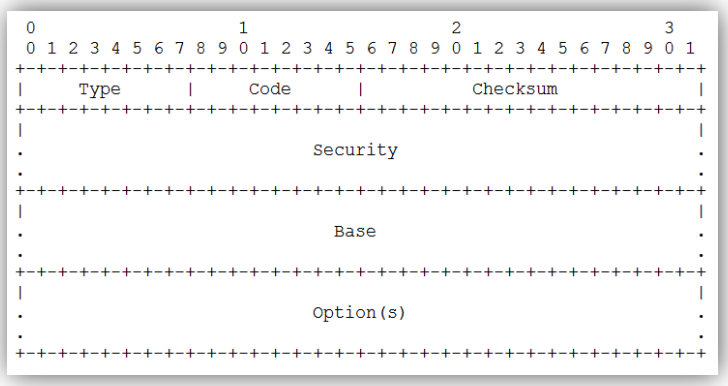


Figura 36 - Controlo de mensagem RPL com segurança [29]

Code	Tipo de mensagem de controlo RPL
0x00	*DODAG Information Solicitation (DIS)*
0x01	*DODAG Information Object (DIO)*
0x02	*Destination Advertisement Object (DAO)*
0x03	*Destination Advertisement Object Acknowledgment (DAO-ACK)*
0x80	*Secure DODAG Information Solicitation*
0x81	*Secure DODAG Information Object*
0x82	*Secure Destination Advertisement Object*
0x83	*Secure Destination Advertisement Object Acknowledgment*
0x8A	*Consistency Check*

Tabela 10 - Valor do campo Code e representação do tipo de mensagens de controlo RPL

5.3.1 DODAG Information Object

Para além do que já foi referido na Secção 5.2, é importante relembrar que as mensagens DIO transportam informação vital para que os nós descubram a sua instância RPL e recolham os parâmetros de configuração, podendo seleccionar um pai (nó preferencial) e manter a DODAG. O formato de uma mensagem DIO é apresentado na Figura 37.

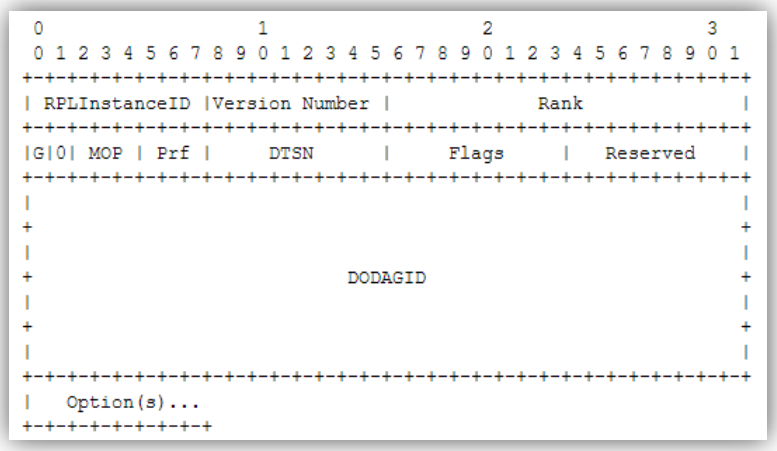


Figura 37 - Formato da mensagem DIO [29]

5.3.2 *Destination Advertisement Object*

Como visto na Secção 5.2, para estabelecer rota descendente (tráfego descendente *point-to-multipoint*) o RPL usa mensagens DAO, representado na Figura 38. Este tipo de mensagem propaga a informação de destino, tal como abordado no capítulo anterior e permite que os nós mais acima conheçam a rota até aos nós mais a baixo. Esta mensagem pode ser enviada por *unicast* de um nó com *rank* superior para o seu pai (*Storing Mode*) ou até ao nó root (*Non-Storing Mode*).

Eventualmente poderá ser necessário o envio de mensagens de conhecimento DAO-ACK. Essas mensagens são enviadas por *unicast* através de um pacote de um pai ou pelo *root* da DODAG em resposta a uma mensagem *unicast* do filho.

a) ***No-Storing Mode*** – Neste método nenhuma rota é guardada. Cada nó usa mensagens DAO para reportar a sua DAO com os respectivos pais até ao root. O root usa essa informação recebida para construir a rota descendente até ao nó baseando-se no IP de origem;

b) ***Storing Mode*** - Neste método cada nó deve conter memória necessária para processar o mecanismo de modo a criar e guardar as rotas nas tabelas de encaminhamento, uma vez que é necessário um estado de encaminhamento em cada nó. Em suma, cada nó recebe mensagens DAO, processa a informação do prefixo e adiciona-o à tabela de encaminhamento. O processo continua até que a informação do prefixo chegue ao *root*, completando assim o caminho do nó até ao *root* [29].

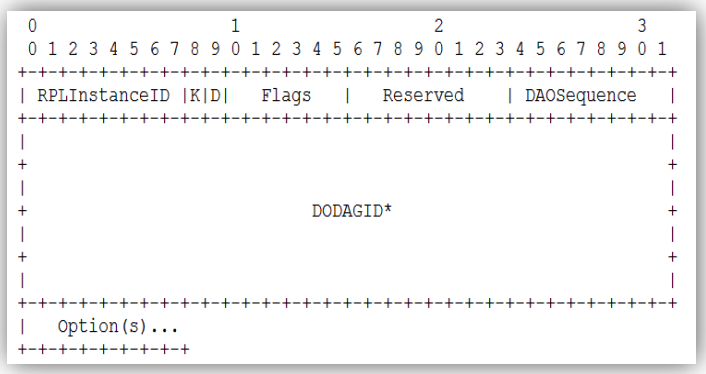


Figura 38 - Formato da mensagem DAO [29]

5.3.3 *DODAG Information Solicitation*

Este tipo de mensagem é usada essencialmente para manutenção da DODAG (exemplo: junção de um novo nó na DODAG). Permite iniciar o envio de uma mensagem DIO num nó, ou seja, em alternativa à espera de recepção de uma mensagem DIO, um nó faz um *broadcast* a todos os seus vizinhos com uma mensagem DIS, de forma a solicitar mensagens DIO para se configurar à DODAG.

5.4 *Trickle timer*

O *Trickle Timer* é usado para controlar o envio das mensagens DIO e DAO. Este assegura que as mensagens são entregues, mesmo que haja perdas de pacotes durante a comunicação. Faz uso de *timers* dinâmicos e do mecanismo de *sequence number*, que permite uma redução no envio redundante das mensagens e melhores consumos energéticos, possibilitando ainda a detecção de possíveis perdas. Dependendo da estabilidade dentro da DODAG o intervalo de tempo do *Trickle* é menor e o envio de mensagens é maior (exemplo de instabilidade: quando um nó se liga a uma nova DODAG). Quando a DODAG estabiliza, as mensagens de controlo são enviadas com menor frequência e com o propósito de reduzir a sobrecarga na DODAG [32].

5.5 Sumário

O RPL (*Routing Protocol for Low Power and Lossy Networks*) ou *Ripple* foi desenvolvido pelo grupo ROLL (*Routing Over Low Power and Lossy Networks*) como a solução para redes de baixa potência e baixas taxas de transmissão. Fornece um sistema livre de ciclos (*loop-free*) através de um sistema de *rank* e mecanismos de reparação. Baseia-se em métricas de encaminhamento e restrições para formar a topologia de rede (*DAG - Directed Acyclic Grap*) construindo a DODAG (*Destination Oriented DAGs*).A criação das DODAGs passa por 3 processos. O primeiro é a criação das rotas descendentes (do coordenador para qualquer dispositivo da sua rede DODAG), o segundo processo é a criação da rota ascendente (de qualquer dispositivo até ao coordenador) e por último a manutenção da rede.

6. O Sistema Operativo Contiki

"Devagar! Quem mais corre, mais tropeça!"

William Shakespeare

6.1 Conceitos básicos de sistemas operativos

O sistema operativo é um programa que age como um intermediário seguro e abstracto de modo a ocultar as diferenças entre as aplicações e o hardware, e consequentemente, disponibiliza uma plataforma consistente para correr aplicações, como representado na Figura 39. Por outras palavras, um sistema operativo é um programa que gere o hardware e fornece a base para as aplicações.

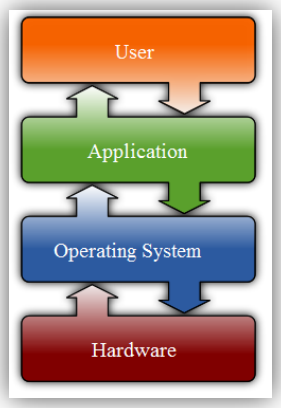


Figura 39 - Localização do Sistema Operativo

Tradicionalmente, os sistemas operativos são classificados como *single-task* (Tarefa única) ou *multitasking* (multi-tarefas), e *single-user* (Utilizador único) ou *multi-user* (multi-utilizadores) [21].

- **Um sistema operativo *single-task*** processa uma tarefa de cada vez, enquanto um sistema operativo *multitasking* consegue executar várias tarefas simultaneamente. Um sistema operativo *multitasking* exige uma grande quantidade de memória por forma a conseguir manter os estados das múltiplas tarefas, permitindo que as tarefas consigam executar em paralelo;

- **Os sistemas operativos *multi-user*** são todos capazes de suportar múltiplos utilizadores e mesmo esses, são também *single-user* mas apenas quando se encontram em modo de administração dos múltiplos utilizadores.

Os principais serviços de um sistema operativo incluem a execução simultânea e comunicação de vários programas, o acesso e gestão de dispositivos de Entrada e Saída (I/O), o armazenamento permanente de dados, controle e protecção do acesso ao sistema entre vários utilizadores no caso de sistemas operativos multi-utilizador [9]. Estes

aspectos são bons para o sistema operativo de uso geral (GPOS), pois facilitam o uso de um computador na perspectiva do utilizador. No entanto, GPOS só executam em computadores pessoais, *workstations*, *mainframes* ou em sistemas com grande quantidade de memória e requisitos de tempo real muito leves.

Nas RSSF, os dispositivos têm memória limitada, alguns são *diskless* (não tem suporte de memória física, exemplo disco rígido), outros fazem uso de memória externa, por exemplo os cartões flash e muitas vezes executam tarefas tão simples que não existe necessidade de um sistema operativo. Nestes casos, os dispositivos são programados usando sistemas *foreground/background* .

- ***Foreground/background*** – Representados na Figura 40, são sistemas que têm um programa de *loop* infinito a correr em background e que pode chamar várias funções ou operações não críticas, enquanto as operações críticas são executadas em primeiro plano nas rotinas de serviço de interrupção (ISR "*Interrupt Service Routines*");

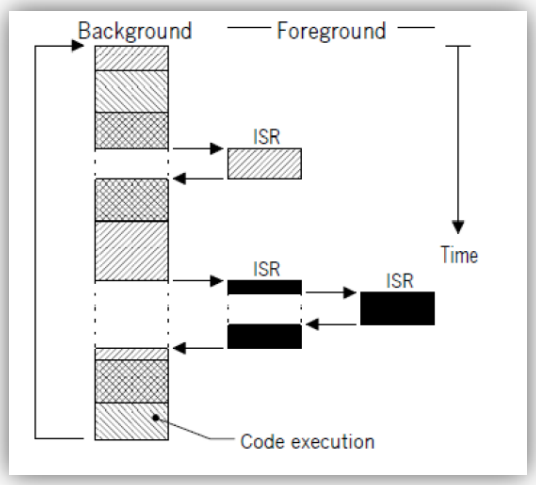


Figura 40 - Foreground/background

- **Multitasking** – Muito utilizado nos sistemas operativos, é um método onde existe múltiplos processos que são executados durante o mesmo período de tempo. Os processos partilham os mesmos recursos disponíveis, como o CPU e a memória principal. É semelhante ao sistema *foreground*/*background* mas com múltiplos programas em *backgound*. Neste tipo de programação podemos diferenciar dois géneros de gestão de tarefas, *cooperative* e *preemtive* (cooperativo e preemptivo).

Existe também o *multitasking multithreading*, permitem que o dispositivo execute mais do que uma tarefa de um único programa, mas devido às limitações dos dispositivos não é utilizada.

a) *Cooperative multitasking* – Permite que sejam executadas múltiplas tarefas em simultâneo, no entanto a tarefa que se encontra em primeiro plano ganha o controlo sobre o processador e mantém o controlo até à sua conclusão.

b) *Preemptive multitasking* - Ao contrário do anterior, este não permite a monopolização do processador, interrompendo periodicamente a execução da tarefa, passando o controlo para outra tarefa que se encontra em espera.

Nas RSSF são normalmente empregues sistemas operativos em tempo real, ou do inglês, *Real Time Operating System* (RTOS), devido aos requisitos especiais abordados anteriormente na secção 2.2, já que os dispositivos das RSSF têm requisitos especiais, tais como [1]:

- **Operações/tarefas em tempo real;**
- **Acesso a periféricos;**
- **Robustez.**

No mundo das redes LoWPAN existem vários sistemas operativos que possuem modelos de programação que facilitam o desenvolvimento de aplicações que interagem com o hardware. Podem ser citados os seguintes modelos de programação:

- ***Thread-based Programming*** – Contextualizada na Figura 41, usa múltiplas *threads* de controlo dentro de um único programa ou espaço de endereço único. Isto significa que, cada *thread* tem uma pilha separada e independente das outras, apesar de cada uma ter o seu próprio âmbito. Isto simplifica a programação, porque tem um fluxo de execução linear; no entanto têm a desvantagem de consumir mais recursos.

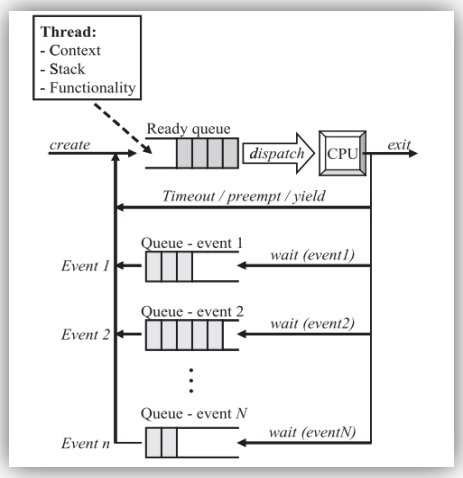


Figura 41 - Modelo Thread-based

- **Event-based Programming** – É uma programação baseada em eventos e gestores de eventos (*event handlers*). As tarefas competem pelo acesso a uma única pilha partilhada, são ordenadas dentro da pilha numa "fila de espera" e só podem sair da pilha através de eventos accionados por interrupções feitas pelo hardware. É importante ter em conta os requerimentos das RSSF, é por isso que neste modelo as tarefas têm que ser pequenas e o programador tem que ter em conta quais as tarefas com mais prioridade.

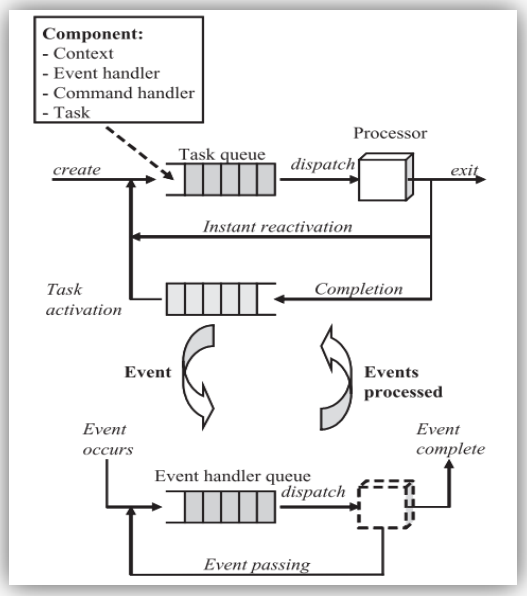


Figura 42 - Modelo Event-based [1]

Existem muitos RTOS utilizados em RSSF, no entanto apenas dois (TinyOS e o Contiki) são os mais usados. O Contiki foi utilizado neste projecto por ter uma boa comunidade, documentação, actualizações frequentes e implementação do RPL.

6.2 Sistema Operativo Contiki

O Contiki é um sistema operativo "*open source*" popular, especialmente desenhado para sistemas embebidos e dispositivos com pouca memória (sensores) que são usados em redes de baixa potência e com perdas. Desenvolvido na linguagem de programação C standard por Adam Dunkels, no grupo *Networked Embedded Systems do Swedish Institute of Computer Science*, tem sido adaptado numa variedade de plataformas de hardware diferentes como por exemplo, 8051, MSP430 e AVR [33]. Também inclui a adaptação para o microcontrolador ATmega128RFA1 utilizado neste trabalho.

As várias restrições em termos de memória e poder de computação foram ponderadas na construção da arquitectura do Contiki. O sistema é todo modularizado de forma a manter o núcleo o mais leve possível e um consumo de memória mínimo tanto quanto possível, de forma a evitar a segmentação da memória. As aplicações são dinamicamente carregadas e descarregadas em tempo real num *kernel event-driven*.

Ao contrário de outros SO, as aplicações são escritas com um método próprio, intitulado *protothreads*, que permite ter um estilo de programação leve, simples e fácil. Uma *protothread* pode parecer uma *thread* mas internamente é totalmente baseada em eventos. Sem qualquer variável local, a *thread* é invocada através de um evento, tal como, a alteração de um dado de um sensor (evento produzido pelo sensor), recepção de um pacote no dispositivo de rede (evento produzido pelo dispositivo de rede), términos de um temporizador (evento produzido por exemplo, pelo relógio do sistema), entre outros eventos [34] . O uso de *protothreads* será explicado na Secção 6.9

O Contiki utiliza os compiladores e ferramentas da GNU, como o GNU make, juntamente com as bibliotecas das plataformas, para compilar o código fonte das aplicações e exemplos, levando a que uma típica configuração utilize apenas 2Kb de RAM e 40Kb de ROM, para todo o sistema operativo. Ao nível da comunicação, seja em IPv4 ou IPv6, o Contiki faz de uso de duas pilhas separadamente [35]. O µIP é uma pilha TCP/IP pequena que possibilita a comunicação do Contiki com a Internet e a pilha de comunicação RIME, desenvolvida para transmissões rádio de baixa potência, permitindo a comunicação dentro das redes LoWPAN [36] [37]. Apesar de serem duas pilhas separadas, as aplicações podem usar, uma das duas pilhas, ambas ou nenhuma dependendo da aplicação.

6.3 Arquitectura do sistema

Uma aplicação que utiliza Contiki é constituída pelo *kernel* do SO, bibliotecas, processos e a aplicação, sendo tudo compilado e carregado no dispositivo. Os processos são aplicações do programa ou serviços, onde os serviços podem ser usados por uma ou mais aplicações. Ambos podem ser dinamicamente substituídos em tempo real através de eventos. O *kernel* não implementa uma camada de abstracção para a comunicação entre os processos e o hardware. Em vez disso, permite que os *drivers* dos dispositivos e as aplicações comuniquem directamente com o *hardware*, como demonstrado na Figura 43.

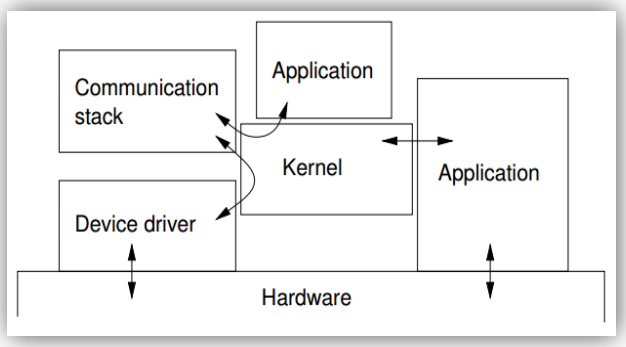


Figura 43 - Interacção do sistema Contiki [37]

Um processo consiste num bloco de código, uma parte da memória RAM e uma função *event handler*. A gestão do estado do processo, que é mantido numa memória privada é feita pelas funções *event handler* e *pool handler*, que definem uma forma para que o *kernel* apenas mantenha um ponteiro para o estado do processo. Todos os processos partilham o mesmo espaço de endereço e não correm em domínios de protecção diferentes, sendo a comunicação entre processos feita por eventos.

No final, quando o código fonte é compilado, este faz uma separação do sistema principal (core) e do programa que é carregado *(Loaded program*), como exposto na Figura 44. A aplicação é guardada numa memória não volátil e a informação é recebida e processada na RAM. O core do sistema, consiste no kernel do Contiki, *program loader*, pilha de comunicação com os drivers para a comunicação com o hardware, bibliotecas, serviços da linguagem de programação e plataforma usada (*Common Language Runtime*).

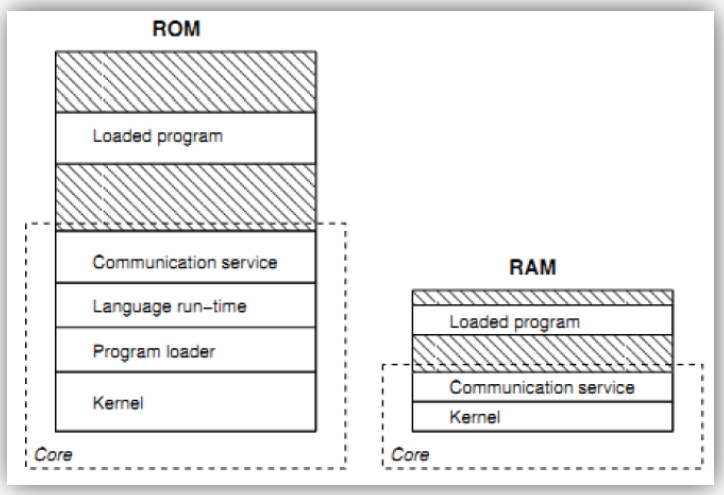


Figura 44 - Particionamento dos Core e da aplicação programada [37]

6.4 Kernel

O *kernel* do Contiki consiste num programador de eventos muito leve *(lightweight event scheduler*) que envia eventos para os processos que se encontram em execução e periodicamente chama os processos através do uso de *polling handlers*. Isto significa que consiste num *kernel event-driven* , onde as aplicações são dinamicamente carregadas e descarregadas em tempo real, sempre que algo aconteça (eventos como temporizadores, interrupções, pressionar um botão, entre muitos outros).

Uma vez que um *kernel event-driven* por si só, não suporta *threading*, de forma a consegui-lo, o Contiki emula o suporte *Multi-threading* através de directivas do pré-processador. O kernel suporta dois tipos de eventos:

- **Síncronos** – Este tipo de eventos são imediatamente processados;
- **Assíncronos** – Os eventos são colocados em fila pelo kernel e são mais tarde enviados para o respectivo processo.

Além destes eventos, o núcleo fornece um mecanismo de *polling*, onde o estado dos componentes do hardware é testado periodicamente. Durante esse tempo, os *polling handlers* que expressem interesse num dispositivo de hardware são notificados, de acordo com a sua prioridade. O mecanismo de *polling* pode ser visto como evento com prioridade mais elevada, sendo programados entre cada evento assíncrono, evitando assim falhas "*racing conditions*".

6.5 Processos

Serviços ou aplicações são considerados como processos, em que cada processo é uma função definida na linguagem C, que contém um ciclo (*loop*) infinito e algumas chamadas de bloqueio num fluxo sequencial. Este tipo de código permite uma programação em máquinas de estado, uma vez que cada processo pode ter uma ou mais chamadas de bloqueio diferentes e quando iniciado, este é executado até bloquear à espera de um evento.

6.6 Serviços

Um serviço é um processo que implementa as funcionalidades de outro processo e partilha os seus recursos para serem usados por outro processo podendo ser dinamicamente substituídos em *runtime* e por essa razão devem estar dinamicamente vinculados (por exemplo, a pilha de protocolos de comunicação e as *drives* de um equipamento). Os serviços são geridos por uma camada de serviços normalmente situada junto ao *kernel*. Cada serviço é identificado por uma *String* e pode ser localizado pela mesma, sendo possível a consulta dos serviços instalados.

A interface do serviço é composto por um número de versão e uma tabela de funções com ponteiros para as funções implementadas pela interface, como apresentado na Figura 45. Os programas usam um serviço através da ligação com biblioteca stub, caso um serviço não esteja na biblioteca stub é usada a camada de serviço para encontrar o processo do serviço. Uma vez encontrado, é criado um ponteiro para a interface do serviço e o serviço stub armazena o ID do processo para ser usado em pedidos futuros.

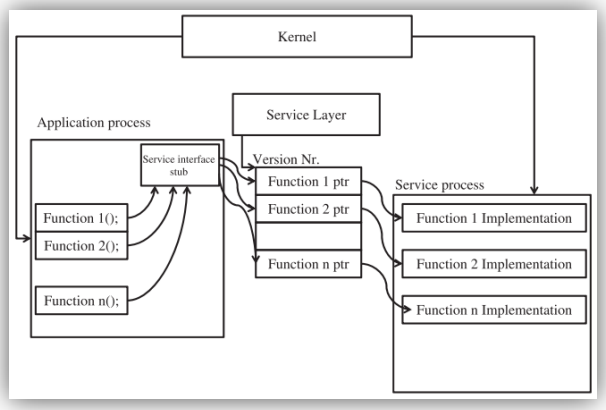


Figura 45 – Arquitectura da interacção dos serviços [37]

6.7 Bibliotecas

Para além do que o kernel fornece, um CPU *Multiplexing e polling handlers*, o resto do sistema é implementado como um sistema de bibliotecas ligadas com os programas, seja por ligação de bibliotecas que estão ligadas ao core, seja por bibliotecas que fazem parte da imagem do programa ou ainda por bibliotecas que permitem chamar serviços específicos.

6.8 Eventos

Um processo normalmente é executado durante algum tempo e depois espera que haja um evento, ficando bloqueado. Ao surgimento de um evento, o *kernel* desbloqueia o processo passando a informação sobre o evento. Os eventos podem ser classificados em:

- **Eventos Internos** – Este tipos de eventos são originados dentro do próprio processo ou outros processos, (por exemplo, enviar para outro processo informação para processamento e ter de esperar que a informação seja processada);

- **Eventos Externos** – Este tipo de eventos estão relacionados com as entradas e saídas do dispositivo, podendo originar interrupções que produzem eventos, (exemplo: quando o sistema rádio receber informação);
- **Evento temporizado** – Sempre que um processo estabelece um tempo para gerar um evento, este é bloqueado até o tempo expirar, desbloqueando-o. Exemplo: colocar um dispositivo em espera durante 30 seg. Nestes casos é importante relembrar que é necessário sincronismo, de forma a não haver falhas devido à *race conditions*.

A informação proveniente dos eventos encontra-se classificada por:

- **Informação** – Corresponde aos dados que um sensor ou processo envia para outro e vice-versa (exemplo, a informação de um sensor de temperatura);

- **Processo** – Quando um evento é executado, pode enviar informação sobre um processo específico ou todos os processos registados;

- **Tipo de evento** – O programador pode definir o tipo de eventos para os processos, de forma a diferenciá-los.

6.9 *Multi-threading, Event-driven e Protothreads*

Uma vez que a palavra-chave numa RSSF é o consumo, um modelo de programação baseado em *multi-threading* iria consumir elevada memória que levaria a efeitos em termos de custo, tamanho e consumo de energia. Isto porque, no sistema *multi-threading*, cada *thread* é alocada na pilha de execução em memória (*stack*), como apresentado na Figura 46.

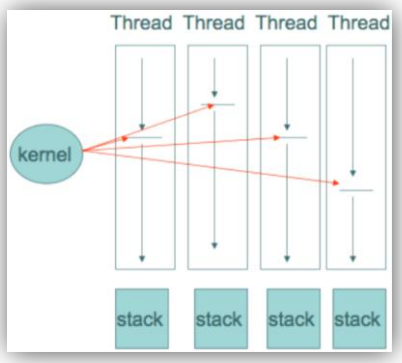


Figura 46 – *Multi-thread* [38]

De modo a obter-se um modelo de programação baseado em eventos e representado na Figura 47, todas as *threads* partilham a mesma pilha, o que permite minimizar o uso da memória, embora com o problema de não ser possível bloquear a execução de um programa enquanto se espera que ocorra um evento. Devido a este modelo apenas responder a eventos, isto implica a utilização de máquinas de estado para libertar o programa, o que aumenta a complexidade na programação, uma vez que o código é implementado de forma não sequencial.

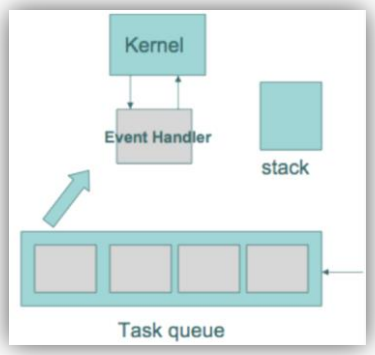
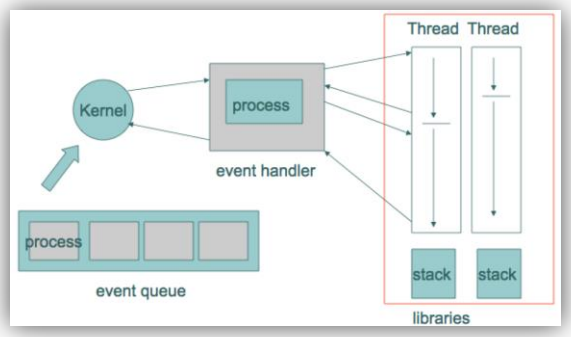


Figura 47 - *Event Driven* [38]

Para beneficiar das vantagens dos dois modelos, o Contiki faz uso de um mecanismo, denominado *Protothread*, que permite implementar blocos condicionais, mas com a possibilidade de partilhar a mesma pilha com outras *protothreads*. Apesar desta abordagem, o Contiki também suporta *threads* reais, num género de uma biblioteca, permitindo retirar vantagem da programação com *threads*, podendo ser analisado na Figura 48.



.

Figura 48 - *Protothread* [38]

Um processo é uma *protothread* que é chamada sempre que um processo recebe um evento, quando este é iniciado, através da função PROCESS_BEGIN(), este é bloqueado até receber um evento. Existem vários tipos de eventos, tais como:

- **PROCESS_BEGIN()** - Declara o início da *protothread*;
- **PROCESS_END()** - Declara o fim da *protothread*;
- **PROCESS_EXIT()** - Sai do processo;
- **PROCESS_WAIT_EVENT()** - Espera por um evento;
- **PROCESS_WAIT_EVENT_UNTIL()** - Espera por um evento mas com a condição definida no with;
- **PROCESS_YIELD()** - Espera por um evento, é o equivalente ao PROCESS_WAIT_EVENT();
- **PROCESS_WAIT_UNTIL()** - Espera para uma dada condição, podendo existir a possibilidade de não se obter o processo;
- **PROCESS_PAUSE()** - Temporariamente fica em espera.

Quando um evento ocorre, o *kernel* invoca o processo e este é executado, como pode ser analisado na Figura 49.

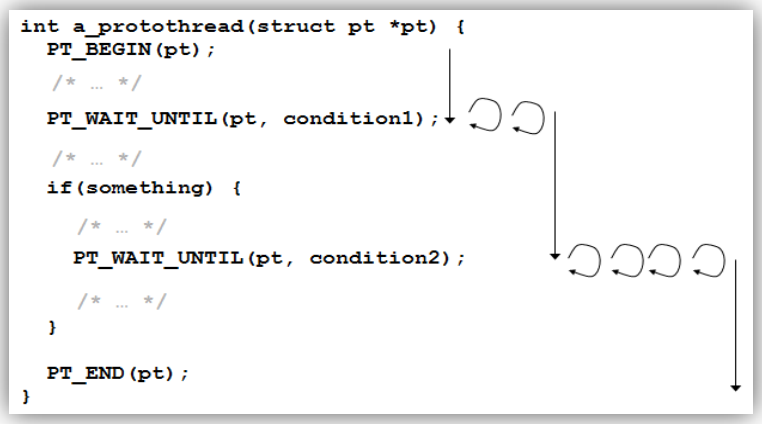


Figura 49 - Exemplo de código simples de uma Protothread [34]

Uma *protothread* pode implementar 3 padrões básicos:

- **Sequência** - A *protothread* é parada até que se verifique a condição, logo que a condição seja verdadeira, a *protothread* continua a sua execução, exemplo:

```
uma_sequencia:
        PT_BEGIN()
          (*codigo*)
        PT_WAIT_UNTIL(Condição)
          (*codigo*)
         PT_END()
```

- **Selecção** - A *protothread* apenas continua a execução depois da verificação de uma de duas ou mais condições;

```
uma_selecção:
       PT_BEGIN()
            (*codigo*)
            IF(condição1)
                 PT_WAIT_UNTIL(Condição2a)
            ELSE
                 PT_WAIT_UNTIL(Condição2b)
             (*codigo*)
        PT_END()
```

- **Interacção** - A *protrothread* permanece no mesmo estado até se verificar uma condição, de seguida é parada até que se verifique outra condição, logo que essa condição seja verdadeira, a *protrothread* continua a sua execução.

```
uma_interacção:
       PT_BEGIN()
         (*codigo*)
         WHILE(condição1)
            PT_WAIT_UNTIL(Condição2a) or (Condição2b)
            (*codigo*)
        PT_END()
```

6.10 *Instant Contiki*

O *Instant Contiki*, na versão 2.6 apresenta-se como uma máquina virtual com a distribuição *linux Ubuntu* pré-instalada, contendo algumas das ferramentas necessárias para instalar e testar uma rede de sensores utilizando o Contiki. Este possui para além do próprio SO Contiki, alguns exemplos: um simulador gráfico para redes de sensores, o *Cooja*, todas os aplicativos, ferramentas para ajuda no desenvolvimento (exemplo: ferramentas AVR) e aprendizagem de aplicações, incluindo as ferramentas utilizadas nesta dissertação.

.

6.11 Sumário

A maioria dos Sistemas Operativos necessitam de muita memória, alimentação eléctrica constante entre outras características, de forma a poder ser usado para diversas aplicações (como por exemplo edição de texto/imagem/vídeo/áudio, programação, visualização de filmes e jogos). No entanto, ao contrário desses SO, nas RSSF os SO são projectados para usufruir de recursos limitados e para executar um conjunto limitado de tarefas. Cada SO pode ser de uma única tarefa ou de múltiplas tarefas sendo essas tarefas programadas com base em eventos ou em threads.

O sistema operativo Contiki é *open source*, extremamente portátil, de múltiplas tarefas para redes de baixa potência e baixas taxas de transmissão. Este possui um kernel baseado em eventos, no qual as aplicações são dinamicamente carregadas e descarregadas em tempo real. Com duas pilhas de comunicação, consegue comunicar dentro das redes IEEE 802.15.4 com a pilha Rime e com a pilha µIP para as comunicações à Internet. Escrito na linguagem de programação C, pode ser executado numa variedade de plataformas. O código é compacto (*small footprint*) sendo tipicamente de 2 KB na RAM e 40KB na ROM, sendo um dos SO mais leves.

7. Implementação e testes

"Penso 99 vezes e nada descubro. Deixo de pensar, mergulho no silêncio e a verdade me é revelada."

Albert Einstein

7.1 Implementação

O sistema operativo escolhido foi o Contiki OS 2.6, já que têm uma completa compatibilidade com o protocolo 6LoWPAN e RPL para a plataforma AVR, sendo neste projecto usado o chip ATmega128RFA1 apresentado na Figura 50, como coordenador e nó da rede, sendo aos nós adicionado um sensor de luminosidade.

Figura 50 - Dispositivo usado no projecto

Numa fase inicial foi testada a comunicação *unicast* e anycast usando os respectivos protocolos. De seguida foram testadas as aplicações baseadas no protocolo de transmissão UDP, com capacidade de enviar todos os dados de qualquer nó para o coordenador periodicamente e opcionalmente, receber também pedidos do coordenador.

Devido à falta de um dispositivo que fizesse a ponte directa entre a rede IPv6 e a RSSF, foi testado e usado o protocolo SLIP (*Serial Line Internet Protocol*), que permite a ligação de um dispositivo através da porta serial do computador. Apesar do SLIP ter sido substituído pelo protocolo Point-to-Point (PPP), foi aperfeiçoado com a inclusão de mais recursos e sem a necessidade de configuração de endereço IP. Os utilizadores dos dispositivos LoWPAN, ainda assim, continuam a preferir o uso do SLIP para encapsular pacotes IP, devido ao seu pequeno *overhead* [39].

Por último, foi desenvolvida uma aplicação que monitoriza a luminosidade e estado de um LED, bem como o controlo do estado do LED de cada dispositivo da rede 6LoWPAN na Internet.

7.2 Ambiente de desenvolvimento

Graças ao sistema de desenvolvimento do Contiki (*Instant Contiki*), que contém o simulador *Cooja* e todas as ferramentas necessárias para o desenvolvimento das aplicações, não foram necessárias grandes configurações para obtenção de bom ambiente de trabalho. Para além do ambiente Linux (*Ubuntu*), foi possível testar o desenvolvimento de aplicações também no ambiente Windows (Windows 7).

7.2.1 Configuração do sistema (hardware e software)

Ao contrário do ambiente Linux, no Windows foi necessário a instalação das seguintes ferramentas pela seguinte ordem:

1. Emulador da consola do Linux no Windows – Cygwin;

2. O conjunto de executáveis AVR-WinAVR;

3. Ambiente de desenvolvimento Eclipse;

Seguindo-se a explicação de cada uma destas ferramentas.

a) Cygwin

O *Cygwin* é um programa que permite instalar no sistema operativo Windows um conjunto de programas e ferramentas que reproduzem a consola e ambiente de trabalho de um sistema Linux. Tal como sucede nestes sistemas, todos os programas e ferramentas que fazem parte do *Cygwin* são de código aberto e gratuito.

A instalação deverá ser feita na raiz do sistema. No final deverá conter o sistema em: *"c:\cygwin"*.

Aquando a escolha das ferramentas, poderá instalar todas as ferramentas (*Full instalation*) ou apenas as necessárias, sendo para este projecto consideradas as seguintes ferramentas:

- Automake;
- Make;
- GCC;
- GDB.

b) WinAVR

O WinAVR é um conjunto de executáveis, ferramentas de desenvolvimento de software livre Windows, para microcontroladores Atmel AVR. Este inclui o poderoso compilador GNU GCC para C e C++. A instalação deverá ser feita na raiz do sistema e no final deverá conter o sistema em: "`C:\WinAVR`".

Para outras distribuições Linux, será necessário a instalação das seguintes ferramentas: avr-libs e binutils.

```
$sudo apt-get install avr-libc
$sudo apt-get install binutils
```

c) Programa Avrdude

O Avrdude é um programa que se encontra dentro do WinAVR, no entanto foi escrito inicialmente para Linux, permitindo maior facilidade na programação dos microcontroladores Atmel AVR. Embora sem qualquer interface gráfica, a comunidade de software livre, entre outros, têm desenvolvido ferramentas GUI para este programa. Caso se pretenda instalar noutra distribuição Linux, deve ser utilizado o seguinte comando:

```
$ sudo apt-get install avrdude
```

d) Comando Make

Antes de usar qualquer exemplo/aplicação é importante usar o comando apresentado de seguida na pasta da aplicação:

```
$make savetarget TARGET=avr-atmega128rfa1
```

Este comando cria um ficheiro denominado "Makefile.target", contendo a plataforma previamente inserida e permitindo utilizar o comando make, sem necessidade de definir o tipo de plataforma que o Contiki irá usar para compilar. Útil para quem tem dispositivos com a mesma plataforma. Caso não use este comando, a compilação utilizará a plataforma nativa do Contiki.

Sendo apresentada a seguinte mensagem:

```
$ make
TARGET not defined, using target 'native'
```

Outro método de compilação do código, bom para quem tem vários dispositivos e necessita de compilar em várias plataformas, é a utilização do comando com a indicação da plataforma, por exemplo:

```
make TARGET=avr-atmega128rfa1 nomeDoApplicativo.hex
make TARGET=sky nomeDoApplicativo.hex
make TARGET=avr-raven nomeDoApplicativo.hex
```

e) SO Contiki

Por defeito o Cygwin cria uma pasta com o nome do computador e é essa a localização por defeito aquando a execução do aplicativo, podendo ser alterada através do ficheiro *passwd* localizado em: "`c:\cygwin\etc\passwd`".

O sistema operativo Contiki deverá ser instalado dentro da pasta do *cygwin*, por exemplo na pasta HOME. No final terá o seguinte aspecto.

```
C:\cygwin\home\contiki-2.6\
```

f) Term Terminal

De modo a visualizar os dados lidos na porta série foi usado o aplicativo Term Terminal, cuja janela de configuração é mostrada na Figura 51. As seguintes configurações foram (ou devem ser) utilizadas:

- **COM** – A disponível
- **Rate** – 38400
- **Data bits** - 8
- **Stop bit** – 1
- **Parity** - none
- **Handshaking** – none

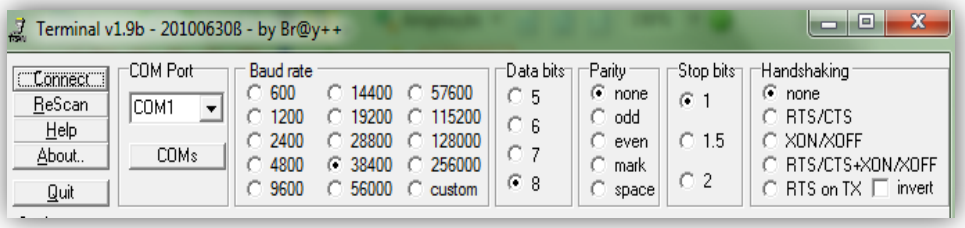


Figura 51 - Opções Term Terminal

7.2.2 Material necessário

Segue se uma breve descrição do material necessário à implementação de testes:

a) 2 Adaptadores usb-serial – Foram necessários 2 adaptadores de modo a testar, receber informação proveniente do coordenador e do nó da rede e carregar o aplicativo no dispositivo. Foi utilizado um adaptador *Prolific* USB-to-Serial *Comm Port* e USB-Serial CH340;

b) Interface serial no padrão RS232 – Este dispositivo liga directamente ao adaptador usb-serial ou à porta serial do mesmo padrão no computador (caso haja);

c) Microcontrolador – Foi usado o microcontrolador ATmega128RFA1 da AVR em todos os pontos da rede;

d) 1 Portátil – Foi usado na programação e *debug* dos dispositivos;

e) Sensor de luminosidade – Foi utilizado um sensor de luminosidade (LDR) de 200Kb para verificar o estado de iluminação;

f) Resistência – Foi necessário uma resistência de forma a controlar os valores de dados do sensor de luminosidade;

g) LED – O L, já incorporado no dispositivo, surge como um simulador de uma luz com a possibilidade de ligar e desligar;

h) Multímetro – Foi necessário para testar os valores e as ligações entre o dispositivo e o hardware ligado ao mesmo.

7.2.3 Problemas observados

Visto que os *Instant Contiki* já contêm um conjunto de ferramentas para o desenvolvimento e testes das aplicações desenvolvidas, não foi necessário nenhuma instalação adicional, apenas o uso de uma máquina virtual. No entanto apesar desta mais-valia, o simulador *Cooja*, ainda não permite a criação de simulações usando a plataforma ATmega128RFA1. Sendo o *Cooja* apenas usado para entendimento da execução dos exemplos, tal como testes simples.

Tal como no *Instant Contiki*, no ambiente Windows foi instalado o Eclipse para aceleração no processo de escrita e debug do código. Infelizmente muito dos trabalhos/ exemplos de conexão do Eclipse ao dispositivo, necessitavam de *drives* que no

momento de escrita da tese se encontravam obsoletos. A solução passou pelo uso simples da linha de comandos do emulador da consola do Linux, o Cygwin. Este tornou-se essencial numa primeira fase onde não existia a necessidade de configuração do túnel SLIP, mais tarde verificou-se a inexistência de uma aplicação funcional no Windows, por forma a criar o túnel SLIP.

7.3 Encaminhamento Linux e Windows

Configurou-se as interfaces do computador (Windows 7) e da máquina virtual (Ubuntu) de modo a criar uma ligação virtual entre duas máquinas separadas, podendo ser analisado na Figura 52.

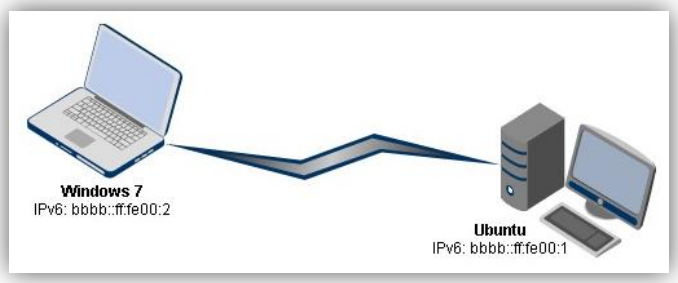


Figura 52 - Ligação entre computador e máquina virtual

Em ambos os casos foi necessário configurar o encaminhamento através da consola com o uso dos seguintes comandos:

Linux	
Opção	**Comando**
Adicionar endereço IPv6 estático	*sudo vi /etc/network/interfaces*
iface eth0 inet6 static pre-up modprobe ipv6 address bbbb::ff:fe00:1 netmask 64 gateway bbbb::ff:fe00:2	
Permitir o encaminhamento de pacotes IPv6	*sudo vi /etc/sysctl.conf*
net.ipv6.conf.all.forwarding=1	
Reiniciar a interface	*sudo /etc/init.d/networking restart*

Windows	
Opção	**Comando**
Adicionar endereço IPv6 estático	*netsh interface ipv6 set address 19 bbbb::ff:fe00:2*

7.3.1 Problemas

No caso do Windows, se o encaminhamento não for bem-sucedido, provavelmente devido a rotas pré estabelecidas, será necessário reforçar, usando o comando a seguir apresentado, permitindo que todos os pacotes IPv6 sejam encaminhados para a interface 19 (interface da maquina virtual no Windows), onde o próximo salto é o endereço bbbb::ff:fe00:1 (interface dentro da máquina virtual)

```
netsh interface ipv6 add route ::/0 19 bbbb::ff:fe00:1 metric=0
```

No final, confirmou-se a configuração dos endereços IPv6 na máquina virtual, Figura 53, endereço IPv6 no Windows, Figura 54 e o encaminhamento, que pode ser visto na Figura 55.

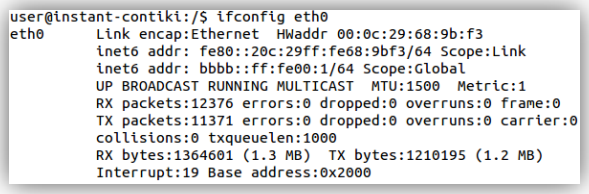


Figura 53 - Confirmação da configuração do endereço IPv6 na máquina virtual

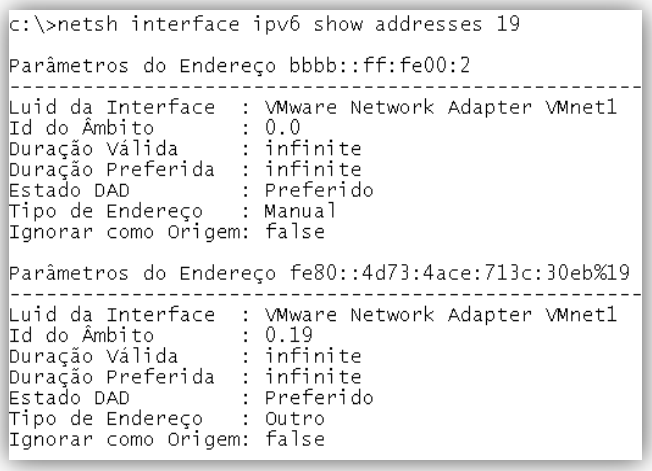


Figura 54 - Configuração do endereço IPv6 no Windows

Figura 55 - Teste de ligação entre Windows e máquina virtual

7.4 Exemplo Hello World

"Olá mundo", ou do inglês "*Hello World*", foi o primeiro exemplo a ser testado. Primeiro o código foi compilado para a plataforma usada (atmega128rfa1) e de seguida enviado para o dispositivo. O teste foi bem-sucedido.

Existem formas de acelerar o processo de compilação, esta pode ser usada tanto na consola *Cygwin*, como na consola do Linux. A compilação 1 permite guardar o tipo de plataforma que se está a utilizar, deixando de ser necessário indicar qual a plataforma, sempre que se executa o comando "*Make*", como na compilação 2. Ambas já definidas, previamente na secção 7.2.

<table>
<tr><th>Compilação 1</th><th>Compilação 2</th></tr>
<tr><td colspan="2">$ cd examples/hello-world/</td></tr>
<tr><td>$make savetarget TARGET=avr-atmega128rfa1
$make hello-world.avr-atmega128rfa1.hex</td><td>$ make TARGET=avr-atmega128rfa1 hello-world.hex</td></tr>
<tr><td colspan="2">Envio do aplicativo para o dispositivo</td></tr>
<tr><td colspan="2">avrdude -b 19200 -P COM1 -pm128rfa1 -c stk500v1 -U flash:w:hello-world.avr-atmega128rfa1.hex</td></tr>
</table>

7.5 Criação da primeira comunicação

Para entender a estrutura da comunicação, foi explorado o exemplo simple-rpl-udp e broadcast-example. O exemplo simple-rpl-udp, que se encontra na directoria *"Contiki/exemples/ipv6"*, permite enviar mensagens *unicast* através do protocolo de comunicação UDP, de um nó para outro sem qualquer tipo de resposta, independentemente se a mensagem é entregue ou não. Já no segundo exemplo

(broadcast, nome errado porque no IPv6 não existe broadcast), permite receber e enviar mensagens em multicast. O resultado bem-sucedido é apresentado no Anexo 1.

É importante visualizar também para o ficheiro Make da directoria, pois contém informação importante, apresentada de seguida, que permite activar o IPv6 na rede LoWPAN e o RPL. Todos os nós que usem o protocolo RPL devem ser compilados com a seguinte estrutura no ficheiro Make:

1. **WITH_UIP6=1** – Activa o IPv6;
2. **UIP_CONF_IPV6=1** – Activa o endereçamento IPv6;
3. **CFLAGS+= -DUIP_CONF_IPV6_RPL** – Activa a flag do encaminhamento RPL.

Deste modo, quando o encaminhamento é usado a *flag* deverá ser adicionada a cada nó. Para compilar este exemplo, executou-se os seguintes comandos:

Compilação

```
$ cd contiki/examples/ipv6/simple-udp-rpl
$ make unicast-sender.avr-atmega128rfa1.hex
broadcast-example.avr-atmega128rfa1.hex
```

7.5.1 Problemas

Por defeito o sistema operativo Contiki mantém as mensagens RPL ocultas, para as visualizar é necessário comentar o código "define DEBUG DEBUG_NONE" e adicionar "define DEBUG DEBUG_PRINT" nos seguintes ficheiros:

- **rpl.c** – Visualização da implementação do RPL;
- **rpl-dag.c** – Visualização da lógica da criação e manutenção da DAG;
- **rpl-ext-header.c** – Visualização da gestão das extensões do cabeçalho RPL;
- **rpl-of0.c** – Visualização da lógica das OF do RPL;
- **rpl-of-etx.c** – Visualização da implementação das métricas do RPL;
- **rpl-timers.c** – Visualização da gestão do tempo do RPL;
- **rpl-icmp6.c** – Visualização da entrada e saída das mensagens ICMP do RPL.

O segundo e maior problema foi atribuição dos endereços IP. Os endereços IP são atribuídos consoante o endereço MAC, isto causou problemas, já que todos os dispositivos continham o mesmo endereço MAC devido à configuração da EPROM. É possível alterar o endereço MAC, através do Contiki, activando a alteração aleatória de endereços MAC "CONTIKI_CONF_RANDOM_MAC" ou usar endereços próprios. Esta opção é permitida alterando-se directamente no ficheiro "*params.c*" localizado em "*platform/avr-atmega128rfa1/*". No entanto, notou-se que era extremamente difícil manter os nós nas mesmas redes devido à forma de atribuição do endereço IPv6.

Por fim, verificou-se que era possível também, atribuir um endereço IPv6 directamente no ficheiro "*params.h*" localizado em "*platform/avr-atmega128rfa1/*".

```
#if UIP_CONF_LL_802154
//#define PARAMS_EUI64ADDR {0x02, 0xNN, 0xNN, 0xNN, 0xNN, 0xNN, 0xNN, 0xNN}
#define PARAMS_EUI64ADDR {0x02, 0x00, 0x00, 0xff, 0xfe, 0x00, 0x00, 0x01}
#else
```

O problema deste modo de atribuição dos endereços é ter sistematicamente de alterar e compilar para cada nó um endereço diferente. Ou seja, por exemplo, se o endereço do coordenador for configurado com o valor NN=01, os nós devem ser configurados com uma incrementação desse valor, como mostrado a seguir:

```
#define PARAMS_EUI64ADDR {0x02, 0x00, 0x00, 0xff, 0xfe, 0x00, 0x00, 0xNN}
```

Outra forma de atribuir o endereço IP é usar atribuição directa em cada aplicação, por exemplo:

```
uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0x00ff, 0xfe00, 1);
```

No entanto, não foi possível usar esta configuração devido ao problema da EPROM.

7.6 RPL UDP

O exemplo representado na Figura 56, permitiu entender o envio e recepção de uma *string* via UDP entre servidor (que funciona como um coordenador) e clientes (os nós) com o uso do protocolo de encaminhamento RPL. É importante frisar que neste código é necessário configurar o endereço do servidor e o porto no código do cliente.

Servidor	`uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0x00ff, 0xfe00, 1);`
Cliente	`uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0, 0x00ff, 0xfe00, 1);`
Compilação	
`$ cd contiki/examples/ipv6/rpl-udp/` `$ make udp-server.avr-atmega128rfa1.hex udp-client.avr-atmega128rfa1.hex`	

Devido ao problema da EPROM a configuração seguinte apenas pôde ser testada no simulador *Cooja*, contudo, no Anexo 2, encontra-se exposta a troca de mensagens entre cliente e servidor com a configuração relativa ao ficheiro params.h, captadas no Windows através do *Term Terminal*. É necessário relembrar que, para cada nó, o ficheiro params.h tem de ter um novo endereço e tem de ser novamente compilado.

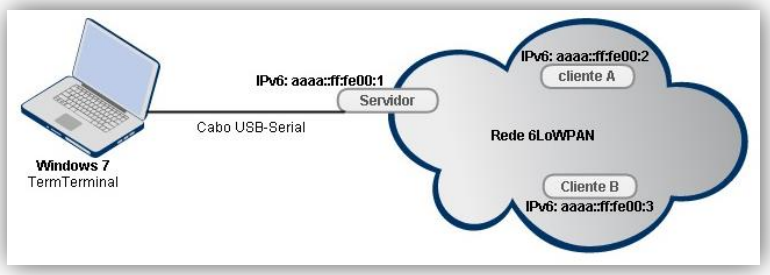


Figura 56 - Esquema de rede do exemplo RPL-UDP

Ao contrário do exemplo anterior no qual apenas existia dois dispositivos, neste exemplo foram usados três dispositivos, um servidor e dois clientes, sendo permitido observar que independentemente do número de nós que haja, se não existir um coordenador (servidor) na rede, o encaminhamento nunca é obtido e ambos os clientes entram num *loop* de mensagens DIS. O dispositivo com endereço "fe80::ff:fe00:3" envia uma mensagem DIS para o endereço "fe80::ff:fe00:2" e vice versa, como explicita a Figura 57.

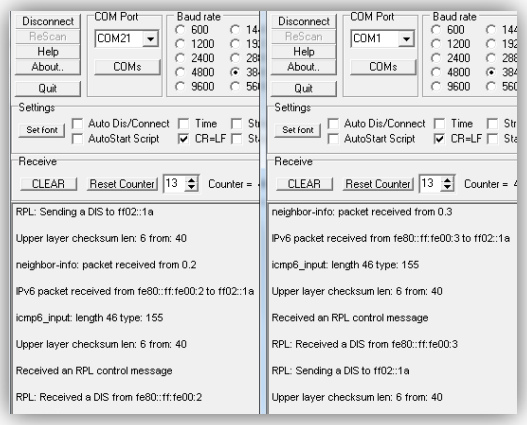


Figura 57 -Term terminal loop de mensagens entre dois nós

7.7 Border router

Como já referido na Secção 5.1, o RPL não usa DHCP ou *router advertisements* para definir o prefixo, ele envia o prefixo através das mensagens RPL. O *border router* faz do uso de um mecanismo chamado "*tunslip6*" e que se encontra em *"/tools"*. Permite encapsular os pacotes IPv6 e fazer a ponte entre a máquina virtual Ubuntu e o dispositivo *border router* via porta serial. Permite também o envio do prefixo pelo túnel para a rede 6LoWPAN.

Este exemplo foi bem-sucedido mas não permite envio de dados, porque o *border router* serve apenas para fazer a ponte entre as duas redes (rede 6LoWPAN e por exemplo a Internet). Para enviar os dados, usou-se o código cliente do exemplo anterior (RPL UDP), formando uma rede que se encontra representada na Figura 58.

Outro aspecto importante a referir é o facto deste exemplo conter uma aplicação simples de um servidor web, que pode ser adicionado para visualizar as rotas e endereços IPv6 que se encontram dentro da rede 6LoWPAN.

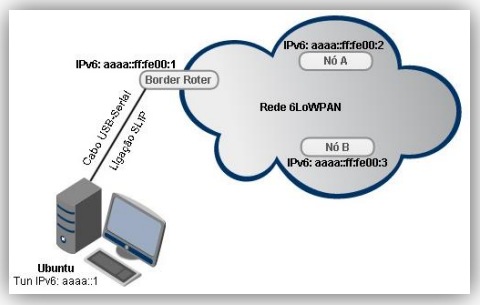


Figura 58 - Esquema de rede do exemplo border router

7.7.1 Compilação

Apesar das várias opções, foi usado para teste a compilação do projecto com SLIP e com servidor web, representado na Figura 59, para melhor visualização da rede 6LoWPAN, através do uso do *browser*:

Figura 59 - Dados da rede 6LoWPAN via HTTP

7.7.2 Criação do túnel SLIP

O *border router* envia pedidos a cada segundo ao tunslip6, o qual responde com o prefixo da rede associada, exemplo "*aaaa::*", de seguida o *border router* envia o prefixo nas mensagens DIO até ao fundo da DAG. É desta forma que o *border router* consegue comunicar para dentro da rede 6LoWPAN e vice-versa. Ao executar o comando a seguir apresentado na Figura 60, permitiu-se criar uma interface de nome "tun0" e com o endereço local *aaaa::1*.

Todo o processo e testes podem ser analisados no Anexo 3.

```
user@instant-contiki:~/contiki/tools$ sudo ./tunslip6 aaaa::1/64 -s /dev/ttyUSB0 -B 38400 -v6
********SLIP started on ``/dev/ttyUSB0''
opened tun device ``/dev/tun0''
ifconfig tun0 inet `hostname` up
ifconfig tun0 add aaaa::1/64
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:127.0.1.1  P-t-P:127.0.1.1  Mask:255.255.255.255
          inet6 addr: fe80::1/64 Scope:Link
          inet6 addr: aaaa::1/64 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

Packet from TUN of length 76 - write SLIP
0000 60 00 00 00 00 24 00 01 fe 80 00 00 00 00 00 00 00 00 00 00 00 00 00 01 ff 02 00 00 00 00
00 00 00 00 00 00 00 00 00 00 16 3a 00 05 02 00 00 01 00 8f 00 70 08 00 00 00 01 04 00 00 00 ff 02
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02
```

Figura 60 - Criação do túnel

7.7.3 Problemas

Por defeito, ao criar o túnel SLIP, é criada automaticamente uma rota que permite o envio dos pacotes com o prefixo de rede "*aaaa::*" através do tun0. Caso não a tenha criado, poderá ter problemas de encaminhamento. Para resolver, é necessário executar o seguinte comando na consola da máquina virtual Ubuntu:

```
sudo route -A inet6 add aaaa::/64 dev tun0
```

7.8 Sumário

Apesar dos muitos exemplos disponibilizados na pasta do Contiki, apenas os exemplos "simple-udp-rpl", "rpl-udp" e "rpl-border-router", foram utilizados neste trabalho, devido ao seu código e objectivo. Permitem o envio de mensagens através do protocolo de comunicação UDP, endereçamento IPv6, protocolo de encaminhamento RPL e ligação entre redes IPv6 e redes 6LoWPAN.

Todos os exemplos usados passaram por testes de configuração, de um coordenador e nós (configuração dos endereços IPv6, LED e sensor de luminosidade) e visualização da interacção entre ambos (relativo à troca de mensagens RPL e UDP, distância de comunicação entre dispositivos e alteração entre as posições dos nós).

Apesar dos testes terem sido bem-sucedidos, não se conseguiu resolver o problema do coordenador não actualizar a sua lista de encaminhamento aquando da remoção de um nó na rede 6LoWPAN.

Após a utilização dos exemplos práticos relativos ao protocolo de encaminhamento RPL e endereçamento IPv6, tornou-se mais evidente o encaminhamento das mensagens e a dimensão de tráfico usado. Com o uso do debug conclui-se que existe muito processamento de mensagens por ambas as partes, tanto do coordenador como pelos nós, numa rede de pequena dimensão. Para tirar partido do sistema, é ideal que os dispositivos estejam separados por uma distância razoável, não superior a metade do raio de acção da antena, permitindo um aumento de performance e baixo consumo de energia.

8. Projecto Final

"Um homem nunca sabe aquilo de que é capaz até que o tenta fazer."

Charles Dickens

8.1 Lista de requisitos

Para além dos requisitos descritos e avaliados no Anexo 4, seguem-se os mais importantes:

- Tem de haver uma comunicação directa entre dois dispositivos com endereços IPv6;
- O acesso aos nós deve ser feito através de qualquer computador na Internet, sabendo apenas o IP e porto do nó;
- Um nó tem de ter a capacidade de captar dados, por exemplo, luminosidade;
- Qualquer utilizador tem que ter a possibilidade de aceder, actualizar e modificar dados dos nós, por exemplo, ligar/desligar LED ou actualizar o valor da luminosidade.

8.1.1 Problemas

Muitas das soluções existentes passam pela inserção de um intermediário (um servidor) que armazena a informação numa base de dados, sendo depois essa acedida via aplicação web. Tal como a solução anterior, outra solução muito utilizada passa por guardar toda a informação no coordenador e aceder a essa informação via HTTP.

Estas soluções foram descartadas uma vez que não preenchem o requisito de comunicação directa entre dois dispositivos com endereços IPv6.

8.1.2 Solução

Depois de analisados os requisitos e problemas, chegou-se à conclusão que a melhor solução seria a construção de uma aplicação Java capaz de satisfazer os requisitos descritos no Anexo 4.

O exemplo *border router* foi escolhido para fazer a ponte entre as duas redes, no entanto, o código necessitou de algumas modificações apresentadas de seguida:

- No exemplo *border router* existia a necessidade de ter um botão para iniciar o processo de reparação da DAG. Para colmatar este problema, uma vez que os dispositivos não devem ter necessidade de intervenção humana, o código foi substituído pela criação de um evento como mostra o seguinte código:

```
 static struct etimer time_to_repair;

etimer_set(&time_to_repair, CLOCK_CONF_SECOND*CONF_SECOND);


  while(1)
{
    PROCESS_YIELD();
    if (ev == PROCESS_EVENT_TIMER)
    {
      PRINTF("----- Initiating global repair -----\n");
      rpl_repair_root(RPL_DEFAULT_INSTANCE);
      rpl_repair_root(RPL_DEFAULT_LIFETIME);
    }


    etimer_reset(&time_to_repair);
  }
```

- O exemplo não permite receber dados via UDP. Criou-se um novo processo associado ao existente capaz de receber dados via UDP. Similar ao exemplo de servidor RPL UDP, este necessitou de ser adicionado aos processos existentes no *border router* entre outras alterações apresentadas de seguida:

Processo adicionado ao border router

```
PROCESS_NAME(udp_server_process);
PROCESS(border_router_process, "Border router process");

AUTOSTART_PROCESSES(&border_router_process,&udp_server_process);
```

Processo de envio dos endereços IPv6 de todos os nós, no processo servidor UDP

```
for(rl = uip_ds6_route_list_head(); rl != NULL; rl = list_item_next(rl))
{
  sprintf(buf,
"%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x",
      ((uint8_t *)&rl->ipaddr)[0],
      ((uint8_t *)&rl->ipaddr)[1], ((uint8_t *)&rl->ipaddr)[2],
      ((uint8_t *)&rl->ipaddr)[3], ((uint8_t *)&rl->ipaddr)[4],
      ((uint8_t *)&rl->ipaddr)[5], ((uint8_t *)&rl->ipaddr)[6],
      ((uint8_t *)&rl->ipaddr)[7], ((uint8_t *)&rl->ipaddr)[8],
      ((uint8_t *)&rl->ipaddr)[9], ((uint8_t *)&rl->ipaddr)[10],
      ((uint8_t *)&rl->ipaddr)[11], ((uint8_t *)&rl->ipaddr)[12],
      ((uint8_t *)&rl->ipaddr)[13], ((uint8_t *)&rl->ipaddr)[14],
      ((uint8_t *)&rl->ipaddr)[15]
  );
  // Copy IPv6 address and port from the list to send
  uip_ipaddr_copy(&server_conn->ripaddr, &mc->client_IP);
  server_conn->rport = UIP_HTONS(mc->client_port);
  // To send
  uip_udp_packet_send(server_conn, buf, sizeof(buf));
  // Put connection and port at zero to be able to receive from other's
  memset(&server_conn->ripaddr, 0, sizeof(server_conn->ripaddr));
  server_conn->rport = 0;
}
```

O simples servidor web do *border router* tornou-se importante para confirmação dos dados sobre a rede 6LoWPAN, uma vez que as mensagens não poderiam ser recebidas em simultâneo no *Term Terminal.*

Tanto para o processo servidor UDP como para os nós, foi necessário criar uma estrutura para existir a capacidade de guardar uma lista com a informação do endereço IPv6 e porto da aplicação Java, apresentado a seguir:

```
struct manage_connections
{
    // this saves the java connections
    struct manage_connections *next;
    uip_ip6addr_t client_IP;
    uint16_t client_port;
};
MEMB(connection_memb, struct manage_connections, MAX_JAVA_CONNECTION);
```

Adicionar da lista de ligações	

Servidor UDP	mc->client_IP = UIP_IP_BUF->srcipaddr;
	mc->client_port = UIP_HTONS(UIP_IP_BUF->srcport);
	list_add(connection_list, mc);

Nós	mc->client_IP = UIP_IP_BUF ->srcipaddr;
	mc->client_port = UIP_HTONS(UIP_IP_BUF->srcport);
	list_add(connection_list, mc);

Remover da lista de ligações	

Servidor UDP	mc->client_IP = UIP_IP_BUF->srcipaddr;
	mc->client_port = UIP_HTONS(UIP_IP_BUF->srcport);
	memb_free(&connection_memb, mc);
	list_remove((connection_list, mc);

Nós	mc->client_IP = UIP_IP_BUF ->srcipaddr;
	mc->client_port = UIP_HTONS(UIP_IP_BUF->srcport);
	memb_free(&connection_memb, mc);
	list_remove((connection_list, mc);

No exemplo RPL UDP o porto é fixo por forma a reduzir o tamanho dos dados que circulam na rede, sendo usada a estrutura "uip_ip_hdr" localizada em *"core/net/uip.h",* permitindo reduzir o tamanho das mensagens, sendo que o tamanho dos cabeçalhos da camada 3 (camada de rede) não é fixo.

Todavia, para este projecto as ligações poderão usar portos diferentes, neste caso foi necessário usar a estrutura "uip_udpip_hdr" localizada no mesmo ficheiro. As suas diferenças podem ser analisadas na Tabela 11.

RPL UDP	Projecto

`struct uip_ip_hdr`
`{`
`/* IPV6 header */`
`uint8_t vtc;`
`uint8_t tcflow;`
`uint16_t flow;`
`uint8_t len[2];`
`uint8_t proto, ttl;`
`uip_ip6addr_t srcipaddr, destipaddr;`
`};`
	`struct uip_udpip_hdr`
	`{`
	`/* IPv6 header. */`
	`uint8_t vtc,`
	`tcf;`
	`uint16_t flow;`
	`uint8_t len[2];`
	`uint8_t proto, ttl;`
	`uip_ip6addr_t srcipaddr, destipaddr;`
	`/* UDP header. */`
	`uint16_t srcport, destport;`
	`uint16_t udplen;`
	`uint16_t udpchksum;`
	`}`

`#define UIP_IP_BUF`
`(`
`(struct uip_ip_hdr *)`
`&uip_buf[UIP_LLH_LEN]`
`)`
	`#define UIP_IP_BUF`
	`(`
	`(struct uip_udpip_hdr*)`
	`&uip_buf[UIP_LLH_LEN]`
	`)`

Tabela 11 - Diferenças entre uip_ip_hdr e uip_udpip_hdr

Para os nós foram necessárias as seguintes configurações apresentadas:

Gestão do estado do led

```
static int
adc_led(int state)
{
  if(state==1){
    //LED = 0
    DDRE = _BV(PE2);
    PORTE ^= _BV(PE2);
    printf(" \n LED OFF");
    return 0;
  }else {
    //LED = 1
       DDRE = _BV(PE2);
       PORTE ^= _BV(PE2);
       printf(" \n LED on");
       return 1;
  }
}
```

Receber o valor da luminosidade

```
static int
adc_light(void){
  ADMUX = 0xC0, ADCSRB &= (0 << MUX5), ADMUX |= _BV(REFS1) | _BV(REFS0);
  ADCSRA |= (1<<ADEN), ADCSRA |= (1<<ADSC);

  while ( ADCSRA & (1<<ADSC) ) {;}

  lightness = ADC;
  printf(" \n adc_light -> ADC Value: %d \n", lightness);
  return lightness;
}
```

Processo de verificação do estado da luminosidade

```
PROCESS_THREAD(timetochecknodes_process, ev, data)
{
  static struct etimer timer;
  PROCESS_BEGIN();
  PRINTF("START Time to check nodes\n");
  //generate event for 10 sec
  etimer_set(&timer, CLOCK_CONF_SECOND * CONF_CLOCK);
  while (1)
 {
    PROCESS_WAIT_EVENT();
    if (ev == PROCESS_EVENT_TIMER)
    {
      //check if luminosity was change
      check_luminosity();
    }

    etimer_reset(&timer);
  }
PROCESS_END();
}
```

No final obteve-se a rede representada na Figura 61. No Windows 7 é executada a aplicação Java, os pedidos provenientes da aplicação são encaminhados até ao Ubuntu que por sua vez encaminha para o *border router*, este verifica se o pedido é para ele ou encaminha para o nó específico.

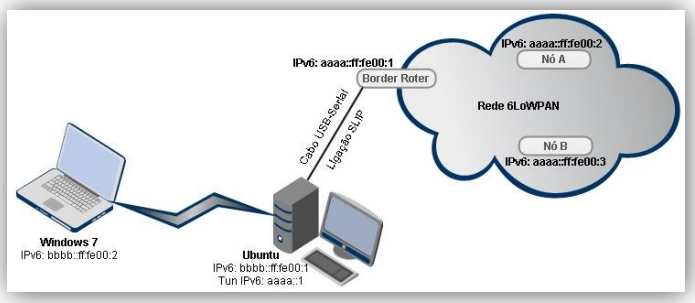


Figura 61 - Diagrama de rede do projecto

8.2 Sistema de comunicação de mensagens

Na conexão inicial, representada na Figura 62, o aplicativo liga-se directamente ao *border router* (coordenador). Por sua vez, o *border router* enviará a informação do número de nós existentes na rede e seu porto, guardando os dados da ligação do aplicativo Java (IPv6 e porto) numa tabela, para futuras actualizações de informação dos nós existentes na rede 6LoWPAN. Com a informação dos nós, enviada pelo *border router* para o aplicativo Java, a qualquer momento a informação (estado do LED ou valor da luminosidade) de um nó pode ser acedida no aplicativo. Este liga-se directamente ao dispositivo e faz o pedido do dado pretendido, como se pode verificar na Figura 63.

Para poder alterar o estado do LED (on/off), é sempre necessário iniciar o processo de registo do aplicativo ao nó pretendido. De seguida o nó regista o IP e o porto do aplicativo, enviando-lhe de volta a confirmação de registo juntamente com a informação relativa ao estado do LED e o valor da luminosidade, como se pode verificar na Figura 64. Para receber automaticamente alterações da informação dos nós, tal como no coordenador, cada nó irá guardar uma lista de ligações (IPv6 e porto) de cada aplicativo registado no nó.

No final, para que a ligação do aplicativo seja removida tanto no coordenador como em cada nó, em que este esteja registado, o aplicativo tem de enviar um pedido de cancelamento de registo e receber dos mesmos a confirmação.

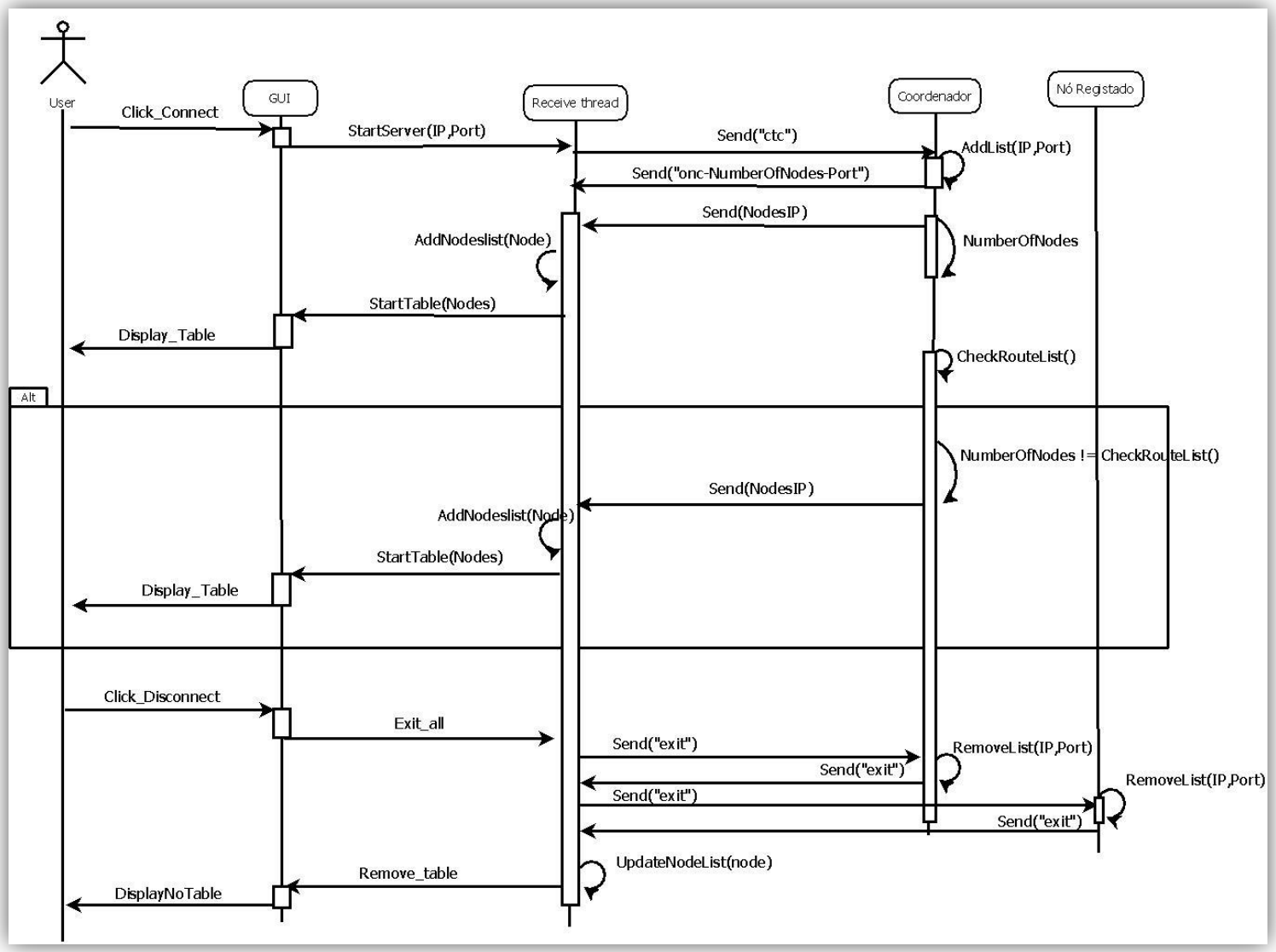


Figura 62- Diagrama de sequência da aplicação Java - coordenador

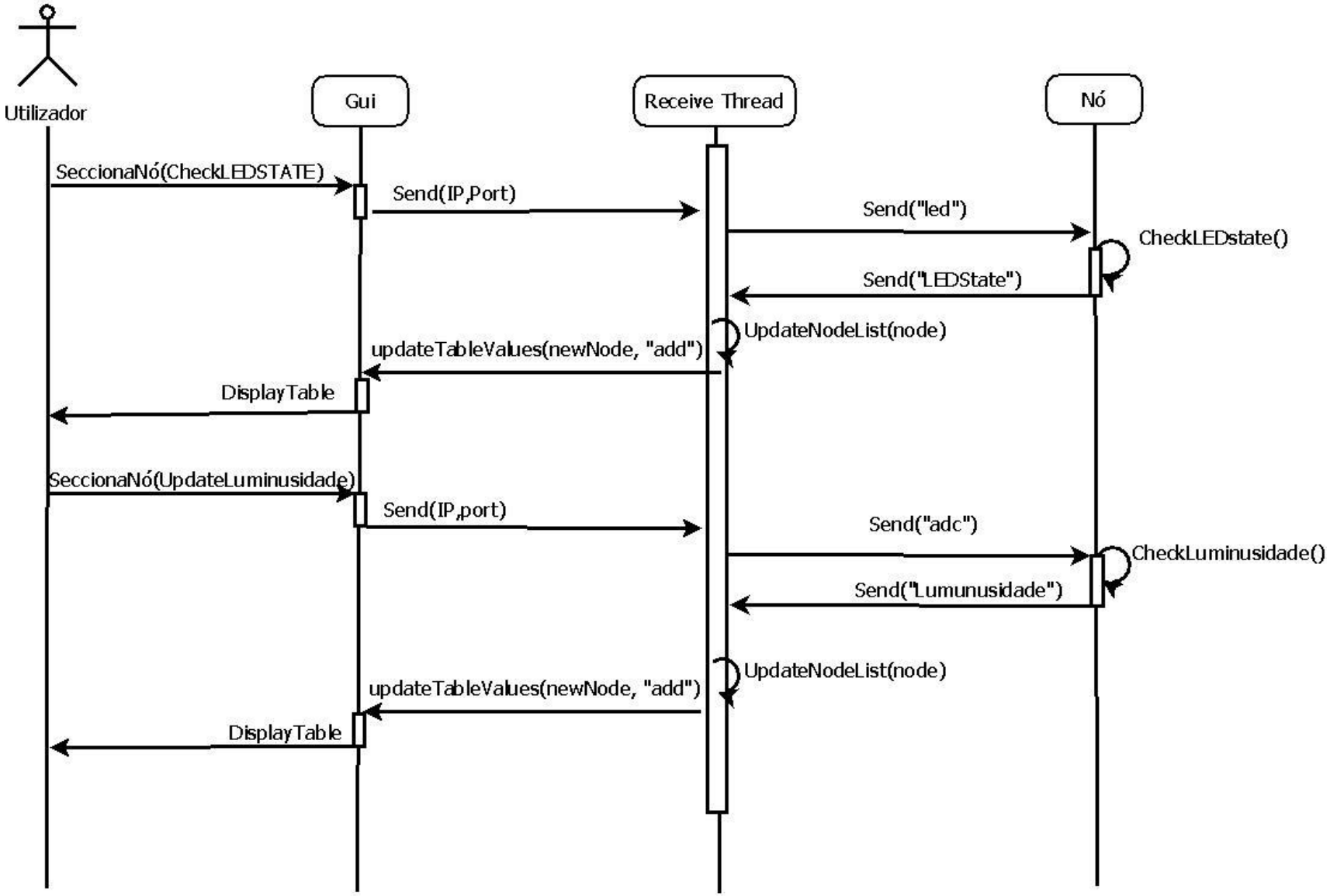


Figura 63 - Diagrama de sequência aplicação Java - nó sem registo

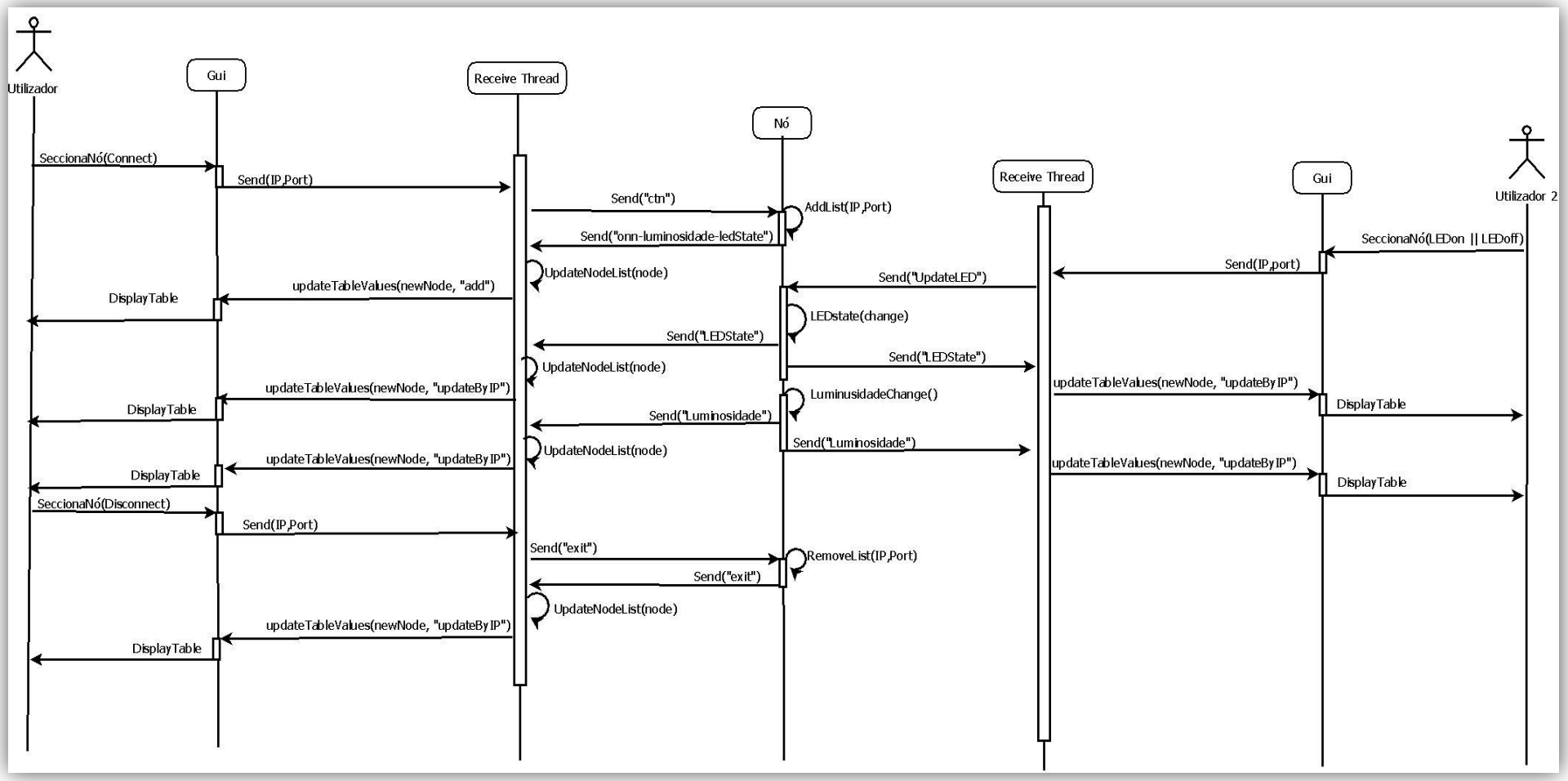


Figura 64 - Diagrama de sequência aplicação Java - nó com registo

8.3 Testes de mensagens

Antes do desenvolvimento da aplicação Java, efectuou-se testes usando o comando *netcat* do Linux, cujo exemplo do resultado é apresentado na Figura 65. Este permitiu, de forma simplista, uma avaliação em termos de envio e recepção das mensagens entre uma aplicação e um nó. O processo de comunicação descrito no diagrama da Figura 64, encontra-se comprovado na Figura 65, sendo um exemplo de sucesso de uma simulação de troca de mensagens entre uma aplicação e um nó.

Figura 65 - Ligação e comunicação da aplicação ao nó

8.4 Desenvolvimento do aplicativo

O comando *netcat* e aplicação *Wireshark* foram essenciais na exclusão dos requisitos, bem como nos testes de mensagens entre aplicativo, coordenador e nós da rede. Depois de todos os testes feitos, como se pode verificar no Anexo 5, iniciou-se o processo de desenvolvimento da aplicação Java.

A aplicação permite fácil acesso à informação e visualização dos dados de cada nó como exposto na Figura 66. O cliente apenas necessita de inserir o endereço IPv6 e porto do coordenador (*border router*) na aplicação Java, caso existam nós na rede 6LoWPAN, estes serão disponibilizados no campo "*Nodes Information*". A qualquer momento o cliente poderá actualizar a informação de todos os nós, carregando no botão "*Refresh All Nodes*" ou esperar que os nós envie nova informação.

Com os nós apresentados na Figura 67, todas as opções serão disponibilizadas seleccionando o nó pretendido e carregando com o botão direito do rato.

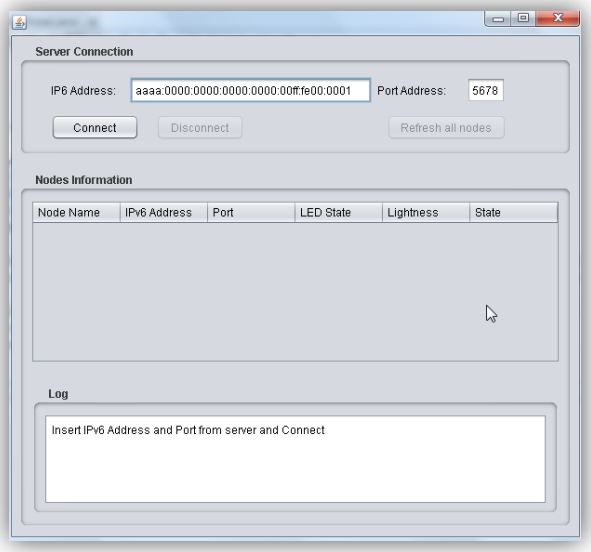


Figura 66 - Cliente na aplicação

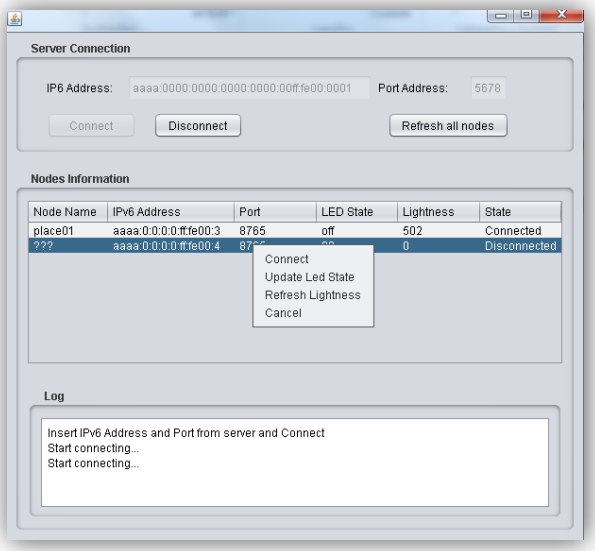


Figura 67 – Cliente na aplicação Java com nós

8.5 Problemas

O coordenador não actualiza a sua lista de encaminhamento aquando da remoção de um nó da rede 6LoWPAN. A solução encontra-se nas variáveis, *RPL_DEFAULT_LIFETIME_UNIT* e *RPL_DEFAULT_LIFETIME*, que são usadas na raiz do rpl (no coordenador), para definirem o tempo de vida das rotas determinadas pelo RPL. Quando o tempo expira a rota deveria ser apagada pelo *rpl_purge_routes*, localizado no ficheiro "*core/net/rpl/rpl.c*".

Outra solução foi a utilização da resolução do problema "*No-Path DAOs are not propagated*" [40], no entanto, também sem sucesso.

Na escolha do processo de verificação da luminosidade, verificou-se que o valor lido pelo sensor dependia da resistência associada. Foi necessário através de um potenciómetro, encontrar o melhor valor de resistência para que o valor captado não se alterasse a qualquer estado de variação.

Aquando o envio das mensagens, é necessário verificar o tamanho uma vez que, se forem de tamanho elevado, pode provocar o reiniciar do equipamento.

8.6 Sumário

As soluções possíveis para este tipo de trabalho, passam pela existência de um coordenador ou servidor que recebe toda a informação e disponibiliza-a. Estas soluções não permitem que haja uma ligação directa entre cliente e nó. Este trabalho permitiu desenvolver uma aplicação em Java que comunica directamente com os dispositivos na rede 6LoWPAN.

A aplicação permite informar o utilizador sobre o número de nós existentes na rede 6LoWPAN, o valor da luminosidade e estado do LED de cada nó. Para que haja inter-actividade entre a aplicação e os nós, existe a possibilidade de ligar ou desligar o LED e tanto o nó como o coordenador, enviam periodicamente informação para a aplicação. O coordenador envia as alterações efectuadas na rede 6LoWPAN e cada nó envia a alteração no valor da luminosidade e/ou estado do LED.

Devido ao problema da actualização de rotas de encaminhamento, não foi possível testar os nós em movimento, apesar disso foi possível verificar as alterações de *Rank* mediante aproximação do nó ao coordenador e encaminhamento através de um nó para o coordenador, sendo visível no Anexo 2.

Conclusão e trabalho futuro

"Apesar do longo percurso decorrido, quando se atinge o objectivo primordial, sentimo-nos realizados, e com grande valimento"

Sérgio Ferreira

Conclusão

Anteriormente as redes pessoais sem fio (WPAN) eram vistas como o fim da ligação da Internet, o 6LoWPAN veio estender a ligação, sendo de momento o LoWPAN o fim da ligação à Internet.

Torna-se evidente que o coordenador é o ponto fraco da rede. Se não existir, a rede não comunica entre si. Apesar dos requisitos parecerem fáceis, tornou-se difícil, devido ao tamanho das mensagens enviadas na rede 6LoWPAN, o processamento da informação e controlo dos processos, que tornam o sistema pesado e por consequência a reinicialização do mesmo.

Apesar de ser uma rede experimental, na opinião do autor, o projecto facilmente pode ser adaptado a qualquer tipo aplicabilidade. Devido à limitação dos dispositivos para testes a rede foi limitada a 3 nós estáticos, levando a uma certa dificuldade na realização do problema da actualização das rotas de encaminhamento. Obstante desta situação, todos os outros objectivos foram conseguidos com sucesso.

Trabalho futuro

Como em todas as trocas de informação, apesar da segurança não ter sido abordada neste projecto, seria importante conter um mecanismo de chaves de segurança que permitisse a confidencialidade ou explorar os mecanismos de segurança do protocolo IPv6.

Por último, seria importante fazer uma comparação entre a performance obtida efectuada com o projecto e as soluções já existentes, tais como: o encaminhamento de toda a informação para o coordenador, ou através de uma aplicação de um servidor HTTP em cada nó disponível na rede.

Bibliografia

Bibliografia

[1] M. Kuorilehto, J. Suhonen, . P. Hämäläinen, M. Hännikäainen and T. D. Hämäläinen, Ultra-Low Energy Wireless Sensor Networks in Practice, Tampere University of Technology, Finland: John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, 2007.

[2] K. Sohraby, D. Minoli e T. Znati, Wireless Sensor Networks Technology, Protocols, and Applications, John Wiley & Sons, Inc., Hoboken, New Jersey.

[3] F. L. Lewis, "Wireless Sensor Networks," D.J. Cook and S.K. Das, John Wiley, New York, The University of Texas at Arlington, 2004.

[4] L. C. W. S. T. W. a. H. Y. Zhihong Qian, Recent Advances in Computer Science and Information Engineering: Volume 4, Springer, 2012.

[5] The Internet Assigned Numbers Authority (IANA), "IPv6 – Successor to IPv4 Confronting Transition," [Online]. Available: http://www.iana.org/about/presentations/20110707-vegoda-ipv6.pdf.

[6] ICANN, "Available Pool of Unallocated IPv4 Internet, Addresses Now Completely Emptied , The Future Rests with IPv6".

[7] P. Loshin, IPv6: Theory, Protocol, and Practice SECOND EDITION, ELSEIER, 2003.

[8] S. D. R. Hinden, "IP Version 6 Addressing Architecture," 02 2006 2006. [Online]. Available: http://tools.ietf.org/html/rfc4291.

[9] Internet Engineering Task Force (IETF), *IPv6 Addressing of IPv4/IPv6 Translators,* 2010.

[10] B. C. C. Huitema, "Deprecating Site Local Addresses," 09 2004. [Online]. Available: http://tools.ietf.org/html/rfc3879.

[11] R. H. S. Deering, "Internet Protocol, Version 6 (IPv6) Specification," 12 1998. [Online]. Available: http://tools.ietf.org/html/rfc2460.

[12] J. Davies, Understand IPv6, Microsoft Press, 2003.

[13] CISCO, "IPv6 Extension Headers Review and Considerations," 10 2006. [Online]. Available: http://www.cisco.com/en/US/technologies/tk648/tk872/technologies_white_paper0900aecd8054d37d.html.

[14] S. D. M. G. E. A. Conta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," 03 2006. [Online]. Available: http://tools.ietf.org/html/rfc4443.

[15] T. I. A. N. A. (IANA), "Internet Control Message Protocol (ICMP) Parameters," 25 02 2013. [Online]. Available: http://www.iana.org/assignments/icmp-parameters/icmp-parameters.xml.

[16] T. N. S. Thomson, "IPv6 Stateless Address Autoconfiguration," 12 1998. [Online]. Available: http://tools.ietf.org/html/rfc2462.

[17] E. N. W. S. T. Narten, "Neighbor Discovery for IP Version 6 (IPv6)," 12 1998. [Online]. Available: http://tools.ietf.org/html/rfc2461.

[18] "RIPng for IPv6," 01 1997. [Online]. Available: http://tools.ietf.org/html/rfc2080.

[19] D. F. J. M. R. Coltun, "OSPF for IPv6," 12 1999. [Online]. Available: http://tools.ietf.org/html/rfc2740.

[20] R. C. D. K. Y. R. T. Bates, "Multiprotocol Extensions for BGP-4," 2007. [Online]. Available: http://tools.ietf.org/html/rfc4760.

[21] Z. S. e. C. Bormann, 6LoWPAN : The wireless Embedded Internet, John Wiley & Sons Ltd, 2009.

[22] Internet Protocol for Smart Objects (IPSO) Alliance, "6LoWPAN: Incorporating IEEE 802.15.4 into the IP architecture," 01 2009. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.169.150.

[23] I. E. T. F. (IETF), "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," 09 2011. [Online]. Available: http://tools.ietf.org/html/rfc6282.

[24] N. K. J. H. D. C. G. Montenegro, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," 08 2007. [Online]. Available: http://tools.ietf.org/html/rfc4944.

[25] D. Y. Wang Huiqin, "An Improved Header Compression Scheme for 6LoWPAN Networks," em *2010 Ninth International Conference on Grid and Cloud Computing*, School of Computer Science and Engineering, Southeast University, 2010.

[26] C. K. N. N. K. N. a. B. M. A. Gee Keng Ee*, "A Review of 6LoWPAN Routing Protocols," Universiti Putra Malaysia, UPM Serdang, 43400 Selangor, Malaysia..

[27] Alliance, Internet Protocol for Smart Objects (IPSO), "RPL: The IP routing protocol designed for low power and lossy networks," 04 2011. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.232.7450.

[28] T. Clausen, U. Herberg e M. Philipp, "A Critical Evaluation of the IPv6 Routing Protocol for Low Power and Lossy Networks (RPL)".

[29] I. E. T. F. (IETF), "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," 03 2012. [Online]. Available: http://tools.ietf.org/html/rfc6550.

[30] I. E. T. F. (IETF), "Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks," 03 2012. [Online]. Available: http://tools.ietf.org/html/rfc6551.

[31] I. Submission, “Performance Evaluation of the Routing Protocol for Low-Power and Lossy Networks (RPL),” 10 2012. [Online]. Available: http://tools.ietf.org/html/rfc6687.

[32] I. E. T. F. (IETF), “ The Trickle Algorithm,” 03 2011. [Online]. Available: https://tools.ietf.org/html/rfc6206.

[33] sdawans, “Contiki wiki,” 23 03 2013. [Online]. Available: https://github.com/contiki-os/contiki/wiki.

[34] A. Dunkels, “Protothreads – Simplifying Programming of Memory - Constrained Embedded Systems,” 2006. [Online].

[35] A. Dunkels, “RFC-compliance,” 03 03 2003. [Online]. Available: http://static.usenix.org/event/mobisys03/tech/full_papers/dunkels/dunkels_html/node5.html.

[36] doxygen, “Contiki 2.6 - The Contiki Operating System,” 24 07 2012. [Online]. Available: http://contiki.sourceforge.net/docs/2.6/index.html.

[37] W. Dargie e C. Poellabauer, Fundamentals of Wireless Sensor Networks Theory and Practice, WILEY, 2010.

[38] A. Sehgal e J. Schönwälder, “Embedded Operating Systems,” 30 03 2011. [Online]. Available: http://cnds.eecs.jacobs-university.de/courses/os-2011/Contiki-2.pdf.

[39] “http://www.tcpipguide.com/free/t_SerialLineInternetProtocolSLIP.htm,” [Online].

[40] S. D. R. Hinden, “Internet Protocol Version 6 (IPv6) Addressing Architecture,” 04 2003. [Online]. Available: http://tools.ietf.org/html/rfc3513.

[41] F. D. P. Marques, “Use of BGP-4 Multiprotocol Extensions for IPv6 Inter-Domain Routing,” 03 1999. [Online]. Available:

https://tools.ietf.org/html/rfc2545.

[42] sdawans, "No-Path DAOs are not propagated," [Online]. Available: https://github.com/contiki-os/contiki/issues/114.

Anexos

Anexo 1

Criação da primeira comunicação (Simple-UDP-RPL)

Início da configuração dos dispositivos		
	UDP-Sender	**UDP-Receiver**
Boot	*******Booting Contiki 2.6******* EUI-64 MAC: 2-0-0-ff-fe-0-0-2	*******Booting Contiki 2.6******* EUI-64 MAC: 2-0-0-ff-fe-0-0-1
Permissão das mensagens RPL e do encaminhamento	RPL Enabled Routing Enabled	RPL Enabled Routing Enabled
Atribuição de endereços IPv6	IPv6 addresses: aaaa::ff:fe00:2 fe80::ff:fe00:2 Online	IPv6 addresses: aaaa::ff:fe00:1 fe80::ff:fe00:1 Online
Demostração de endereços IPv6	Addresses [4 max] aaaa::ff:fe00:2 fe80::ff:fe00:2	Addresses [4 max] aaaa::ff:fe00:1 fe80::ff:fe00:1
Lista de nós vizinhos	Neighbors [20 max] <none>	Neighbors [20 max] <none>
Lista de rotas de encaminhamento	Routes [20 max] <none>	Routes [20 max <none>

Troca de mensagens RPL entre os dispositivos		
Criação da DAG		RPL: My path ETX to the root is 0.0 RPL: Node set to be a DAG root with DAG ID aaaa::ff:fe00:1 RPL: Prefix set - will announce this in DIOs RPL: My path ETX to the root is 0.0
Envio da mensagem DIO		RPL: Sending prefix info in DIO for aaaa:: RPL: Sending a multicast-DIO with rank 256
Envio da mensagem DIS	RPL: Sending a DIS to ff02::1a	
Recepção da mensagem DIS		neighbor-info: packet received from 0.2 IPv6 packet received from fe80::ff:fe00:2 to ff02::1a icmp6_input: length 46 type: 155 Received an RPL control message RPL: Received a DIS from fe80::ff:fe00:2 RPL: Sending prefix info in DIO for aaaa:: RPL: Sending a multicast-DIO with rank 256

Recepção da mensagem DIO	Received an RPL control message RPL: Received a DIO from fe80::ff:fe00:1 Adding neighbor with ip addr fe80::ff:fe00:1link addr 02:00:00:ff:fe:00:00:01state 1 RPL: Neighbor added to neighbor cache fe80::ff:fe00:1, 02:00:00:ff:fe:00:00:01 RPL: Incoming DIO (id, ver, rank) = (30,240,256) RPL: Incoming DIO (dag_id, pref) = (aaaa::ff:fe00:1, 0)	
Ligação à DAG e atribuição do nó preferencial (Pai)	RPL: New instance detected: Joining... RPL: Adding fe80::ff:fe00:1 as a parent: succeeded RPL: My path ETX to the root is 5.0 RPL: Joined DAG with instance ID 30, rank 1536, DAG ID aaaa::ff:fe00:1 RPL: Adding default route through fe80::ff:fe00:1 uip_ds6_defrt_add: adding default route to fe80::ff:fe00:1	
Envio da nova mensagem DIO	RPL: Sending prefix info in DIO for aaaa:: RPL: Sending a multicast-DIO with rank 1536	
Recepção da mensagem DIO		Received an RPL control message RPL: Received a DIO from fe80::ff:fe00:2

Recálculo do Rank		Adding neighbor with ip addr fe80::ff:fe00:2link addr 02:00:00:ff:fe:00:00:02state 1 RPL: Neighbor added to neighbor cache fe80::ff:fe00:2, 02:00:00:ff:fe:00:00:02 RPL: Incoming DIO (id, ver, rank) = (30,240,1536) RPL: Incoming DIO (dag_id, pref) = (aaaa::ff:fe00:1, 0) RPL: My path ETX to the root is 0.0
Envio da mensagem DIO com nova configuração		RPL: Sending prefix info in DIO for aaaa:: RPL: Sending a multicast-DIO with rank 256
Envio da mensagem DAO	RPL: Sending DAO with prefix aaaa::ff:fe00:2 to fe80::ff:fe00:1	
Recepção da mensagem DAO		Received an RPL control message RPL: Received a DAO from fe80::ff:fe00:2 RPL: DAO lifetime: 255, prefix length: 128 prefix: aaaa::ff:fe00:2
Configuração do encaminhamento		uip-ds6-route: Looking up route for aaaa::ff:fe00:2 uip-ds6-route: No route found uip_ds6_route_add num 1 uip_ds6_route_add: adding route: aaaa::ff:fe00:2 via fe80::ff:fe00:2 RPL: Added a route to aaaa::ff:fe00:2/128 via fe80::ff:fe00:2

Resultado depois da conclusão da troca de mensagens RPL		
Lista de nós vizinhos	Neighbors [20 max] fe80::ff:fe00:1	Neighbors [20 max] fe80::ff:fe00:2
Lista de rotas de encaminhamento	Routes [20 max] <none>	Routes [20 max] aaaa::ff:fe00:2/128 (via fe80::ff:fe00:2) 16711425s
Envio da mensagem	Sending unicast to aaaa::ff:fe00:1	
Mensagem recebida		Data received from aaaa::ff:fe00:2 on port 1234 from port 1234 with length 10: 'Message 0'

Anexo 2

Exemplo RPL UDP

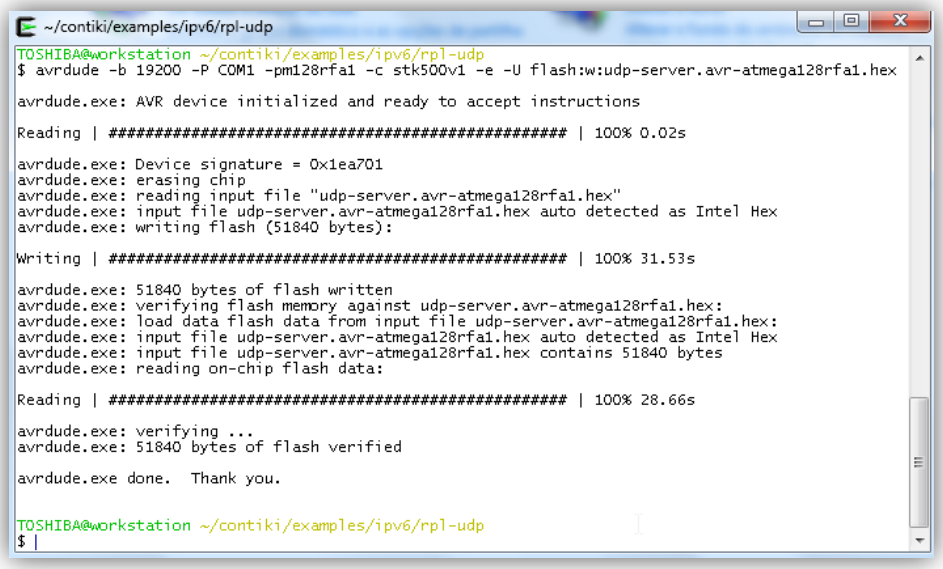


Figura 68 - Envio do executável para o dispositivo coordenador

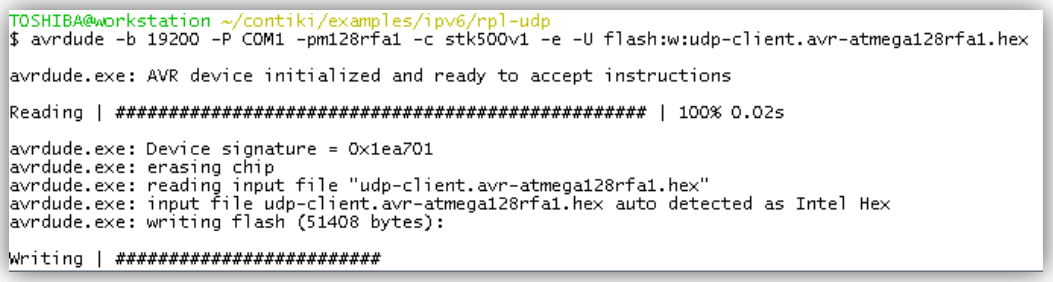


Figura 69 - Envio do executável para o dispositivo nó

Início da configuração dos dispositivos			
	UDP Servidor	**UDP Cliente 1**	**UDP Cliente 2**
Boot	*******Booting Contiki 2.6******* EUI-64 MAC: 2-0-0-ff-fe-0-0-1	*******Booting Contiki 2.6******* EUI-64 MAC: 2-0-0-ff-fe-0-0-2	*******Booting Contiki 2.6******* EUI-64 MAC: 2-0-0-ff-fe-0-0-3
Permissão das mensagens RPL e do encaminhamento	RPL Enabled Routing Enabled	RPL Enabled Routing Enabled	RPL Enabled Routing Enabled
Início do serviço RPL e UDP	RPL started UDP server started	RPL started UDP client process started	RPL started UDP client process started
Atribuição de endereços IPv6 e do prefixo	Adding prefix fe80::length 64, flags 0, Valid lifetime 0, Preffered lifetime 0 RPL: My path ETX to the root is 0.0 RPL: Node set to be a DAG root with DAG ID aaaa::ff:fe00:1 RPL: Prefix set - will announce this in DIOs	Adding prefix fe80::length 64, flags 0, Valid lifetime 0, Preffered lifetime 0	Adding prefix fe80::length 64, flags 0, Valid lifetime 0, Preffered lifetime 0
Criação da DAG e servidor	created a new RPL dag Created a server connection with remote address :: local/remote port 5678/8765	Created a connection with the server :: local/remote port 8765/5678	Created a connection with the server :: local/remote port 8765/5678

Demostração dos endereços IPv6	Server IPv6 addresses: :: aaaa::ff:fe00:1 fe80::ff:fe00:1	Client IPv6 addresses: aaaa::ff:fe00:2 fe80::ff:fe00:2	Client IPv6 addresses: aaaa::ff:fe00:3 fe80::ff:fe00:3
Lista de nós vizinhos	Neighbors [20 max] <none>	Neighbors [20 max] <none>	Neighbors [20 max] <none>
Lista de rotas de encaminhamento	Routes [20 max] <none>	Routes [20 max] <none>	Routes [20 max] <none>
Envio da mensagem DIO	RPL: My path ETX to the root is 0.0 RPL: Sending prefix info in DIO for aaaa:: RPL: Sending a multicast-DIO with rank 256		
Recepção da mensagem DIO no cliente 1 proveniente do servidor		neighbor-info: packet received from 0.1 IPv6 packet received from fe80::ff:fe00:1 to ff02::1a icmp6_input: length 124 type: 155 Received an RPL control message RPL: Received a DIO from fe80::ff:fe00:1	

Adicionamento de um servidor à lista de vizinhos do cliente 1		Adding neighbor with ip addr fe80::ff:fe00:1link addr 02:00:00:ff:fe:00:00:01state 1 RPL: Neighbor added to neighbor cache fe80::ff:fe00:1, 02:00:00:ff:fe:00:00:01 RPL: New instance detected: Joining... RPL: Adding fe80::ff:fe00:1 as a parent: succeeded	
Adicionamento à DAG		RPL: My path ETX to the root is 5.0 RPL: Joined DAG with instance ID 30, rank 1536, DAG ID aaaa::ff:fe00:1	RPL: Received consistent DIO RPL: preferred DAG aaaa::ff:fe00:1, rank 768, min_rank 768, parent rank 512, parent etx 128, link metric 16, instance etx 256 RPL: My path ETX to the root is 2.0 RPL: Sending DAO with prefix aaaa::ff:fe00:3 to fe80::ff:fe00:2
Adicionamento do servidor à lista de encaminhamento		RPL: Adding default route through fe80::ff:fe00:1 uip_ds6_defrt_add: adding default route to fe80::ff:fe00:1	

Recepção de mensagem DIO do cliente 1	neighbor-info: packet received from 0.2 IPv6 packet received from fe80::ff:fe00:2 to ff02::1a icmp6_input: length 124 type: 155 Received an RPL control message RPL: Received a DIO from fe80::ff:fe00:2		neighbor-info: packet received from 0.2 IPv6 packet received from fe80::ff:fe00:2 to ff02::1a icmp6_input: length 124 type: 155 Received an RPL control message RPL: Received a DIO from fe80::ff:fe00:2
Adicionamento do cliente 1 à lista de vizinhos	Adding neighbor with ip addr fe80::ff:fe00:2link addr 02:00:00:ff:fe:00:00:02state 1 RPL: Neighbor added to neighbor cache fe80::ff:fe00:2, 02:00:00:ff:fe:00:00:02		Adding neighbor with ip addr fe80::ff:fe00:2link addr 02:00:00:ff:fe:00:00:02state 1 RPL: Neighbor added to neighbor cache fe80::ff:fe00:2, 02:00:00:ff:fe:00:00:02 RPL: New instance detected: Joining... RPL: Adding fe80::ff:fe00:2 as a parent: succeeded
Recepção da mensagem DAO do cliente 1	IPv6 packet received from fe80::ff:fe00:2 to fe80::ff:fe00:1 icmp6_input: length 90 type: 155 Received an RPL control message RPL: Received a DAO from fe80::ff:fe00:2 RPL: DAO lifetime: 255, prefix length: 128 prefix: aaaa::ff:fe00:2		

Adicionamento do cliente 1 à lista de encaminhamento	uip-ds6-route: Looking up route for aaaa::ff:fe00:2 uip-ds6-route: No route found uip_ds6_route_add num 1 uip_ds6_route_add: adding route: aaaa::ff:fe00:2 via fe80::ff:fe00:2 RPL: Added a route to aaaa::ff:fe00:2/128 via fe80::ff:fe00:2		
Lista de nós vizinhos	Neighbors [20 max] fe80::ff:fe00:2	Neighbors [20 max] fe80::ff:fe00:1	
Lista de encaminhamento de encaminhamento	Routes [20 max] aaaa::ff:fe00:2/128 (via fe80::ff:fe00:2) 16711046s	Routes [20 max] aaaa::ff:fe00:1/128 (via fe80::ff:fe00:1) 16711046s	
Envio da mensagem DIS			RPL: Sending a DIS to ff02::1a
Recepção da mensagem DIS no cliente 1 proveniente do		neighbor-info: packet received from 0.3 IPv6 packet received from fe80::ff:fe00:3 to	

cliente 2		ff02::1a icmp6_input: length 46 type: 155 Received an RPL control message RPL: Received a DIS from fe80::ff:fe00:3	
Envio de mensagem DIS		RPL: Sending a DIS to ff02::1a	
Recepção da mensagem DIS proveniente do cliente 1	IPv6 packet received from fe80::ff:fe00:2 to ff02::1a icmp6_input: length 46 type: 155 Received an RPL control message RPL: Received a DIS from fe80::ff:fe00:2		IPv6 packet received from fe80::ff:fe00:2 to ff02::1a icmp6_input: length 46 type: 155 Received an RPL control message RPL: Received a DIS from fe80::ff:fe00:2
Recepção da mensagem DAO no cliente 1 proveniente do cliente 2		neighbor-info: packet received from 0.3 IPv6 packet received from fe80::ff:fe00:3 to fe80::ff:fe00:2 icmp6_input: length 90 type: 155 Received an RPL control message RPL: Received a DAO from fe80::ff:fe00:3 RPL: DAO lifetime: 255, prefix length: 128 prefix: aaaa::ff:fe00:3	

Adicionamento de o cliente 2 à lista de encaminhamento		uip-ds6-route: Looking up route for aaaa::ff:fe00:3 uip-ds6-route: No route found uip_ds6_route_add num 1 uip_ds6_route_add: adding route: aaaa::ff:fe00:3 via fe80::ff:fe00:3 RPL: Added a route to aaaa::ff:fe00:3/128 via fe80::ff:fe00:3 RPL: Forwarding DAO to parent fe80::ff:fe00:1	
Lista de nós vizinhos		Neighbors [20 max] fe80::ff:fe00:3 fe80::ff:fe00:1	Neighbors [20 max] fe80::ff:fe00:2
Lista de rotas de encaminhamento	Routes [20 max] aaaa::ff:fe00:2/128 (via fe80::ff:fe00:3) 16711406s aaaa::ff:fe00:3/128 (via fe80::ff:fe00:3) 16711420s	Routes [20 max] aaaa::ff:fe00:3/128 (via fe80::ff:fe00:3) 16711072s aaaa::ff:fe00:1/128 (via fe80::ff:fe00:1) 16711046s	Routes [20 max] <none>

Envio de mensagem		DATA send to 1 'Hello 2'	DATA send to 1 'Hello 1'
Recepção de mensagem	DATA recv 'Hello 2' from 2 DATA recv 'Hello 1' from 3		

Anexo 3

Exemplo Border router

```
user@instant-contiki:~/contiki/tools$ sudo ./tunslip6 aaaa::1/64 -s /dev/ttyUSB0 -B 38400 -v6
********SLIP started on ``/dev/ttyUSB0''
opened tun device ``/dev/tun0''
ifconfig tun0 inet `hostname` up
ifconfig tun0 add aaaa::1/64
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:127.0.1.1  P-t-P:127.0.1.1  Mask:255.255.255.255
          inet6 addr: fe80::1/64 Scope:Link
          inet6 addr: aaaa::1/64 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

Packet from TUN of length 76 - write SLIP
0000 60 00 00 00 00 24 00 01 fe 80 00 00 00 00 00 00 00 00 00 00 00 00 00 01 ff 02 00 00 00 00 00 00 00 0
0 00 00 00 00 00 16 3a 00 05 02 00 00 01 00 8f 00 70 08 00 00 00 01 04 00 00 00 ff 02 00 00 00 00 00 00 0
0 00 00 00 00 00 00 02
```

Figura 70 - Criação do túnel SLIP

```
*******Booting Contiki 2.6*******
EUI-64 MAC: 2-0-0-ff-fe-0-0-1
nullmac sicslowmac, channel 26 , check rate 128 Hz tx power 0
RPL Enabled
Routing Enabled
START Time to check nodes
CLOCK_CONF_SECOND 128 =1sec * CONF_CLOCK= 60
 CLOCK_CONF_SECOND*CONF_CLOCK= 7680 =60sec
Online
*** Address:aaaa::1 => aaaa:0000:0000:0000
UDP server started
Created a server connection with remote address :: local/remote port 5678/0
Got configuration message of type P
Setting prefix aaaa::
Server IPv6 addresses:
 aaaa::ff:fe00:1
 fe80::ff:fe00:1

Addresses [4 max]
aaaa::ff:fe00:1
fe80::ff:fe00:1

Neighbors [20 max]
  <none>
Routes [20 max]
 <none>
```

Figura 71 - Inicialização do dispositivo

```
c:\>tracert aaaa::ff:fe00:1

Rastreio da rota para aaaa::ff:fe00:1 até o máximo de 30 saltos

  1    <1 ms     <1 ms     <1 ms  bbbb::ff:fe00:1
  2   540 ms     65 ms     65 ms  aaaa::ff:fe00:1

Rastreio concluído.
```

Figura 72 - Trace do computador para o dispositivo

Anexo 4

Lista de requisitos

Anexo 4.1 – Geral

ID	Funcionalidade	Descrição	Actor	Estado	Notas
Ref-G_1.	Comunicação IPv6 em todos os dispositivos.	Toda a comunicação deverá ser feita com o uso do protocolo IPv6.	Coordenador Nós Aplicação Java	Bem-sucedido	Existiu um pequeno problema na eprom dos dispositivos, encontrava-se configurada com o mesmo código, o que fez com que os dispositivos obtivessem sempre o mesmo endereço IP. A solução passou por editar o endereço mac, directamente no ficheiro "*params.h*" localizado em "*platform/avr-atmega128rfa1/*" no Contiki. "#define PARAMS_EUI64ADDR {0x02, 0x00, 0x00, 0xff, 0xfe, 0x00, 0x00, 0x01}"
Ref-G_2.	Comunicação UDP.	A comunicação deverá ser toda feita com o uso de protocolo UDP.	Coordenador Nós Aplicação Java	Bem-sucedido	
Ref-G_3.	Tem de existir uma comunicação directa entre dispositivos.	O protocolo IPv6 já permite essa comunicação.	Coordenador Nós Aplicação Java	Bem-sucedido	

Ref-G_4.	Capacidade de avisar os utilizadores caso haja alterações na rede ou dados captados nos nós.	Sempre que haja alteração na rede (um nó foi adicionado ou removido da rede), o coordenador deverá notificar o utilizador. Sempre que um dado (LED ou Luminosidade) seja alterado, o nó deverá notificar o utilizador.	Coordenador Nós Aplicação Java	Bem-sucedido	A solução passou por criar uma lista ligada tanto no coordenador como em cada nó, com a informação do IP e Porto de cada ligação estabelecida pela aplicação. Ao fazer este requisito, encontrou-se um problema de overflow de memória, que provocava o reinício do dispositivo e por consequência a perda das ligações pré-estabelecidas. A solução foi limitar o tamanho da lista de ligações com o valor máximo de 4.
Ref-G_5.	Capacidade de conectar a qualquer nó através de um computador na Internet.	O acesso aos nós deve ser feito de qualquer computador da Internet, sabendo apenas o IP e porto do nó.	Coordenador Nós Aplicação Java	Bem-sucedido	A solução passou por criar a Aplicação Java

Anexo 4.2 – Coordenador

ID	Funcionalidade	Descrição	Actor	Estado	Notas
Ref-C_1._	Adicionar uma ligação.	Sempre que um utilizador deseje receber informação (endereços IPv6) da rede 6LoWPAN, deverá adicionar uma ligação à lista ligada referida no requisito Ref-G_4.	Coordenador Aplicação Java	Bem-sucedido	Número máximo de ligações é 4, devido ao problema referido no requisito Ref-G_4.
Ref-C_2.	Remover uma ligação.	Deverá ser possível remover uma ligação específica ou todas as ligações guardadas na lista ligada referida no requisito Ref-G_4.	Coordenador Aplicação Java	Bem-sucedido	
Ref-C_3.	Enviar os nós da rede.	Para que aplicação Java se possa conectar aos nós, é necessário que o coordenador envie os endereços IP de todos os nós existentes.	Coordenador Aplicação Java	Bem-sucedido	
Ref-C_4.	Adicionar nó à rede.	O RPL permite adicionar automaticamente.	Coordenador Nós	Bem-sucedido	
Ref-C_5.	Remover nó da rede.	O RPL permite remover automaticamente.	Coordenador Nós	-	Até à data, este requisito não foi concluído.

Anexo 4.3 – Nós

ID	Funcionalidade	Descrição	Actor	Estado	Notas
Ref-N_1.	Adicionar uma ligação.	Sempre que possível o nó deverá adicionar uma ligação à lista ligada referida no requisito Ref-G_4.	Nós Aplicação Java	Bem-sucedido	
Ref-N_2.	Remover uma ligação.	Deverá ser possível remover uma ligação específica ou todas as ligações guardadas na lista ligada referida no requisito Ref-G_4.	Nós Aplicação Java	Bem-sucedido	
Ref-N_3.	Capacidade de captar, por exemplo, luminosidade.		Nós	Bem-sucedido	Existiu apenas uma pequena dificuldade na escolha da resistência, para o controlo da variação dos valores. A solução passou por usar uma resistência variável para encontrar o melhor valor de resistência.
Ref-N_4.	Visualizar estado do LED.		Nós Aplicação Java	Bem-sucedido	Esta opção poderá ser acedida por ligações não registadas na lista ligada referida no requisito Ref-G_4.
Ref-N_5.	Alterar estado do LED.		Nós Aplicação Java	Bem-sucedido	

Ref-N_6.	Notificar a alteração do estado do LED.	Quando existe uma alteração no estado do LED, este deverá ser notificado pelos utilizadores.	Nós Aplicação Java	Bem-sucedido	O nó irá consultar a lista ligada referida no requisito Ref-N_1, e enviará para todas as ligações o estado actual do LED.
Ref-N_7.	Visualização da luminosidade.		Nós Aplicação Java	Bem-sucedido	
Ref-N_8.	Notificação da alteração da luminosidade.	Quando existe alteração no valor da luminosidade, este deverá ser notificado pelos utilizadores.	Nós Aplicação Java	Bem-sucedido	O nó irá consultar a lista ligada referida no requisito Ref-N_1, e enviará para todas as ligações o valor actual da luminosidade.

Anexo 4.4 – Aplicação Java

ID	Funcionalidade	Descrição	Actor	Estado	Notas
Ref-A_1._	Conectar ao coordenador.		Coordenador Aplicação Java	Bem-sucedido	
Ref-A_2.	Receber dados do coordenador.	A qualquer momento poderá entrar ou sair um nó da rede, o coordenador envia essa informação para aplicação disponibilizando-a.	Coordenador Aplicação Java	Bem-sucedido	
Ref-A_3.	Conectar ao nó.		Nós Aplicação Java	Bem-sucedido	
Ref-A_4.	Receber estado do LED do nó.		Nós Aplicação Java	Bem-sucedido	
Ref-A_5.	Alterar estado do LED do nó.		Nós Aplicação Java	Bem-sucedido	
Ref-A_6.	Actualizar estado do LED pelo respectivo nó.	Caso seja necessário alterar o estado do LED, as restantes aplicações deverão tomar conhecimento.	Nós Aplicação Java	Bem-sucedido	Para receber a actualização de que o LED foi alterado, a aplicação deverá estar registada no nó.

Ref-A_7.	Receber o valor da luminosidade.		Nós Aplicação Java	Bem-sucedido	
Ref-A_8. A	Actualizar o valor da luminosidade pelo respectivo nó.	A qualquer momento a luminosidade do nó pode ser actualizada no nó, este enviará a notificação de actualização a todas as aplicações.	Nós Aplicação Java	Bem-sucedido	Para receber a actualização da alteração da luminosidade, a aplicação deverá estar registada no nó.
Ref-A_9.	Desligar do coordenador.		Coordenador Aplicação Java	Bem-sucedido	
Ref-A_10.	Desligar do nó.		Nós Aplicação Java	Bem-sucedido	

Anexo 5

Testes do Projecto

```
user@instant-contiki:/$ nc -s bbbb::ff:fe00:1 -p 6657 -u aaaa::ff:fe00:1 5678
ctc   Envio de pedido de registo ao coordenador pela aplicação
onc-0-8765- Confirmação e dados do coordenador
updatec-1-aaaa:0000:0000:0000:0000:00ff:fe00:0004 Actualização de um novo nó no coordenador
ctc   Envio de um novo pedido de registo
already_exist- Rejeição do coordenador
exit  Pedido de eliminação do registo
removed- Confirmação da eliminação da aplicação no registo
```

Figura 73 - Teste de conexão do netcat ao coordenador

```
user@instant-contiki:/$ nc -s bbbb::ff:fe00:1 -p 5555 -u aaaa::ff:fe00:3 8765
ctn   Envio de pedido de registo ao nó pela aplicação
nodev-place01-1023-on- Confirmação e dados do nó
ctn Tentativa novo registo
already_exist- Rejeição do nó
led  Pedido do estado do LED
ledstate-on- Envio do estado do LED
uled  Pedido de alteração do LED
ledstate-off-ledstate-off- Confirmação de alteração do estado do led
adc   Pedido do valor da luminosidade
adc-552- Envio do valor da luminosidade
exit  Pedido de eliminação do registo
removed-  Confirmação de eliminação da aplicação no registo
adc   Pedido do valor da luminosidade sem registo
adc-1023- Envio do valor da luminosidade
led   Pedido do estado do LED sem registo
ledstate-off- Envio do estado do LED
```

Figura 74 - Teste de conexão do netcat a um nó

```
*******Booting Contiki 2.6*******
EUI-64 MAC: 2-0-0-ff-fe-0-0-1
nullmac sicslowmac, channel 26 , check rate 128 Hz tx power 0
RPL Enabled
Routing Enabled
START Time to check nodes
CLOCK_CONF_SECOND 128 =1sec * CONF_CLOCK= 60
 CLOCK_CONF_SECOND*CONF_CLOCK= 7680 =60sec
Online
*** Address:aaaa::1 => aaaa:0000:0000:0000
UDP server started
Created a server connection with remote address :: local/remote port 5678/0
Got configuration message of type P
Setting prefix aaaa::
Server IPv6 addresses:
 aaaa::ff:fe00:1
 fe80::ff:fe00:1

Addresses [4 max]
aaaa::ff:fe00:1
fe80::ff:fe00:1

Neighbors [20 max]
  <none>
Routes [20 max]

Routes [20 max]
  <none>
---------
Never-used stack > 6630 bytes
```

Figura 75 - Inicio de ligação do coordenador ao túnel

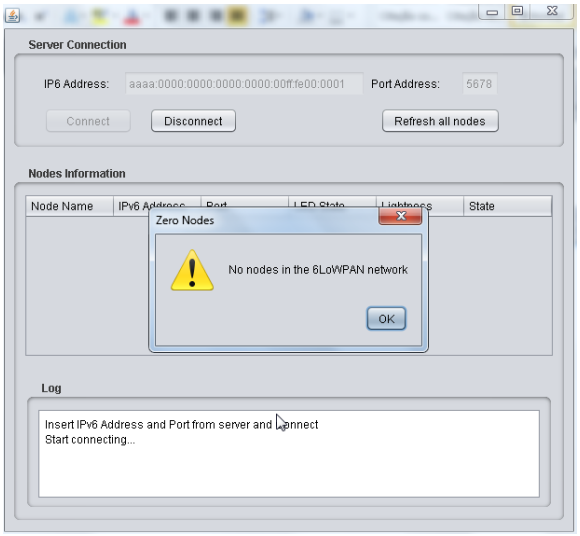


Figura 76 - Inicio de ligação ao coordenador sem nós na rede

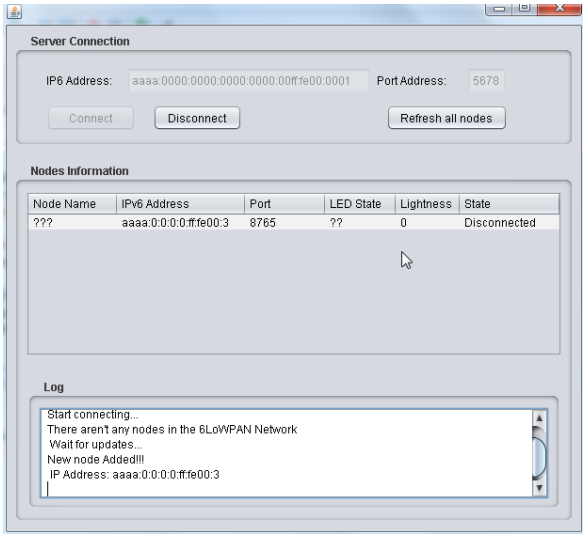


Figura 77 - Adição de um nó relativo à actualização do coordenador

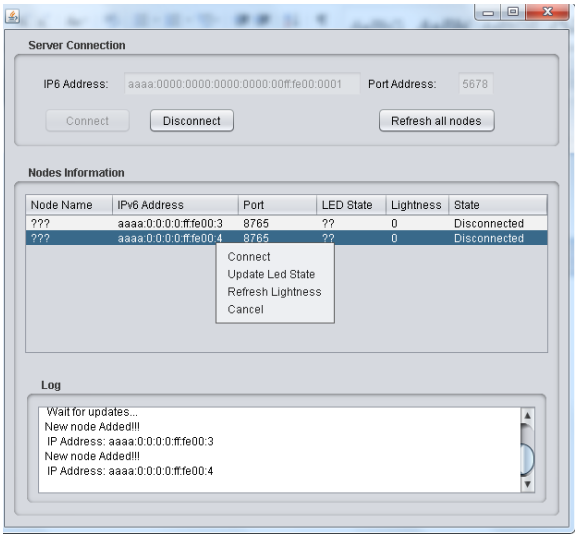


Figura 78 - Opções disponíveis para os nós na aplicação sem registo

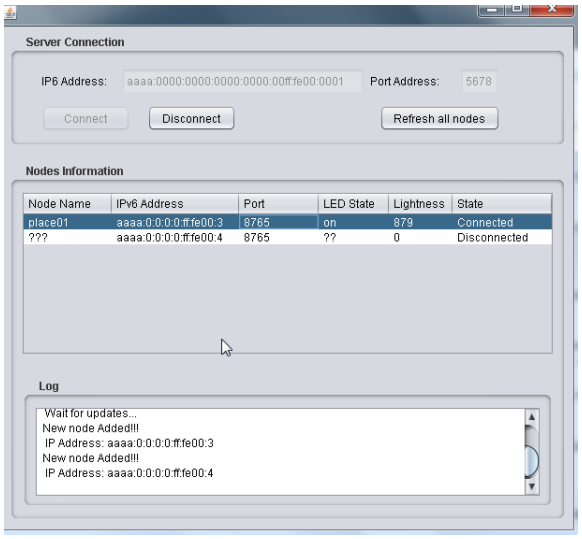


Figura 79 - Dados recebidos do nó depois da conexão da aplicação

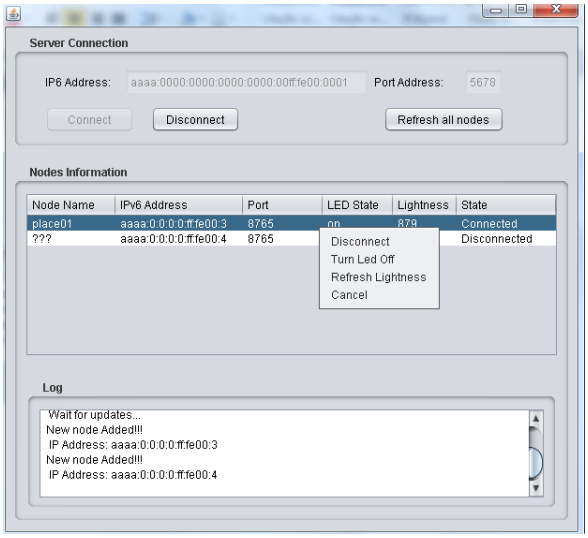


Figura 80 - Opções disponíveis para os nós na aplicação registada

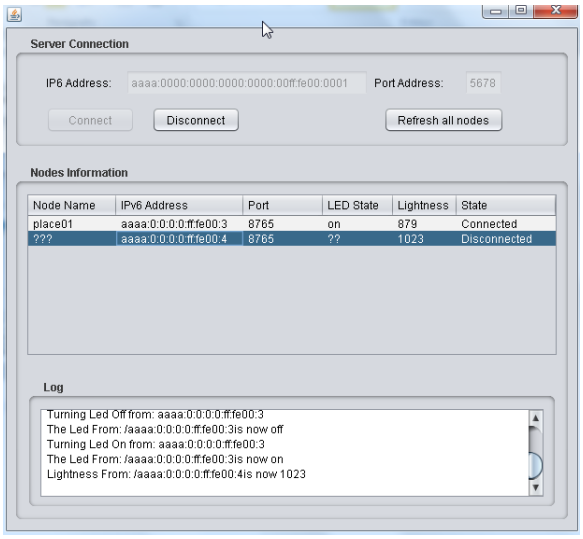


Figura 81 - Actualização do valor da luminosidade

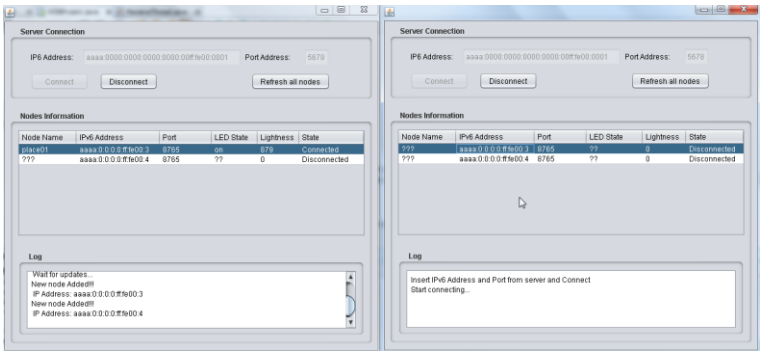


Figura 82 - Conexão da segunda aplicação

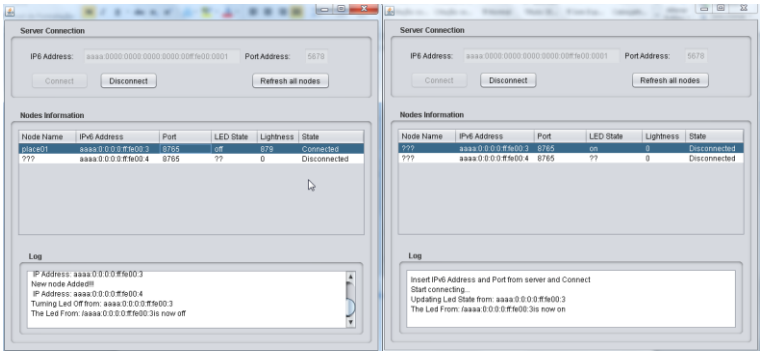


Figura 83 - Actualização do estado do LED na segunda aplicação

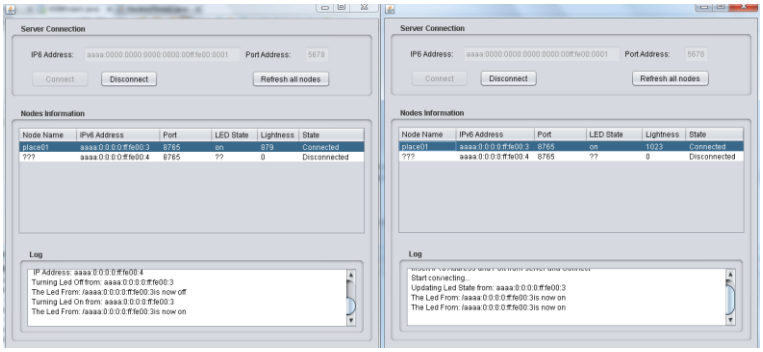


Figura 84 - Alteração do LED com registo da segunda aplicação