Capítulo 1

Internet das Coisas: da Teoria à Prática.

Bruno P. Santos, Lucas A. M. Silva, Bruna S. Peres, Clayson S. F. S. Celes, João B. Borges Neto, Marcos Augusto M. Vieira, Luiz Filipe M. Vieira, Olga N. Goussevskaia e Antonio A. F. Loureiro

Departamento de Ciência da Computação – Instituto de Ciências Exatas
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brasil

{bruno.ps, lams, bperes, claysonceles, joaoborges, mmvieira, lfvieira, olga, loureiro}@dcc.ufmg.br

Abstract

Resumo

A proliferação de objetos com capacidade de monitoramento, processamento e comunicação tem sido crescente nos últimos anos. Diante disso, aparece o cenário de Internet das Coisas (Internet of Things - IoT) no qual objetos podem se conectar à Internet e prover comunicação entre usuários, dispositivos, possibilitando novas aplicações. Várias questões teóricas e práticas surgem no desenvolvimento da IoT, por exemplo, como conectar esses objetos à Internet e como endereçar os objetos. Aliado a essas perguntas diversos desafios devem ser superados, por exemplo, como lidar com as restrições de processamento, largura de banda e energia dos dispositivos. Neste sentido, novos paradigmas de comunicação e roteamento podem ser explorados. Questões relacionadas ao endereçamento IP e como adaptá-lo precisam ser respondidas. Oportunidades de novas aplicações em uma rede de objetos inteligentes aparecem e, junto com essas aplicações, também surgem novos desafios. O objetivo deste minicurso é descrever o estado atual da Internet das Coisas da teoria à prática, explorando os desafios e questões de pesquisa. Através de uma bordagem crítica, é exposto uma visão geral da área, ilustrando soluções parciais ou totais propostas na literatura para as questões mencionadas, além de destacar os principais desafios e oportunidades que a área oferece.

1.1. Introdução

A Internet das Coisas (do inglês *Internet of Things (IoT)*) emergiu dos avanços de várias áreas como sistemas embarcados, microeletrônica, comunicação e tecnologia de informações. De fato, a IoT tem ganhado enfoque no cenário acadêmico e industrial, porque espera-se que seu impacto seja significante. Este capítulo aborda a Internet das Coisas através de uma perspectiva teórica e prática. O conteúdo aqui abordado explora a estrutura, a organização e os eventuais desafios e aplicações da IoT. Nesta seção, serão conceituadas a IoT e os objetos inteligentes. Além disso, são postos em evidência a perspectiva histórica da Internet das Coisas e as motivações que influenciam o aumento dos interesses, expectativas e pesquisas na área. Logo em seguida, são introduzidos os blocos básicos de construção da IoT. Para iniciar a discussão, será levantada a seguinte questão: o que é a Internet das Coisas?

A Internet das Coisas, em poucas palavras, nada mais é que uma extensão da Internet atual. Esta extensão é feita ao proporcionar que objetos do dia-a-dia (quaisquer que sejam) se conectem à Internet. A conexão com a rede mundial de computadores viabilizará, primeiro, controlar remotamente os objetos e, segundo, permitir que os próprios objetos sejam acessados como provedores de serviços. Estas novas habilidades, dos objetos comuns, geram um grande número de oportunidades tanto no âmbito acadêmico quanto no industrial. Todavia, estas possibilidades apresentam riscos e acarretam amplos desafios técnicos e sociais.

A IoT tem alterado aos poucos o conceito de redes de computadores, neste sentido, é possível notar a evolução do conceito ao longo do tempo como mostrado a seguir. Para Tanenbaum [Tanenbaum 2002], "Rede de Computadores é um conjunto de computadores autônomos interconectados por uma única tecnologia". Entende-se que tal tecnologia de conexão pode ser de diferentes tipos (fios de cobre, fibra ótica, ondas eletromagnéticas ou outras). Em 2011, Peterson definiu em [Peterson and Davie 2011] que a principal característica das Redes de Computadores é a sua generalidade, isto é, elas são construídas sobre dispositivos de propósito geral e não são otimizadas para fins específicos tais como as redes de telefonia e TV. Já em [Kurose and Ross 2012], os autores argumentam que o termo "Redes de Computadores" começa a soar um tanto envelhecido devido à grande quantidade de equipamentos e tecnologias não tradicionais que são usadas na Internet.

Os objetos inteligentes, definidos mais adiante, possuem papel fundamental na evolução acima mencionada. Isto porque os objetos possuem capacidade de comunicação e processamento aliados a sensores, os quais transformam a utilidade destes objetos. Atualmente, não só computadores convencionais estão conectados à grande rede, como também uma grande heterogeneidade de equipamentos tais como TVs, *Laptops*, automóveis, *smartphones*, consoles de jogos, *webcams* e a lista aumenta a cada dia. Neste novo cenário, a pluralidade é crescente e previsões indicam que mais de 40 bilhões de dispositivos estarão conectados até 2020 [Forbes 2014]. Usando os recursos deses objetos será possível detectar seu contexto, controlá-lo, viabilizar troca de informações uns com os outros, acessar serviços da Internet e interagir com pessoas. Concomitantemente, uma gama de novas possibilidades de aplicações surgem (ex: cidades inteligentes (*Smart Cities*), saúde (*Healthcare*), casas inteligentes (*Smart Home*)) e desafios emergem (regulamentações, segurança, padronizações). É importante notar que um dos elementos

cruciais para o sucesso da IoT encontra-se na padronização das tecnologias. Isto permitirá que a heterogeneidade de dispositivos conectados à Internet cresça, tornando a IoT uma realidade. Também é essencial frisar que nos últimos meses e nos próximos anos serão vivenciados os principais momentos da IoT, no que tange as definições dos blocos básicos de construção da IoT.

Parafraseado os autores [Mattern and Floerkemeier 2010], ao utilizar a palavra "*Internet*"no termo "*Internet of Things*"como acima mencionado, pode-se fazer uma analogia com a Web nos dias de hoje, em que brevemente as "coisas"terão habilidades de comunição umas com as outras, proverão e usarão serviços, proverão dados e poderão reagir a eventos. Outra analogia, agora mais técnica, é que IoT vista como uma pilha de protocolos utilizados nos objetos inteligentes.

Na IoT, eventualmente, a unidade básica de *hardware* apresentará ao menos uma das seguintes características [Ruiz et al. 2004]: i) unidade(s) de processamento; ii) unidade(s) de memória; iii) unidade(s) de comunicação e; iv) unidade(s) de sensor(es) ou atuador(es). Aos dispositivos com essas qualidades é dado o nome de *objetos inteligentes* (*Smart Objects*). Estes objetos, ao estabelecerem comunicação com outros dispositivos, manifestam o conceito de estarem em rede, como discutido anteriormente.

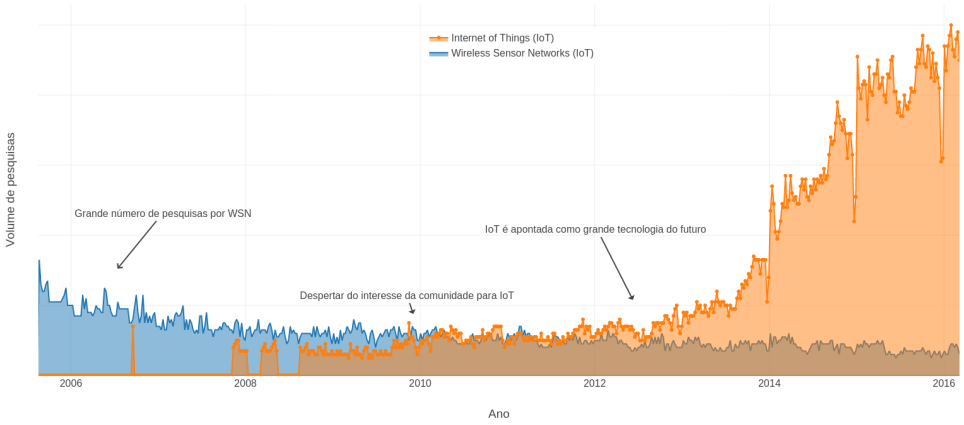


Figura 1.1. Volume de pesquisas no Google sobre *Wireless Sensor Networks* e *Internet of Things*[2].

1.1.1. Perspectiva histórica da IoT

Kevin Ashton comentou, em junho de 2009 [Ashton 2009], que o termo *Internet of Things* foi primeiro utilizando em seu trabalho titulado "*I made at Procter & Gamble*"em 1999. Na época, a IoT era altamente relacionada ao uso da tecnologia RFID. Contudo, o termo ainda não era foco de grande número de pesquisas como pode ser visto na Figura 1.1. Por volta de 2005, o termo bastante procurado (tanto pela academia quando indústria) e que apresenta relação com a IoT foi Redes de Sensores Sem Fio (RSSF) (do inglês *Wireless Sensor Networks (WSN)*). Estas redes trazem avanços na automação residencial e industrial [Kelly et al. 2013, Da Xu et al. 2014], bem como técnicas para ex-

[2]Fonte: https://www.google.com/trends/

plorar as diferentes limitações dos dispositivos (ex: memória e energia), escalabilidade e robustez da rede [Loureiro et al. 2003]. Nos anos seguintes (entre 2008 e 2010), o termo Internet das Coisas ganhou popularidade rapidamente. Isto se deve ao amadurecimento das RSSF e o crescimento das expectativas sobre a IoT. A Figura 1.1 mostra que em 2010, a buscas para IoT dispararam chegando a ultrapassar as pesquisas sobre RSSF.

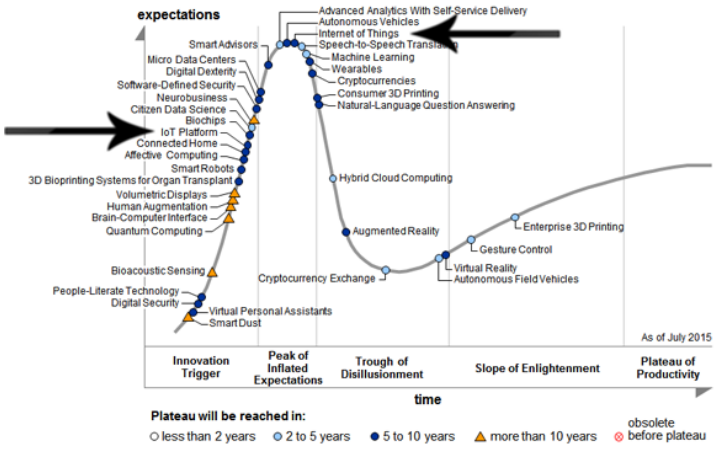


Figura 1.2. Tecnologias emergentes[4].

A IoT foi identificada como uma tecnologia emergente em 2012 por especialistas da área [Gartner 2015]. A Figura 1.2 apresenta uma maneira de representar o surgimento, adoção, maturidade e impacto de diversas tecnologias chamada de *Hype Cycle*. Em 2012, foi previsto que a IoT levaria entre 5 e 10 anos para ser adotada pelo mercado e hoje é vivenciado o maior pico de expectativas sobre a tecnologia no âmbito acadêmico e industrial. Também pode-se notar o surgimento das primeiras plataformas da IoT (discutidas mais adiante) que recebem grande expectativa da comunidade nos próximos anos. Estes fatos podem servir de incentivo inicial para despertar a curiosidade do leitor para a área, bem como indica o motivo do interesse da comunidade científica e industrial para a IoT.

1.1.2. Motivação para manutenção da evolução da IoT

Ao conectar objetos com diferentes recursos a uma rede, potencializa-se o surgimento de novas aplicações. Neste sentido, conectar esses objetos à Internet significa criar a Internet das Coisas. Na IoT, os objetos podem prover comunicação entre usuários, dispositivos. Com isto emerge uma nova gama de aplicações, tais como coleta de dados de pacientes e monitoramento de idosos, sensoriamento de ambientes de difícil acesso e inóspitos, entre outras [Sundmaeker et al. 2010].

Entretanto, ao passo que a possibilidade de novas aplicações é crescente, os desafios de conectar os objetos à Internet surgem. Naturalmente, os objetos são heterogêneos, isto é, divergem em implementação, recursos e qualidade e, em geral, apresentam limitações [Loureiro et al. 2003] de energia, capacidade de processamento, memória e comunicação. Questões tanto teóricas quanto práticas surgem, por exemplo, como prover endereçamento aos dispositivos, como encontrar rotas de alta vazão e melhor taxa de entrega mantendo o baixo o uso dos recursos limitados. Deste modo, fica evidente a necessidade da adaptação dos protocolos existentes (ex: o IP). Além disso, sabe-se que os paradigmas de comunicação e roteamento nas redes de objetos inteligentes podem não se-

[4]Fonte: http://www.gartner.com/newsroom/id/3114217

guir os mesmos padrões de uma rede como a Internet [Chaouchi 2013], sendo assim como devem ser explorados para extrair melhor desempenho das redes de objetos inteligentes.

Novos desafios surgem enquanto são criadas novas aplicações para IoT. Os dados providos pelos objetos agora podem apresentar imperfeições (calibragem do sensor), inconsistências (fora de ordem, *outliers*), serem de diferentes tipos (gerados por pessoas, sensores físicos, fusão de dados). Assim, as aplicações e algoritmos devem ser capazes de lidar com esses desafios sobre os dados. Outro exemplo diz respeito ao nível de confiança sobre os dados obtidos dos dispositivos da IoT e como/onde pode-se empregar esses dados em determinados cenários. Deste modo, os desafios impostos por essas novas aplicações devem ser explorados e soluções devem ser propostas para que a IoT contemplem as expectativas em um futuro próximo como previsto na Figura 1.2.

Alguns autores já pressupõem que a IoT será a nova revolução da tecnologia da informação [Ashton 2009, Forbes 2014, Wang et al. 2015]. Sendo assim, a IoT possivelmente não deve ser entendida como um fim, mas sim um meio de alcançar algo maior, a saber a computação ubíqua. Desde modo, esta relação entre IoT e a computação ubíqua será objeto de detalhamento em momento oportuno ao longo do capítulo.

1.1.3. Blocos Básicos de Construção da IoT

A IoT pode ser vista como um combinado de diversas tecnologias, as quais são complementares no sentido de viabilizar a integração dos objetos no ambiente físico ao mundo virtual. Na Figura 1.3 são apresentados os blocos básicos de construção da IoT sendo eles:

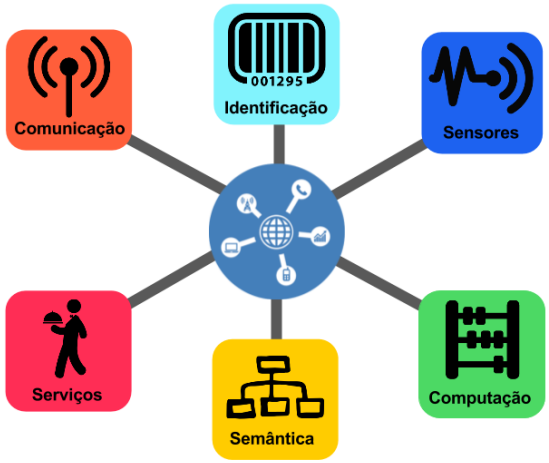


Figura 1.3. Blocos básicos da IoT[6].

- **Identificação**: um dos blocos mais importantes, visto que é primordial identificar os objetos unicamente para então conectá-los à Internet. Tecnologias como RFID, NFC (*Near Field Communication*) e endereçamento IP podem ser empregados para identificar os objetos.
- **Sensores/Atuadores**: os objetos coletam informações sobre o contexto ao seu redor, em seguida, os objetos podem armazenar esses dados, encaminhar. No caso de Atuadores, os objetos podem manipular o ambiente ou reagir de acordo ao dados lidos.
- **Comunicação**: a comunicação também é um dos blocos fundamentais, pois através dela é possível conectar os objetos inteligentes. Além disso, este bloco desempenha papel importante no consumo de energia dos objetos, sendo portanto um fator

[6]Imagem e texto baseados em [Al-Fuqaha et al. 2015, Mattern and Floerkemeier 2010].

crítico, visto que grande quantidade de energia (recurso limitado) é empregada para realizar a comunicação. Exemplos de tecnologias utilizadas nesse item são: WiFi, *Bluetooth*, IEEE 802.15.4, RFID;

- **Computação**: diz respeito à unidade de processamento, por exemplo, micro controladores, processadores, FPGAs. Em outras palavras, tudo o que dá aos objetos inteligentes a capacidade de computar;

- **Serviços**: em geral a IoT pode prover diversas classes de serviços, dentre elas, destacam-se, por exemplo, *Serviços de Identificação*, o quais são os mais básicos e importantes serviços, pois as aplicações os utilizam para mapear as Entidades Físicas (EF) (de interesse do usuário) em Entidades Virtuais (EV), em outras palavras, estes serviços facilitam mapear objetos do mundo real e os respectivos objetos do mundo virtual. Os *Serviços de Agregação de Informações*, os quais coletam e sumariza dados brutos obtidos dos objetos inteligentes que precisam ser processados e enviados para as aplicações. Os *Serviços de Colaboração e Inteligência* são aqueles que agem sobre os serviços de agregação de informações usando os dados obtidos e processados para tomar decisões e reagir de modo adequado. Para finalizar, será pontuado os *Serviços de Ubiquidade* visam prover Serviços de colaboração e Inteligência em qualquer momento e qualquer lugar em que eles sejam necessários;

- **Semântica**: refere-se à habilidade de extração de conhecimento da diversidade de objetos na IoT. Em outras palavras, a semântica trata da descoberta e uso inteligente dos recursos, para possibilitar o provimento de determinado serviço. Além disso, deve efetuar o reconhecimento e análise dos dados para realizar corretamente a tomada de decisões. Para tanto, podem ser usadas diversas tecnologias tais como: *Resource Description Framework (RDF)*, *Web Ontology Language (OWL)* e *Efficient XML Interchange (EXI)*.

1.1.4. Objetivos do minicurso

Este capítulo tem por objetivo apresentar ao leitor uma visão geral da Internet das Coisas. Neste sentido, um amplo espectro de conhecimento será posto para apreciação do leitor. Isto será realizado de modo a deixar o mais claro possível o real significado, a funcionalidade e posicionamento dos principais blocos de construção da IoT.

Após a breve apresentação dos blocos da IoT (Seção 1.1.3) fica evidente o quão ampla é a área. Diante disto, serão discutidos somente alguns destes blocos ao longo do capítulo sendo eles: **Identificação, Comunicação** e **Serviços/Semântica**. Sempre pontuando em momento oportuno questões que envolvem os demais blocos. Estes blocos foram escolhidos, pois capturam a essência da IoT e dá ao leitor condições de explorar com maior facilidade os demais blocos básicos de construção.

1.1.5. Estrutura do capítulo

Este capítulo é dividido em 5 partes principais. A primeira, Seção 1.2 serão abordados pontos sobre os dispositivos e as tecnologias de comunicação para IoT. A segunda, Seção 1.3, discute sobre os *softwares* que orquestram a operação da IoT, pontuando a

identificação única através dos dispositivos com IPv6, bem como modelos de conectividade, protocolos de roteamento e aplicação para IoT e ambientes de desenvolvimento. Na terceira parte, Seção 1.4, serão realizadas duas práticas para consolidar os conteúdo visto. Estas práticas serão o elo para o conteúdo da quarta parte do capítulo, Seção 1.5, o qual aborda o gerenciamento e análise de dados que permeiam os blocos básicos de semântica e serviços. O quinto momento, Seção 1.6, focará na perspectiva futura da IoT, apresentando um olhar diferenciado para a IoT, em que ela é um meio para se atingir a Computação Ubíqua. Finalmente, na Seção 1.7, serão apresentadas as considerações finais e destacados os pontos que não foram cobertos no capítulo, mas sempre fazendo referências para que o leitor possa ter um ponto de partida para eventuais leituras extra.

1.2. Dispositivos e Tecnologias de Comunicação

Nesta primeira seção, serão abordados em mais detalhes a arquitetura básica dos dispositivos inteligentes e as tecnologias de comunicação. O bloco de construção foco desta seção é o de comunicação, o qual permite interligar dispositivos. As tecnologias sem fio são evidenciadas, pois capturam a essência sem fio do ambiente IoT.

1.2.1. Arquiteturas básica dos dispositivos

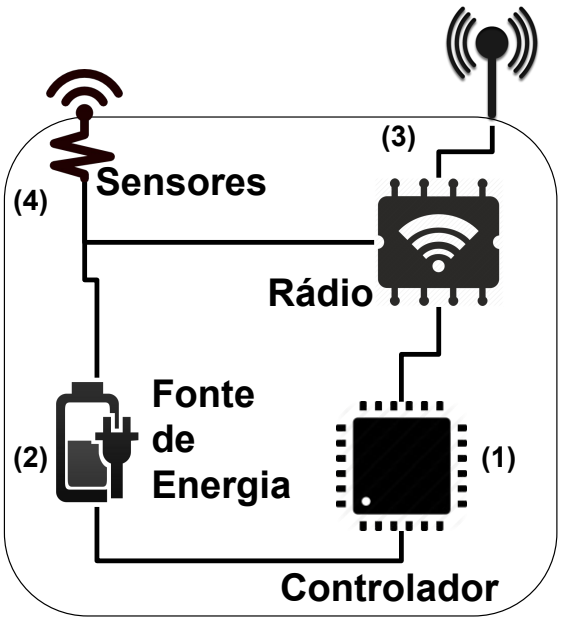


Figura 1.4. Arquitetura de um objeto inteligente[8].

A arquitetura de sistema de um objeto inteligente é composta de cinco blocos principais [Loureiro et al. 2003] unidade de fonte de energia, comunicação, unidade de processamento, unidade de armazenamento e sensores. A unidade de fonte de energia consiste normalmente de uma bateria e um conversor ac-dc e tem a função de alimentar o objeto inteligente. A unidade de comunicação consiste de um canal de comunicação sem fio bidirecional. A maioria das plataformas usam rádio de curto alcance. Outras possíveis soluções incluem laser e mídia infra-vermelho. A unidade de processamento é composta de uma memória interna para armazenamento de dados e programas, um microcontrolador e um conversor analógico-digital para receber sinais do bloco dos sensores. A unidade de armazenamento é uma memória externa que serve como memória secundária, por exemplo, manter um "log" de dados. A unidade de processamento é um bloco que liga um objeto inteligente ao mundo físico e tem um grupo de sensores e atuadores que

dependem da aplicação da IoT.

A Figura 1.4 representa a arquitetura básica de um dispositivo inteligente. O número 1 representa o controlador, geralmente são CPUs utilizadas em sistemas embarcados e possuem baixo consumo energético. O número 2 apresenta as interfaces de comunicação, existem duas opções as cabeadas e as tecnologias sem fio, ambas serão apresentadas na seção seguinte. O número 3 representa a fonte de energia necessária para o funcionamento do nó, podem ser baterias ou rede elétrica convencional. Por último, o número 4 representa sensores e atuadores. Sensores são responsáveis por lidar com grandezas físicas como temperatura, umidade, pressão, presença, etc. Atuadores são dispositivos que realizam conversão de energia elétrica em energia hidráulica, pneumática ou mecânica, um exemplo seriam os motores.

1.2.2. Tecnologias de comunicação

Nesta seção serão abordadas as principais tecnologias de comunicação utilizadas em IoT, sempre pontuando as características mais relevantes de cada um das tecnologia.

Ethernet O padrão Ethernet (IEEE 802.3) foi oficializado em 1983 e está presente em grande parte das redes de hoje. Seu da tecnologia é devida a sua simplicidade e a facilidade de adaptação, manutenção e custo. Atualmente, existem dois tipos de cabos, o cabo de par trançado e os de fibra óptica. Cada tipo de cabo pode alcançar uma vazão de dados diferente. Os cabos de par trançado podem atingir velocidades de até 1Gbps (categoria 5) e são limitados a ter 100m de comprimento (para distâncias maiores é necessário o uso de repetidores). Os cabos de fibra-óptica alcançam valores de vazão de 10Gbps e possuem a limitação de comprimento de até 2000m [Tanenbaum 2011]. O uso do padrão Ethernet apresenta as vantagens de baixo requisito de energia, alta vazão de dados e simplicidade. Entretanto, usar Ethernet em geral implica que os dispositivos estarão fixos (sem mobilidade), o que pode limitar as aplicações em IoT.

Wi-Fi O Wi-Fi é a tecnologia de comunicação mais popular, pois elas está presente em quase todos os lugares, fazendo parte do cotidiano das casas, escritórios, indústrias e até espaços públicos das cidades. O padrão IEEE 802.11 (Wi-Fi) define um conjunto de padrões de transmissão e codificação, que hoje estão disponível em 25% das casas pelo mundo [Alliance 2015]. Uma das principais preocupações do Wi-Fi é a vazão, a primeira versão do padrão IEEE 802.11, lançada em 1997, alcançava a vazão de 2 Mbps, a próxima versão elevou esse valor para 11 Mbps. Atualmente, os valores disponíveis para vazão são 600 Mbps ou 1300 Mbps na versão IEEE 802.11 ac.

O Wi-Fi foi desenvolvido como uma alternativa para o padrão cabeado Ethernet, com pouca, ou talvez nenhuma preocupação com dispositivos que possuem consumo energético limitado, como é o caso das aplicações para IoT. Neste sentido, não é esperado que muitos dispositivos utilizados em IoT adotem o padrão Wi-Fi como principal protocolo de comunicação. Contudo, o Wi-Fi possui algumas vantagens, como alcance de conexão e vazão, o que o torna adequado para navegação na Internet em dispositivos móveis, como *smartphones* e *tablets*. A principal desvantagem do Wi-Fi é o maior consumo de energia, quando comparado com outras tecnologias de comunicação.

[8]Fonte: http://www.gartner.com/newsroom/id/3114217

ZigBee O padrão ZigBee é baseado na especificação IEEE 802.15.4 para a camada de enlace. As principais características do ZigBee são a baixa vazão, reduzido consumo energético, e baixo custo. O ZigBee opera na frequência 2.4-GHz ISM, mas é capaz de operar em outras duas frequências, 868 MHz ou 915 MHz ISM. Esta tecnologia pode alcançar a vazão de 250Kbps, mas na prática somente taxas inferiores são alcançáveis. O ZigBee também permite que os dispositivos entrem em modo *sleep* por longos intervalos de tempo para economizar o consumo de energia, fazendo com que a vida útil da bateria seja prolongada. Atualmente, novos dispositivos ZigBee permitem o uso técnicas de colheita de energia do ambiente (*Energy Harvesting*, discutida mais adiante) para diminuir o uso da bateria por operações [Reiter 2014].

O padrão ZigBee é mantido pela *ZigBee Alliance*, organização que é responsável por gerir o protocolo e garantir a interoperabilidade entre dispositivos. O padrão ZigBee também atende a especificação IP, compatível com IPv6 e formação de rede utilizando a topologia Malha (*Mesh*)[9]. O ZigBee já é adotado em aplicações residenciais e comerciais. A sua integração ao ambiente de Internet das Coisas depende de um *gateway*. Neste caso, o *gateway* é o elemento de rede responsável por permitir a comunicação entre dispositivos que usam ZigBee e os que não usam.

Bluetooth Low Energy *Bluetooth* é um protocolo de comunicação inicialmente criado pela Ericsson para substituir a comunicação serial RS-232 [10]. Atualmente, o órgão *Bluetooth Special Interes Group* (SIG) é responsável por criar, testar e manter a tecnologia. Além disso, Bluetooth é uma das principais tecnologias de rede sem fio para *Personal Area Networks* PANs, que é utilizada em smartphones, headsets, PCs e outros. O *Bluetooth* se divide em: *Bluetooth* Clássico *Basic Rate/Enhanced Data Rate* (BR/EDR), que são as versões 2.0 ou inferiores; *Bluetooth High Speed* (HS), versão 3.0; e o *Bluetooth Low Energy* (BLE), versão 4.0 ou superior. As versões mais antigas do *Bluetooth*, focadas em aumentar a vazão, tornou o protocolo mais complexo e, por consequência, não otimizado para dispositivos com limitações energéticas. Ao contrário das versões anteriores, o BLE possui especificação voltada para baixo consumo de energia, permitindo dispositivos que usam baterias do tamanho de moedas (coin cell batteries) prolonguem a sua vida útil, ao passo que mantém comunicação.

Atualmente o BLE possui três versões, são elas 4.0, 4.1 e 4.2, lançadas em 2010, 2013, e 2014 respectivamente. BLE 4.0 e 4.1 possuem o *Maximum Transmit Unit* (MTU) de 27 bytes, diferentemente da mais nova versão (BLE 4.2) que possui 251 bytes. Outra diferença entre as versões é o tipo de topologia de rede que cada versão pode criar. Na versão 4.0 apenas a topologia estrela é disponível, cada dispositivo pode atuar exclusivamente como *Master* ou como *Slave*. A partir da versão 4.1, um dispositivo é capaz de atuar como *Master* ou *Slave* ao mesmo tempo, permitindo a criação de topologias em malha (*Mesh*). Recentemente foi proposta uma camada de adaptação para dispositivos BLE similar a 6LoWPAN é chamada de 6LoBTLE. A especificação do 6LoBTLE pode ser consultada na RFC 7668[11].

[9] http://www.zigbee.org/zigbee-for-developers/network-specifications/

[10] https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-fact-or-fiction

[11] https://tools.ietf.org/html/rfc7668

3G/4G Os padrões Celular 3G/4G podem também ser aplicados no cenário da IoT. Projetos que precisam alcançar grandes distâncias podem aproveitar as redes 3G/4G. Por outro lado, o consumo energético da tecnologia 3G/4G é alto em comparação com outras tecnologias. Porém, a sua utilização em locais afastados e baixa mobilidade podem compensar esse gasto. No Brasil, a frequência utilizada para o 3G 1900MHz e 2100MHz (UMTS), já o padrão 4G (LTE) utiliza a frequência 2500MHz. A vazão de dados alcançada no padrão 3G é de cerca de 1Mbps e no padrão 4G pode alcançar até 10Mbps [12].

LoRaWan O *Long Range Wide Area Network*, foi projetado e otimizado para prover redes de longa distância (*Wide Area Networks* (WANs)). Como principais características tem-se o baixo custo, a mobilidade, e segurança na comunicação para IoT. Além disso, o padrão oferece suporte a IPv6, a adaptação 6LoWPAN e opera sobre a topologia estrela [13]. O fator atrativo do LoRAWAN é o seu baixo custo e a quantidade de companhias de *hardware* que estão adotando. A vazão de dados alcançada desta tecnologia alcança valores entre 0.3kbps a 50kbps. O consumo de energia da LoRaWan é considerada pequena, o que permite que os dispositivos mantenham-se ativos por longos períodos. A LoRaWANs utiliza a frequência ISM sub-GHz, o que permite que as ondas penetrem grandes estruturas e superfícies com raio de 2km a 5km em meio urbano e 45km no meio rural. Os valores de frequência mais usadas pelo LoRaWan são: 109 MHz, 433 MHz, 866 MHz, 915 MHz. O MTU adotado pelo padrão LoRaWAN é de 256 bytes[14].

Sigfox O padrão SigFox utiliza a tecnologia *Ultra Narrow Band* (UNB) que foi projetada para lidar com pequenas taxas de transferência dados. O padrão ainda é bastante recente, e já possui uma larga cobertura que abrange dezenas de milhares de dispositivos, os quais estão espalhados na Europa e na América do Norte. A SigFox atua como uma operadora para IoT, com suporte a uma série de dispositivos. A principal função é abstrair dificuldades de conexão e prover uma API para que os usuários implementem sistemas IoT com maior facilidade. O raio de cobertura oficialmente relatada, em zonas urbanas, está entre 3km e 10km e em zonas rurais entre 30km a 50km. A vazão de dados varia entre 10bps e 1000bps. O MTU utilizado é de 96 bytes. O SigFox possui baixo consumo de energia e opera na faixa de 900MHz[15].

Finalmente, a Tabela 1.1 resume as tecnologias de comunicação apresentadas nesta seção. As principais características de cada uma são elencadas, o que permite compará-las. Neste sentido, destaca-se a grande variedade de possibilidades para conectar dispositivos. Portanto, é preciso ponderar acerca das características das tecnologias e finalidade do dispositivo para escolher a melhor forma de conectá-lo.

1.2.3. Perspectivas e desafios sobre os objetos inteligentes LOUREIRO

A evolução nas tecnologias de *hardware* empregados na RFID, RSSF , consequentemente, na IoT é dinâmica. Os dispositivos estão cada vez menores e ainda apresentam as características mencionadas nas seções anteriores. Contudo, é notório que as tecnologias de *hardware* empregadas na IoT hoje, certamente não serão as empregadas no futuro. Isto

[12] https://www.opensensors.io/connectivity
[13] http://www.semtech.com/wireless-rf/lora/LoRa-FAQs.pdf
[14] https://www.lora-alliance.org
[15] http://makers.sigfox.com

Protocolo	Alcance	Frequência	Vazão	IPv6	Topologia
Ethernet	100/2000m	N/A	10Gbps	Sim	Variada
Wi-Fi	50m	2.4/5GHz	1300Mbps	Sim	Estrela
BLE	80m	2.4GHz	1Mbps	Sim*	Estrela/Mesh
ZigBee	100m	915MHz/2.4GHz	250kbps	Sim	Estrela/Mesh
3G/4G	35/200km	1900/2100/2500Mhz	1/10Mbps	Sim	Estrela
SigFox	10/50km	868/902MHz	10-1000bps	-	-
LoraWan	2/5km	Sub-GHz	0.3-50kbps	Sim	Estrela

Tabela 1.1. Tabela Comparativa entre tecnologias de conexão.

se deve a possibilidade de empregar novas tecnologias aos dispositivos inteligentes tais como circuitos impressos ou sistemas-em-um-chip [Vasseur and Dunkels 2010]. Neste sentido, esta seção trata das perspectivas e desafios para a IoT. A seguir serão abordados dois pontos centrais. No primeiro, sobre a questão da fonte de energia, no segundo sobre as futuras tecnologias de *hardware* para dispositivo IoT.

1.2.3.1. Bateria e Colheita de Energia (*Energy Harvesting*)

Como mostrado na Figura 1.4, os dispositivos da IoT requerem uma fonte de energia. Atualmente, os objetos inteligentes são alimentados, em geral, por baterias, entretanto existem diversas outras fontes de alimentação como energia elétrica, solar e outras. As baterias (recarregáveis ou não) são as fontes de alimentação mais empregadas nos dispositivos IoT hoje, embora não sejam as mais adequadas para a tarefa. Isto porque, em geral, os dispositivos estão em locais de difícil acesso (ex: embutidos em outros dispositivos) ou simplesmente não é desejável manipulá-los fisicamente para substituição das baterias. Neste sentido, tanto o *hardware* quando to *software* devem ser pensados para estender ao máximo a vida útil destes equipamentos [RERUM 2015].

Uma possível estratégia para mitigar o problema da energia é fazer uso técnica de colheita de energia (do inglês *Energy Harvesting*) [Ramos et al. 2012]. A estratégia consiste em derivar energia de fontes externas ao dispositivo, por exemplo, energia solar, térmica, eólica, cinética. A energia colhida (geralmente em pequenas quantidades) é armazenada e deve ser utilizada para suprir as necessidades dos objetos como comunicação e processamento, como é feito pelos autores [Liu et al. 2013]. Contudo, a colheita de energia também trás ao menos um outro desafio que discutiremos a seguir, o qual pode ser ponto de partida para novas pesquisas.

Dado que os dispositivos possuam a capacidade de colher energia do ambiente em que estão inseridos diversas questões surgem. Por exemplo, como programar as atividades que o dispositivo deve desempenhar dado o orçamento de energia (*Energy Budget*). Em outras palavras, como gastar a energia em função das atividades que se deve fazer? Para exemplificar, imagine a situação em que dispositivo deve realizar tarefas durante as 24 horas do dia, porém somente quando há luz do sol é possível captar energia do ambiente e o dispositivo não está acessível fisicamente. Além disso, sabe-se que a carga da bateria

não suporta 24 horas com o dispositivo em operação perenemente. Sendo assim, as atividades do dispositivo deve ser realizada de modo intermitente, ou seja, alterando entre realizar os comandos requeridos e entrar em modo de espera ou baixa potência (*sleep mode*). Portanto, a atividade deve ser realizada de modo inteligente dado a demanda de atividades e orçamento de energia residual e adquerida do ambiente.

1.2.3.2. Tecnologias de *hardware*

Diante do que foi mencionado nas seções anteriores fica claro que os objetos inteligentes e tecnologias de comunicação apresentam diferentes características e limitações. Ao passo que os dispositivos diminuem, as limitações tendem a aumentar devido ao espaço reduzido dos dispositivos. Anos atrás foi previsto a possibilidade de criar dispositivos em escala nanométrica e hoje isto é uma realidade com os *Microelectromechanical systems (MEMS)* [Angell et al. 1983]. Já hoje, estas ideias estão se consolidando com os conceitos *Claytronics* e *Programmable Matter* [Goldstein et al. 2005]. A ideia dos conceitos consiste em combinar robôs de escala nanométrica ("*claytronic atoms*" ou "*catoms*") para formar outras máquinas. Os *catoms* eventualmente serão capazes se comunicar-se, mover-se e se conectar com outros *catoms* para apresentar diferentes formas e utilidades.

Outro perspectiva futura quanto ao *hardware* diz respeito ao *system on a chip (SoC)*. Neste conceito, a ideia é ter um circuito integrado que detém todos os componentes de um computador. Para o caso dos objetos inteligentes, este sistema integrará o rádio, o microcontrolador e sensores/atuadores. Atualmente isto é um problema devido ao componente rádio que requer sofisticado arranjo para ser posto em um único chip [Vasseur and Dunkels 2010].

Há poucos anos era possível apontar o fator custo como limitante para adoção e desenvolvimento dos objetos inteligentes. Entretanto, hoje é possível encontrar soluções IoT disponíveis no mercado com custo regular. No que tange desenvolver para IoT, é possível afirmar que o custo do *hardware* já é acessível diante de produtos como o Raspberry Pi, Arduino e similares permitem desde de a prototipagem até a produção final de soluções IoT a baixo custo, por exemplo, o Raspberry Pi 3 custa US$ 35.

1.3. *Softwares* e Ambientes de Desenvolvimentos para IoT

Esta seção aborda assuntos referentes a camada de *software* que orquestra a lógica de operação dos objetos inteligentes. O bloco básico de identificação e os mecanismos de comunição são os pontos centrais da seção. Além disso, serão pontuados os sistemas operacionais para IoT, bem como os principais ambientes desenvolvimento. O tópico a seguir introduz alguns argumentos e requisitos para *softwares* utilizados na IoT. Os desdobramentos dos pontos levantados serão objetos de análise ao longo da seção.

1.3.1. *Software* para rede de objetos inteligentes

RSSF foi objeto de grande análise por acadêmicos e industriais nos últimos anos como exposto na Seção 1.1. Em seguida, a conexão entre as RSSF e a Internet foi uma evolução natural. Entretanto, pesquisadores da área notam que uso dos protocolos da

Internet (TCP/IP), sem modificações, é impraticável nos dispositivos com recursos limitados, os quais são bastante utilizados na IoT de hoje. Para alcançar as demandas da IoT por escalabilidade e robustez, a experiência em RSSF mostra que são necessários algoritmos e protocolos especializados, por exemplo, protocolos que viabilizem processamento ao longo da rede (*in-network processing*) [Santos et al. 2015b, Fasolo et al. 2007] para mitigar as restrições impostas pelos dispositivos e prover escalabilidade e eficiência. Por outro lado, a arquitetura fim-a-fim da Internet não foi planejada para essas exigências (ex: dezenas de milhares de dispositivos em uma única sub-rede e extremas limitações de energia), portanto tal arquitetura necessita de ajustes para comportar essas novas demandas.

As RSSF e sua evolução, a IoT, cresceram em um ambiente com maior liberdade de desenvolvimento do que a Internet, por exemplo, não era preciso enfrentar questões de compatibilidade. Diante desta autonomia diversas ideias surgiram tanto ao nível de *software* quanto de *hardware*. Entretanto, muitas dessas ideias não frutificaram, pois não eram completamente interoperáveis com a Internet, o que diminui significativamente o impacto da técnica no mundo real. Essas novas ideias, que não empregam o uso da arquitetura da Internet, foram obrigadas a usar um *gateway* para intercambiar informações entre os objetos inteligentes e outros sistemas na Internet. A implementação de um *gateway* é, em geral, complexa e o seu gerenciamento é complicado, pois além de realizarem funções de tradução, possuem encargos semântico e o de provedor de serviços para a camada de aplicações. A complexidade destas funções torna o *gateway* um gargalo para IoT em dois sentidos. No primeiro, toda informação de e para a rede de objetos inteligentes transita através do *gateway*. No segundo, a lógica de operação da Internet diz que a inteligência do sistema deve ficar nas pontas [Saltzer et al. 1984], porém com o emprego dos *gateways* a inteligência da IoT fica no meio da rede, o que contradiz os princípios da arquitetura da Internet. Para tratar desses problemas a *Internet Engineering Task Force (IETF)* criou dois grupos para gerenciar, padronizar e levantar os requisitos para as redes de baixa potência (*Low-Power and Lossy Networks (LLN)*) relacionadas a seguir:

6LoWPAN: IETF atribuiu ao *IPv6 in Low-Power Wireless Personal Area Networks Working Group* a responsabilidade de padronizar o *Internet Protocol version 6* (IPv6) para redes que fazem uso de rádios sobre o padrão IEEE 802.15.4 que, por sua vez, especifica as regras das camada mais baixas (enlace e física) para redes sem fio pessoais de baixas potência de transmissão.

RoLL: *Routing over Low-Power and Lossy Links Working Group* é caracterizado pelo IETF como a organização que padronizará o protocolo de roteamento, que utilizará o IPv6 em dispositivos com limitações de recursos. O grupo já definiu o protocolo: *IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL)*. Este protocolo é o estado-da-arte em roteamento para IoT, o qual constrói uma topologia robusta com mínimo de estados necessários. O RPL será objeto de discussão mais adiante.

1.3.2. Arquitetura para IoT

Para conectar bilhões de objetos inteligentes à Internet se faz necessário um arquitetura flexível. Na literatura é possível notar uma variedade de propostas de arquitetura sofisticadas, que se baseiam nas necessidades da academia e industria [Al-Fuqaha et al. 2015].

O modelo básico de arquitetura apresenta 3 camadas, como exibido na Figura 1.5.

Figura 1.5. Arquitetura de 3 camadas para IoT.[17]

A primeira camada é a de objetos inteligentes ou camada de percepção. Esta camada representa os objetos físicos, os quais através de sensores coletam e processam informações. Na Camada de Rede, as abstrações das tecnologias de comunicação, serviços de gerenciamento, roteamento e identificação devem ser realizados. Logo acima, encontra-se a Camada de Aplicação, a qual é responsável por prover serviços para os clientes, por exemplo, uma aplicação pode medições de temperatura e umidade para clientes que requisitam estas informações.

1.3.3. IP para IoT

As redes de computadores foram criadas inicialmente para conectar as universidades, e não era esperado que ela se expandisse para os computadores pessoais. O endereçamento utilizado nestes período foi o IPv4, entretanto, não era imaginado que as redes cresceriam e poderiam conter dezenas de milhares de pontos finais em uma única sub-rede, tal como agora é previsto para a IoT (veja Seção 1.1). O crescimento da rede mundial de computadores, implica em menor quantidade de endereços IPv4 disponíveis. Para contornar a situação, o IETF criou o *Network Address Translator (NAT)*, capaz de mapear endereços IPs inválidos para endereços válidos. Com o NAT, uma rede local pode definir seus próprios endereços IPs, que não podem ser acessados por endereços IPs externos. Essa rede local contém apenas um endereço IP acessível por redes externas. Esta abordagem funciona, entretanto, ela cria outros problemas como, por exemplo, as redes não são mais conectadas diretamente o que fere o princípio fim-a-fim.

O IPv6 é uma abordagem mais eficaz para solucionar a escassez de endereços IPv4. Os 32 bits alocados originalmente para o protocolo IPv4 foram expandidos para 128 bits, viabilizando que qualquer dispositivo no mundo possua um IP válido na Internet, eliminando a necessidade do NAT. O IPv6 é um passo importante em direção à nova fase da Internet, chamada de Internet das Coisas. Na IoT, os elementos da rede são endereçados unicamente usando IPv6 e geralmente possuem o objetivo de enviar pequenas quantidades de dados contendo estados dos dispositivos. Para operar plenamente, o IPv6 requer 128 bits para endereços e MTU de 1280 bytes. Contudo, o IPv6 não se adaqua as limitações na maioria dos dispositivos empregados na IoT, por exemplo, dispositivos sobre o padrão IEEE 802.15.4 estão limitados a pacotes de 128 bytes. Para resolver estes problemas, IETF criou o 6LoWPAN [Kushalnagar et al. 2007].

[17] Imagem e texto baseados em [Khan et al. 2012, Al-Fuqaha et al. 2015].

6LoWPAN[18] é uma camada de adaptação primariamente desenvolvida para o padrão IEEE 802.15.4. A principal ideia é realizar a compressão de pacotes IPv6, permitindo a dispositivos com baixo poder computacional o uso do IPv6. A compressão do 6LoWPAN é possível através da utilização de informações de protocolos presentes em outras camadas. Por exemplo, o 6LoWPAN pode utilizar parte do endereço MAC do dispositivo para atribuir um endereço IPv6 para o objeto inteligente. Desta forma, o 6LoWPAN requer menos bits que o IPv6 para realizar o endereçamento.

O pesquisador Adam Dunkels[19] realizou uma série de contribuições na área da Internet das Coisas. Três de suas principais contribuições são as implementações da pilha TCP/IP para dispositivos de baixa potência. Estas implementações são conhecidas como: *low weight IP* (lwIP), o micro IP (μIP) e o sistema operacional para IoT Contiki.

A lwIP é uma implementação reduzida da pilha TCP/IP para sistemas embarcados[20]. A lwIP possui mais recursos do que a pilha μIP, discutida a seguir. Porém, a lwIP pode prover uma vazão maior. Atualmente esta pilha de protocolos é mantida por desenvolvedores espalhados pelo mundo[21]. A lwIP é utilizada por vários fabricantes de sistemas embarcados como, por exemplo, Xilinx e Altera. lwIP conta com os seguintes protocolos IP, ICMP, UDP, TCP, IGMP, ARP, PPPoS, PPPoE. As aplicações incluídas são: servidor HTTP, cliente DHCP, cliente DNS, ping, cliente SMTP e outros.

A μIP (Micro IP) é uma das menores pilhas TCP/IP completas do mundo[22]. Desenvolvida para pequenos microcontroladores (processadores de 8-bit ou 16-bit), sistemas onde, o tamanho do código e memória RAM disponível são escassos, μIP precisa de no máximo 5 kilobytes de espaço de código e de poucas centenas de bytes de RAM. Atualmente, a μIP foi portado para vários sistemas[23] e integra o Contiki.

1.3.4. Modelos de conectividade em redes de objetos inteligentes

Nesta subseção, serão abordados os modelos de conectividade para uma rede IPv6 de objetos inteligentes. Os modelos de conectividade serão classificados como um espectro que varia de uma rede de objetos inteligentes autônoma sem conexão com a Internet, até a, aqui considera, "autêntica" *Internet of Things* em que os objetos possuem acesso a Internet pública. As redes de objetos inteligentes serão parte de nossas vidas nos próximos anos. Por isso, entender as bases da IoT no que tange seus paradigmas de comunicação e modelos de conectividade é de grande importância, pois inevitavelmente estes dois quesitos serão as chaves para construir novas aplicações sobre a IoT. A Figura 1.6 destaca três modelos de conectividade de uma rede de objetos inteligentes. A seguir serão discutidas cada um dos modelos de conectividade:

- **Rede de Objetos Inteligentes Autônoma:** neste modelo a rede de objetos não possui conexão com a Internet pública. Existem diversos casos de uso para este

[18] https://tools.ietf.org/html/rfc4919
[19] http://www.dunkels.com/adam/
[20] http://www.ece.ualberta.ca/~cmpe401/docs/lwip.pdf
[21] http://savannah.nongnu.org/projects/lwip/
[22] https://github.com/adamdunkels/uip
[23] http://www.dunkels.com/adam/software.html
[25] Imagem e texto baseados em [Vasseur and Dunkels 2010].

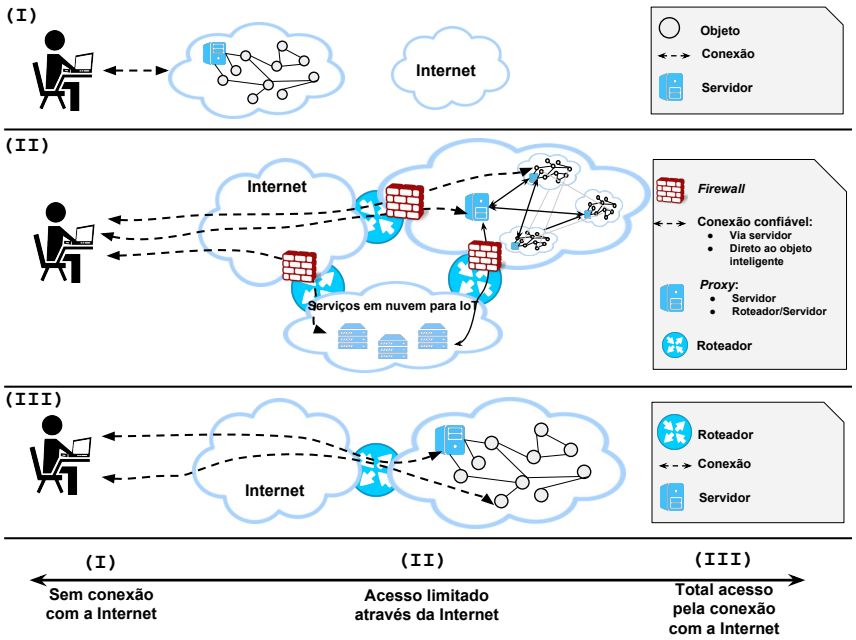


Figura 1.6. Modelos de conectividade dos objetos inteligentes. (I) Rede autônoma em que objetos inteligentes não possuem conexão com a Internet pública; (II) Rede de objetos inteligentes limitada, pois o acesso aos dispositivos é restrito; (III) IoT "autêntica" em que os objetos estão conetados à Internet pública.[25]

modelo, por exemplo, em uma rede de automação industrial (ex: usina nuclear) pode-se requerer uma rede de objetos conectados entre si, porém completamente inacessível através da Web, ou seja, o uso da rede é estritamente interno da corporação. A Figura 1.6 (I) apresenta uma abordam esquemática do modelo.

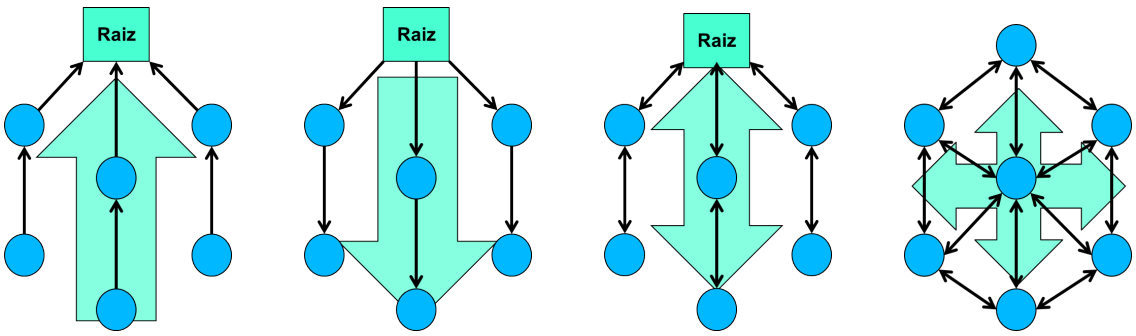
Uma questão pode ser levantada aqui é: "Neste modelo de conectividade é necessário que os objetos inteligentes sejam endereçados com IP?" Para responder a esta questão é preciso refletir sobre a experiência passada com o IP em redes convencionais. O IP apresenta diversas características que indicam um sim como resposta. Por exemplo, a arquitetura IP é interoperável, no sentido de que ele opera sobre a camada de enlance e física com diferentes características. Outro argumento a favor do sim, é que o IP é versátil e evolutivo, pois a arquitetura é baseada no princípio fim-a-fim, em que a inteligencia da rede (aplicações) residem nos pontos finais, enquanto a rede é mantida simples (somente encaminhando pacotes). Isto permitiu a evolução contínua do procolo ao longo do tempo. Tendo dito isto, é válido alegar que ao usar IP neste modelo, e os demais apresentados a seguir, a rede manterá compatibilidade com a arquitetura da Internet e estará de acordo com as tendências regidas pelos grupos RoLL e 6LoWPAN.

- **Internet Estendida:** o termo "Internet Estendida" refere-se ao posicionamento "central" deste modelo de conectividade em relação à rede de objetos autônoma e a "autêntica" IoT. Em outras palavras, este modelo apresenta as redes de objetos inteligentes parcialmente ou complementante conectadas com a Internet, porém

com apropriados requisitos de proteção e segurança [Vasseur and Dunkels 2010]. Aplicações para Cidades Inteligentes (*Smart Cities*) são exemplos desse modelo de conectividade. Estas aplicações produzirão informações úteis para que seus habitantes possam melhorar a qualidade de vida e tomar decisões importantes diariamente. Para viabilizar as cidades inteligentes, pode-se usar o modelo apresentado na Figura 1.6 (II), em que o acesso à rede de objetos se dá via *firewall*, que por sua vez tem como tarefa controlar o acesso de modo seguro às redes de objetos privadas. Deste modo, o modelo de conectividade "estende" a Internet por permitir o acesso aos objetos inteligentes que até então estavam isolados.

- **Internet das Coisas:** este modelo, no espectro considerado, é o extremo oposto ao modelo de rede de objetos autônoma. Na IoT, os objetos inteligentes estão verdadeiramente conectados à Internet. Deste modo, os objetos podem prover serviços tais como quaisquer outros dispositivos na Internet, por exemplo, um objeto inteligente pode ser um servidor *web*. Qualquer usuário da Internet, seja humano ou máquina, poderão ter acesso aos recursos dos objetos inteligentes. Como mostrado na Figura 1.6 (III), o acesso se dá ou conectando-se diretamente com o objeto ou através de *proxy*[26]. Estes servidores podem está localizados dentro ou fora da sub-rede de objetos inteligentes e possuem a tarefa de coletar informações dos dispositivos. Ao usar *proxy* tem-se que a Internet conecta o usuário ao servidor *proxy*, o qual está conectado aos dispositivos. Assim, técnicas de processamento dentro da rede (*in-networking processing*) podem ser utilizadas, na rede entre o *proxy* e os dispositivos, para preservar os recursos e manter o princípio fim-a-fim.

1.3.5. Paradigmas de comunicação para objetos inteligentes



(a) Muitos- para-Um (b) Um-para-Muitos (c) Mescla de (a) e (b) (d) Um-para-Um

Figura 1.7. Paradigmas de comunicação.

A IoT é uma evolução e combinação de diversas tecnologias como discutido na Seção 1.1. Neste sentido, existe interseção entre RSSF e IoT no que tange os paradigmas de comunicação. Esta seção serão abordados tais paradigmas, classificando-os em quatro categorias como mostrado na Figura 1.7. Os paradigmas diferem significativamente, sendo assim o modo de comunicação pode acarretar em maior ou menor impacto no uso

[26]Um *proxy* é um servidor (sistema de computador ou uma aplicação) que age como um intermediário para requisições de clientes solicitando recursos de outros servidores

dos recursos, em especial de memória, energia e na viabilidade de aplicações sobre a rede. A seguir cada um dos paradigmas será descrito, bem como exemplificado através de um protocolo que o implementa:

- **Muitos-para-Um (*Many-to-One*):** em que os objetos inteligentes reportam informações, as quais são coletadas por estações base (Raiz na Figura 1.7). Este paradigma é conhecido como coleta de dados, sendo o mais comum dos paradigmas, visto que atende às demandas de comunicação de muitas aplicações. Para realizar a coleta de dados cria-se uma árvore de roteamento com raiz nas estações base. Em geral, este paradigma não acarreta em grande consumo de memória e energia. Entretanto, aplicações que necessitam de confirmação de entrega de dados são inviabilizadas, pois não existem rotas reversas entre a raízes e os objetos na rede;

  O exemplo estado-da-arte no que tange a coleta de dados é o *Collection Tree Protocol (CTP)* [Gnawali et al. 2009]. O CTP emprega a métrica *Expected Transmission count (ETX)* [Javaid et al. 2009] e Trickle [Levis et al. 2004] para respectivamente criar rotas de alta qualidade e manter tais rotas a baixo custo.

- **Um-para-Muitos (*One-to-Many*):** é conhecido como disseminação de dados, o qual tem característica reversa ao do paradigma de coleta de dados. Na disseminação de dados, comumente as estações base enviam comandos para um ou vários objetos da rede. O intuito, em geral, é reconfigurar ou modificar parâmetros dos dispositivos na rede. Para realizar a disseminação de dados pode-se, por exemplo, efetuar inundações na rede para alcançar os dispositivos alvo. Entretanto, não é possível confirmar se os dados enviados foram entregues tal como ocorre com o CTP para a coleta de dados. O custo em número de mensagens também pode ser alto, caso sejam realizadas diversas inundações na rede.

  Deluge é um exemplo de protocolo de disseminação [Chlipala et al. 2004]. O protocolo é otimizado para disseminação para grandes quantidades de dados e faz isso utilizando a métrica ETX para encontrar rotas de alta qualidade;

- **Um-para-Muitos e vice-versa (*One-to-Many and Many-to-One*)**: é um paradigma que combina os dois anteriormente apresentados. Nesta abordagem, os objetos inteligentes podem se comunicar com as estações base e vice versa. Isto amplia a gama de aplicações antes não possíveis, tal como protocolos de transporte confiáveis. Entretanto, o paradigma necessita de algum custo de memória adicional para manter as rotas bi-direcionais.

  O *eXtend Collection Tree Protocol (XCTP)* [Santos et al. 2015a] é um exemplo desta categoria. Os autores argumentam que o XCTP é uma extensão do CTP e, por esse motivo, mantém todas as características do CTP ao mesmo tempo que o estende ao viabilizar rotas reversas entre o sorvedouro e os elementos da rede.

- **Um-para-Um (*Any-to-Any*):** esse paradigma é o mais geral possível, pois permite que quaisquer dois objetos inteligentes da rede se comuniquem. Por ser mais abrangente, esta abordagem é a mais complexa e geralmente exige maior quantidade de recursos, principalmente de armazenamento, pois se faz necessário manter rotas

para todos os dispositivos alcançáveis na rede. Por outro lado, não apresenta restrições significativas para as aplicações, exceto se os recursos computacionais dos dispositivos for bastante limitados.

O principal protocolo de roteamento da Internet das Coisas reside nesta categoria. O protocolo RPL, descito em detalhes a seguir, é flexível o suficiente para permitir rotas um-para-um, além de possibilitar que elementos de rede com diferentes capacidades sejam empregados para otimizar o armazenamento das rotas. Ainda existe um modo de operação em que, o RPL, opera sob o paradigma muitos-para-um, sendo portanto, um protocolo flexível no que tange as suas opções de operação.

1.3.6. IPv6 Routing Protocol for Low-Power and Lossy Networks

O *Routing Protocol for Low-Power and Lossy Networks* (RPL) é um protocolo de roteamento projetado para LLNs, projetado e padronizado pelo *ROLL Working Group in the IETF*. Ele foi projetado para ser o protocolo padrão que utiliza IPv6 para LLNs.

1.3.6.1. Grafo Acíclico Direcionado

Dispositivos de rede executando RPL estão conectados de maneira acíclica. Para esse propósito, um Grafo Acíclico Direcionado Orientado ao Destino (DODAG, do inglês *Destination-Oriented Directed Acyclic Graph*) é criado, como mostra a Figura 1.8. Cada nó mantém a melhor rota para a raiz do DODAG. Para encontrar a melhor rota os nós usam uma função objetivo (OF) que define a métrica de roteamento (ex: ETX da rota [Javaid et al. 2009]) a ser computada.

O RPL utiliza quatro tipos de mensagens de controle, elencadas a seguir, para manter e atualizar as rotas. O processo de roteamento inicia pela construção do DAG. A raiz anuncia informações sobre seu DODAG (de um único nó) para todos vizinhos alcançáveis. Em cada nível da árvore de roteamento os nós toam decisões sobre rotas baseadas na OF. Uma vez que o nó se junta ao DODAG, ele escolhe uma raiz como seu pai e calcula seu *rank*. O *rank* é uma métrica que indica as coordenadas do nó na hierarquia da rede [Vasseur et al. 2011]. Os demais nós irão repetir esse processo de seleção do pai e notificação as informações do DODAG para possíveis novos dispositivos. Quando esse processo se estabiliza, o roteamento de dados pode então começar. O processo descrito cria rotas ascendentes (dos nós para uma rais), para construir as rotas descendentes o RPL emite mensagens especiais discutidas a seguir.

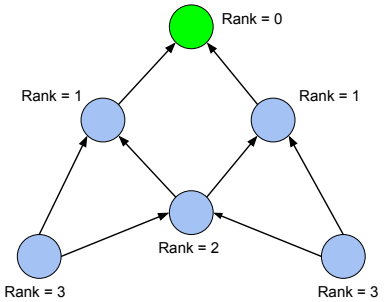


Figura 1.8. DODAG RPL[28].

[28] Imagem e texto basedos em [Vasseur et al. 2011, Cordero et al. 2013].

1.3.6.2. Mensagens

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

- DIO: mensagens desse tipo são utilizadas para anunciar um DODAG e suas características. Dessa forma, elas são usadas para a descoberta de DODAGs, sua formação e manutenção. O intervalo de envio de DIO é controlado de modo eficiente pelo algoritmo Trickle [Levis et al. 2004].

- DIS: esse tipo de mensagem é similar a mensagens de solicitação de rotas (RS) do IPv6, e são usadas para descobrir DODAGs na vizinhança e solicitar DIOs de nós vizinhos, sem possuir corpo de mensagem.

- DAO: mensagens DAO são utilizadas durante o processo de notificação de rotas descendentes. Elas são enviadas em sentido ascendente (dos nós que manifestam o desejo de receber mensagens para seus pais preferenciais) para propagar informações de destino ao longo do DODAG. Essas informações são utilizadas para o preenchimento das tabelas de roteamento descendente, que permitem o tráfego P2MP (ponto a multi-ponto) e P2P (ponto a ponto).

1.3.6.3. Rotas Descendentes

As rotas descendentes, da raiz para os nós, são ativadas por meio de mensagens DAO propagadas como unicast por meio dos pais em direção à raiz. Essas mensagens contêm informações sobre a quais prefixos pertencem a qual roteador RPL e quais prefixos podem ser alcançados através de qual roteador RPL. O protocolo especifica dois modos de operação para o roteamento descendente: *storing* e *non-storing*. Ambos os modos requerem a geração de mensagens DAO, que são enviadas e utilizadas de maneira diferente em cada modo de operação. Os dois modos de operação são descritos a seguir:

1. Modo *storing*: No modo de operação *storing*, cada roteador RPL deve armazenar rotas para seus destinos em um DODAG. Essas informações são repassadas dos nós para seus pais preferenciais. Isso faz com que, em nós mais próximos da raiz, o armazenamento necessário seja definido pelo número de destinos na rede. Com isso, a memória necessária em um nó próximo à raiz e um outro distante da raiz pode variar drasticamente, o que faz com que algum tipo de implementação e manutenção administrativa contínua nos dispositivos sejam necessárias, conforme a rede evolui [Clausen et al. 2013]. Entretanto, tal intervenção é inviável, devido ao perfil dinâmico da rede.

2. Modo *non-storing*: No modo *non-storing* cada nó gera e envia mensagens DAO para a raiz do DODAG. O intervalo no qual o DAO é enviado varia de acordo com a implementação. Entretanto, a especificação do protocolo sugere que esse intervalo seja inversamente proporcional à distância do nó a raiz. Dessa forma, um nó

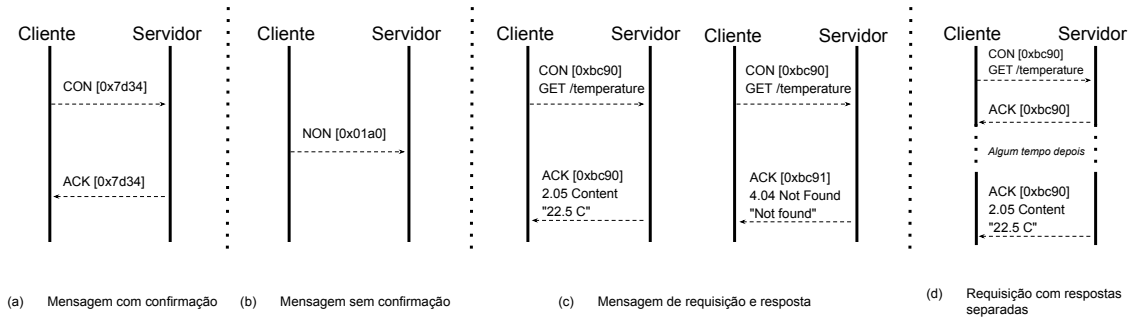


Figura 1.9. Tipos de mensagem do CoAP[31].

folha gera mais mensagens que um nó intermediário, por exemplo. Após coletar toda informação necessária, a raiz agrega essa informação. Se ela precisa enviar uma mensagem descendente na rede, ela deve utilizar um cabeçalho IPv6 para roteamento de origem (*source routing*). Dessa forma, os nós encaminham a mensagem até que ela alcance seu destino [Tsvetko 2011]. Ou seja, caso os nós não possuam capacidade de armazenamento para operarem no modo *storing*, a rede sofre maior risco de fragmentação e, portanto, perda de pacotes de dados, consumindo a capacidade da rede com o roteamento de origem [Clausen et al. 2013].

1.3.7. Protocolos da camada de aplicações para IoT

O famoso protocolo HTTP foi utilizado para acessar informações na Internet usando a estratégia requisição/resposta sobre o paradigma cliente/servidor. O HTTP foi desenvolvido para redes predominantemente formada por PCs. Diferentemente dos PCs, os dispositivos usados na IoT possui poder computacional restrito, o que limita a utilização do protocolo HTTP nesses dispositivos. Para resolver esse problema, dois protocolos da camada de aplicação desenvolvidos especificamente para recuperar informações de dispositivos com baixo poder computacional.

O *Constrained Application Protocol (CoAP)* é definido e mantido pelo *IETF Constrained RESTful Environments (CoRE) working group*[29]. O CoAP define uma forma de transferir dados tal como é feito através do *REpresentational State Transfer (REST)* e, para tanto, utiliza funcionalidades similares ao do HTTP tais como: *GET*, *POST*, *PUT*, *DELETE*. REST permite que clientes e servidores acesse ou consumam serviços *web* de maneira fácil usando *Uniform Resource Identifiers (URIs)*. O CoAP diferente do REST por usar o protocolo UDP, o que o coloca como mais adequado para aplicações em IoT.

O CoAP apresenta duas camadas em sua arquitetura interna, objetivando de permitir que dispositivos de baixa potência possam interagir através de RESTfull. A primeira camada implementa implementa os mecanismos de requisição/resposta. A segunda, detecta mensagens duplicadas e fornece comunicação confiável sobre o UDP. O CoAP utiliza quatro tipos de mensagens: *confirmable*, *non-confirmable*, *reset* e *acknowl edgement*. Estas imagens estão exibidas na Figura 1.9. A confiabilidade do protocolo CoAP é implementada ao combinar as mensagens *confirmable* e *non-confirmable* sobre o UDP.

[29] https://datatracker.ietf.org/doc/charter-ietf-core/

[31] Fonte: https://tools.ietf.org/html/rfc7252

CoAP alcança toda sua funcionalidade quando interconectado com o HTTP. As URIs CoAP são utilizadas da seguinte forma coap://URI. Existem algumas extensões que são utilizadas para integrar dispositivos CoAP com navegadores *web*, por exemplo, a Copper para Firefox. Outras informações podem ser encontradas na RFC 7252 e o conjunto de implementações existentes podem ser encontradas no site de um dos autores do CoAP[32].

O *Message Queue Telemetry Transport* (MQTT) é um protocolo da camada de aplicação para IoT. O MQTT foi projetado para dispositivos extremamente limitados e utiliza a estratégia de *publish/subscribe* para transferir mensagens. O principal objetivo do MQTT é minimizar o uso de largura de banda da rede e recursos dos dispositivos. Além disso, o MQTT prover mecanismos para a garantia de entrega de mensagens. MQTT utiliza o TCP/IP como protocolos das camada de transporte e rede respectivamente. O cabeçalho do protocolo MQTT pode assumir tamanho fixo ou variável. O tamanho fixo possui 2 bytes. Um exemplo de uma implementação *open source* do MQTT é o Mosquitto[35].

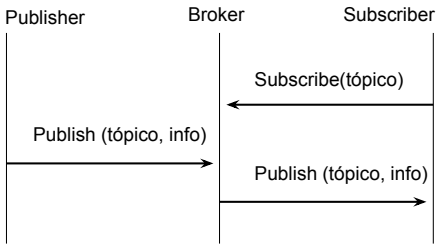


Figura 1.10. Processo *publish/subscribe*[34].

O MQTT consiste de três componentes básicos: o *subscriber*, *o publisher* e o *broker*. A Figura 1.10, mostra a ordem de operação do MQTT. Inicialmente dispositivos se registram (*subscribe*) a um *broker* para obter informações sobre dados específicos, para que o *broker* os avise sempre que publicadores (*publishers*) publicarem os dados de interesse. Os dispositivos inteligentes (*publishers*) transmitem informações para os *subscriber* através do *broker*.

1.3.8. Ambientes de desenvolvimento

Assim como softwares para computadores de propósito geral, o software que roda no microcontrolador dentro de um objeto inteligente também é escrito em uma linguagem de programação e compilado para o código de máquina para o microcontrolador. O código de máquina é escrito na ROM do microcontrolador e quando o objeto inteligente é ligado, esse software é executado [Vasseur and Dunkels 2010].

1.3.8.1. Sistemas Operacionais

Assim como os computadores domésticos, os objetos inteligentes também utilizam sistemas operacionais. Entretanto, como os objetos inteligentes possuem recursos físicos limitados, devido ao pequeno tamanho, baixo custo e baixo consumo de energia, os sistemas operacionais para esses objetos devem ser muito menores e consumir menos recursos. Nesta Seção, serão descritos brevemente os dois principais sistemas operacio-

[32] http://coap.technology
[34] Imagem baseada no conteúdo do site: http://mqtt.org/.
[35] http://mosquitto.org

nais para objetos inteligentes: Contiki e TinyOS, além do sistema operacional Android que opera em grande parte dos dispositivos de sensoriamento participativo (*smartphones*) e algumas variações do sistema Linux orientadas a Internet das coisas . Todos esses sistemas operacionais são código aberto e os respectivos códigos fonte podem ser encontrados na Web. Ao final, a Tabela 1.2 apresenta uma comparação entre os principais sistemas apresentados.

1. **Contiki** [36]**:** Contiki é um sistema operacional leve de código aberto para sistemas embarcados de rede, que fornece mecanismos para o desenvolvimento de *softwares* para IoT e mecanismos de comunicação que permitem a comunicação dos dispositivos inteligentes. Além disso, o Contiki também fornece bibliotecas para a alocação de memória, abstrações de comunicação e mecanismos de redes de rádios de baixa potência. O Contiki foi o primeiro sistema operacional para objetos inteligentes a fornecer comunicação IP com a pilha μIP TCP/IP e, em 2008, o sistema incorporou o μIPv6, a menor pilha IPv6 do mundo. Por utilizar IP, o Contiki pode se comunicar diretamente com outros aplicativos baseados em IP e serviços de *web* [Vasseur and Dunkels 2010]. Tanto o sistema operacional quanto suas aplicações são implementados na linguagem C, o que faz o sistema ser altamente portável [Dunkels et al. 2004].

2. **TinyOS**[37]**:** Assim como o Contiki, o TinyOS é um sistema operacional de código aberto para redes de sensores e objetos inteligentes. Um programa do TinyOS é descrito em [Levis et al. 2005] como um grafo de componentes, cada um sendo uma entidade computacional independente que expõe uma ou mais interfaces. Os componentes podem ser chamados comandos (requisições a um componente para executar algum serviço), eventos (sinalizam a finalização desse serviço) ou tarefas (usadas para expressar a concorrência intra-componente). TinyOS possui um modelo de programação baseado em componentes, codificado pela linguagem NesC, um dialeto do C. O modelo de programação fornecido pela linguagem NesC se baseia em componentes que encapsulam um conjunto específico de serviços, e se conectam (comunicam) por meio de interfaces.

3. **Android**[38]**:** Android é uma plataforma de *software* de código aberto para dispositivos móveis que inclui um sistema operacional, *middleware* e aplicativos[39]. Os vários componentes do sistema operacional são projetados como uma pilha, com o núcleo do Linux na camada mais baixa. Acima da camada do Linux, estão as bibliotecas nativas do Android, o que permite ao dispositivo manusear diferentes tipos de dados. Na mesma camada que as bibliotecas está também o *Android Runtime*, que possui um tipo de máquina virtual Java utilizada para a execução de aplicativos no dispositivo Android. A próxima camada está relacionada com o *Framework* de

[36] https://github.com/contiki-os

[37] https://github.com/tinyos

[38] O código do Android é disponibilizado pela Google sob licença de código aberto, apesar da maior parte dos dispositivos ser lançada com uma combinação de software livre e software privado: *https://github.com/android*

[39] http://developer.android.com/guide/basics/what-is-android

Sistema	Min. RAM	Min. ROM	Linguagem
Contiki	<2kB	<30kB	C
TinyOS	<1kB	<4KB	nesC e oTcl
RIOT	$\sim$ 1.5kB	$\sim$ 5kB	C e C++
Snappy	128 MB	–	Python, C/C++, Node JS e outras
Raspbian	256MB	–	Python, C/C++, Node JS e outras

Tabela 1.2. Comparativo entre os principais Sistemas Operacionais para IoT.

aplicação, que fornece vários serviços de alto nível ou APIs para os desenvolvedores de aplicativos. Por fim, no topo da pilha, está a camada de aplicação.

4. **Linux:** Com a popularização e o crescimento no desenvolvimento e pesquisa em IoT, alguns sistemas operacionais baseados em Linux foram desenvolvidos. o RIOT [40] é um sistema gratuito de código aberto [41] desenvolvido por uma comunidade base formada por empresas, acadêmicos e amadores, distribuídos em todo o mundo. Essa plataforma baseada em Linux roda em 8, 16 ou 32-bits e possui suporte às linguagens C e C++. Além disso, possui suporte para IoT com implementações 6LoWPAN, IPv6, RPL, e UDP. O Ubuntu também possui sua versão IoT, chamada Ubuntu Core ou Snappy [42]. Essa versão é reduzida em comparação ao Ubuntu Desktop, uma vez que exige apenas 128 MB de memória RAM e um processador de 600 Mhz. O desenvolvimento pode ser feito em diversas linguagens, como o de uma aplicação Linux comum. Raspian [43] é um sistema operacional gratuito baseado no Debian e otimizado para o hardware do Raspberry Pi. Oferece mais de 35.000 pacotes *deb* de software, que estão pré-compilados, para serem facilmente instalados no Raspberry Pi. O sistema está disponível em três versões: Raspbian Wheezy, DRaspbian Jessie e Raspbian Jessie Lite.

1.3.8.2. Emuladores e Simuladores

Durante a fase de avaliação de arquiteturas e protocolos para redes de computadores, simulações e emulações são muito úteis. Estes ambientes de desenvolvimento permitem modelar uma rede de computadores arbitrária especificando o comportamento dos elementos da rede, bem como os enlaces de comunicação. Esta seção apresentará os principais simuladores e emuladores de redes de computadores que possuem suporte para Internet das Coisas.

Essencialmente existem diferenças entre emuladores e simuladores. Emulador é um sistema de *hardware* ou *software* que permite que um sistema de computador (chamado de *host*) se comporte como um outro sistema de computador (chamado de *guest*).

[40] http://www.riot-os.org/
[41] https://github.com/RIOT-OS/RIOT
[42] http://www.ubuntu.com/internet-of-things
[43] https://www.raspbian.org/

Ou seja, um sistema que se comporta exatamente como o sistema de *guest* e age de acordo com todas as regras do sistema a ser emulado, mas operando em um ambiente diferente do ambiente do sistema emulado original. Um simulador, por sua vez, é um sistema de hardware ou *software* que permite que um sistema de computador (*host*) se comporte como um outro sistema de computador (*guest*), mas é implementado de uma forma totalmente diferente. Ou seja, um sistema projetado para recriar a operação ou o comportamento do sistema convidado, com princípios fundamentais podendo ser os mesmos que os originais ou diferentes [Bagula and Erasmus 2015]. Abaixo são listados e caracterizados os principais ambientes de simulação e emulação. Ao final, a Tabela 1.3 apresenta uma comparação entre os simuladores/emuladores para IoT.

- **ns-2/ns-3:** Ns-2 é um simulador de eventos discretos para rede de computadores de código aberto. As simulações para ns-2 são compostas por código em C++ e scrips oTcl. O código é utilizado para modelar o comportamento dos nós, enquanto o script controlam a simulação e especificam aspectos adicionais, como a topologia da rede. Ns-3 também é um simulador de eventos discretos, mas não é uma extensão de seu antecessor, ns-2. Assim como o ns-2, o ns-3 adota a linguagem C++ para a implementação dos códigos, mas não utiliza mais scripts oTcl. Com isso, as simulações podem ser implementadas em C++, com partes opcionais implementadas usando Python [Weingärtner et al. 2009].

- **Cooja:** Cooja (**Co**ntiki **OS** **Ja**va) é um simulador de aplicações do sistema operacional Contiki, desenvolvido da linguagem Java. As simulações no Cooja consistem em nós sendo simulados, cada nó possuindo seu tipo de nó, sua própria memória e um número de interfaces [Österlind and of Computer Science 2006]. Um nó simulado no Cooja é um sistema Contiki real compilado e executado. Esse sistema é controlado e analisado pelo Cooja [44]. Cooja possui uma interface para analisar e interagir com os nós simulados. Essa interface facilita o trabalho e a visualização dos nós, além de ser possível alcançar um comportamento de simulação altamente personalizado.

- **Tossim:** Tossim é o simulador de eventos discretos do sistema operacional TinyOS. Ao invés de compilar uma aplicação TinyOS para um nó sensor, os usuários podem compilá-lo no TOSSIM, que é executado em um PC. O Tossim é capaz de simular milhares de nós sensores simultaneamente e cada nó na simulação executa o mesmo programa TinyOS. O simulador se concentra em simular o TinyOS e sua execução, ao invés de simular o mundo real. Por isso, apesar de poder ser usado para entender as causas do comportamento observado no mundo real, não é capaz de capturar todos eles, não deve ser usado para as avaliações absolutas [Levis and Lee 2010]. Na abstração de rede sem fio do Tossim, a rede é um grafo orientado no qual cada vértice é um nó e cada aresta tem uma probabilidade de erro de bit. O simulador também fornece um conjunto de serviços de comunicação para interagir com aplicativos externos [Levis et al. 2003].

- **OMNet++/Castalia:** OMNeT++ é um simulador de eventos discretos baseado em C++ para modelar redes de comunicação, multiprocessadores e outros sistemas dis-

[44] https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja

Nome	Suporte GUI	Licença	Linguagem	Sistema Operacional
ns-2	Não	Código aberto	C++ e oTcl	GNU/Linux, FreeBSD, Mac OS e Windows
ns-3	Limitado	Código aberto	C++ e python	GNU/Linux, FreeBSD, Mac OS e Windows
Cooja	Sim	Código aberto	C	Linux
Tossim	Limitado	Código aberto	nesC e python	Linux ou Cygwin no Windows
OMNet++	Sim	Comercial e código aberto	C++	Linux, Mac OS e Windows
Castalia	Sim	Código aberto	C++	Linux e Windows
Sinalgo	Sim	Código aberto	Java	Linux, Mac OS e Windows

Tabela 1.3. Comparativo entre os principais simuladores/emuladores para IoT.

tribuídos ou paralelos [Varga and Hornig 2008]. Em vez de fornecer diretamente os componentes de simulação para redes de computadores, o OMNet++ fornece o maquinário e as ferramentas básicas para escrever tais simulações. OMNeT++ oferece uma IDE baseada no Eclipse, um ambiente de execução gráfica e uma série de outras ferramentas. Existem extensões para simulação em tempo real, emulação de rede, integração de banco de dados, integração de sistemas, e várias outras funções [Varga et al. 2001]. Castalia é um simulador para RSSF e outras redes de dispositivos embarcados de baixa potência. Ele é baseado na plataforma do OMNeT++ e pode ser usado por pesquisadores e desenvolvedores que querem testar seus algoritmos distribuídos e/ou protocolos em modelos realistas de canais sem fio e rádio [Boulis et al. 2011].

- **Sinalgo:** Sinalgo (**Si**mulator for **N**etwork **Algo**rithms) é um *framework* de simulação para testar e validar algoritmos de rede. Diferente da maioria dos outros simuladores de rede, que passam a maior parte do tempo simulando as diferentes camadas da pilha de rede, o Sinalgo foca na verificação de algoritmos de rede. Ele oferece uma vazão da troca de mensagens na rede, sendo capaz de capturar bem a visão dos dispositivos reais de rede. Apesar de ter sido desenvolvido para simulação de redes sem fio, não é limitado para apenas essa tecnologia. O Sinalgo utiliza a linguagem JAVA, o que torna o desenvolvimento mais fácil e rápido, se comparado com as linguagens específicas do hardware, além de facilitar o processo de *debug* [45].

1.3.8.3. Testbeds

Testbed é uma plataforma para implantar aplicações em um contexto real, ou seja, utilizando hardware real em larga escala, apropriada para a gestão de experimentação. Atualmente, existem vários *testbeds* em todo o mundo, que permitem a experimentação

[45] http://dcg.ethz.ch/projects/sinalgo/

mais rápida, com tamanho, hardware e topologias diferentes. A lista a seguir apresenta os principais *testbeds* para a experimentação em IoT.

- **WHY-NET:** WHY-NET é um *testbed* escalável para Redes Sem-Fio Móveis. Ele investiga diferentes tipos de tecnologias sem fio, individualmente e em conjunto. As tecnologias sem fio diferentes incluem, Redes de Sensores e Antenas inteligentes [Patel and Rutvij H. 2015].

- **ORBIT:** ORBIT consiste de 400 nós de rádio que são fixos. Cada nó físico está logicamente ligado a um nó virtual em uma rede. Os nós de rádio possuem duas interfaces 802.11x e *Bluetooth*. As medições podem ser realizados no nível físico, MAC e rede [Patel and Rutvij H. 2015].

- **MiNT:** Neste *testbed*, nó de rádio 802.11 são aplicados em robôs de controle remoto. Para evitar fontes de ruído, o *testbed* é configurado em uma única sala de laboratório. MiNT suporta reconfiguração topologia completa e mobilidade de nó irrestrita [Patel and Rutvij H. 2015].

- **IoT-LAB:** IoT-LAB oferece um mecanismo de infra-estrutura de grande escala adequado para testar dispositivos sensores sem fios pequenos e objetos heterogêneos comunicando. Além disso, ele oferece ferramentas web para desenvolvimento de aplicações, juntamente com o acesso de linha de comando direto à plataforma. *Firmwares* de nós sensores podem ser construídas a partir da fonte e implantado em nós reservados, atividade de aplicação pode ser controlada e monitorada, o consumo de energia ou interferência de rádio podem ser medidos usando as ferramentas fornecidas [46].

- **Indriya:** Indriya é um *testbed* rede de sensores sem fio, que possui 127 motes TelosB. Mais de 50% dos motes são equipados com diferentes módulos de sensores, incluindo infravermelho passivo (PIR), magnetômetro, acelerômetro, etc. Indriya usa o software de interface do usuário do Motelab, que fornece acesso baseado na web para os nós do *testbed*. O *testbed* está localizado na Universidade Nacional de Singapura [Doddavenkatappa et al. 2012].

1.3.9. Desafios e questões de pesquisa

Computadores pessoais e dispositivos móveis tais como *tablets ou smartphones* possuem a característica de se comunicarem facilmente a Internet. Isto se deve aos protocolos de comunicação utilizados. Em alguns casos é utilizado o padrão Ethernet, bastante utilizado em dispositivos que não necessitam de mobilidade. Para os dispositivos móveis, o Wi-Fi e a tecnologia celular (3G/4G) estão bastante difundidos e com grande área de cobertura, tornando o acesso a Internet universal e ubíquo. A próxima geração de dispositivos, representada pelos objetos inteligentes, os quais enfrentam limitações de recursos computacionais e energia. Devido a estas restrições, padrões de tecnologias de comunicação amplamente difundidos (Ethernet, Wi-Fi e 3G/4G) não são adequados,

[46] https://www.iot-lab.info

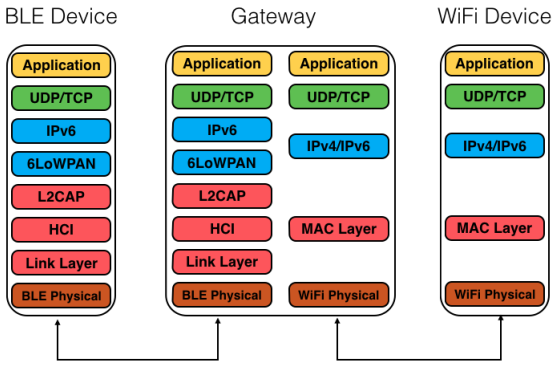


Figura 1.11. Pilha de protocolos de um *gateway* BLE-WiFi.

pois ou não permitem mobilidade ou apresentam grande consumo de energia. Para resolver estas questões são adotados protocolos otimizados para LLN como, por exemplo, o BLE, o ZigBee e outros. Contudo, no estado da arte atual, é necessário a presença de um *gateways*[47]. Neste caso, o *gateway* é responsável por realizar a conexão entre a Internet e os dispositivos dotados de tecnologias de comunicação heterogêneas para LLNs. Diante desta tarefa, o *gateway* torna-se complexo o que pode ser um gargalo para a IoT [Zachariah et al. 2015a].

A complexidade inerente do *gateway* existe devido à necessidade da tradução de protocolos. Os protocolos de rede usam, não apenas formato de pacotes distintos, como também diferentes mecanismos e lógica para roteamento, qualidade de serviço (QoS), recuperação de erros, modelos de transporte, gerenciamento, solução de problemas e segurança, e todos esses quesitos são considerados na tradução dos protocolos. Além disso, *gateway* de tradução de protocolo não escalam e se tornam gargalos da rede. Esses *gateway* também impactam na confiabilidade geral da rede, pois são pontos únicos de falha [Vasseur and Dunkels 2010].

A Figura 1.11, exibe um exemplo de *gateway* utilizando dois tipos de interface de comunicação, no caso BLE e Wi-Fi. A esquerda é representado o dispositivo inteligente e toda a sua pilha de protocolos da camada física até a camada de aplicação. Igualmente, o *gateway* precisa da mesma pilha implementada em sua arquitetura, caso seja adotada alguma camada de adaptação como (ex: 6LoWPAN), o *gateway* participa da realização da compressão de endereços, reduzindo a quantidade de bytes trafegados na rede. A direita é representado um dispositivo utilizando Wi-Fi como tecnologia de comunicação, junto com a sua pilha de protocolos. A Figura 1.11, representa um cenário de troca de mensagens em uma rede heterogênea, deixando a operação transparente para o usuário.

Segundo [Zachariah et al. 2015b], o problema do *gateway* existe pois atualmente estes elementos unem as tarefas de conectividade de rede, processamento interno da rede e funções de interface de usuário. Para solucionar tal problema, os autores apresentam

[47]O *Gateway* é o dispositivo intermediário geralmente destinado a interligar redes, separar domínios de colisão ou traduzir protocolos.

uma proposta de solução que separa essas funções, com o objetivo de melhorar a conectividade dos dispositivos da IoT. Para isso, os autores sugerem a utilização de *smartphones* como *gateways*, devido a sua conexão à Internet quase sempre constante, mobilidade e ubiquidade, além de ditarem o que os dispositivos de IoT devem usar, baseado no que está disponível nesses celulares. Além disso, ao invés de utilizar Wi-Fi para conectar os dispositivos IoT os autores sugerem a utilização do *Bluetooth Low Energy* (BLE). Isso porque o Wi-Fi exige um consumo de energia alto, tornando-se inadequado para aplicações de baixa potência. O BLE, por sua vez, é capaz de manter o mesmo raio de comunicação do *Bluetooth* clássico, mas com consumo e custo reduzidos.

1.4. IoT na prática

As seções anteriores trouxeram uma visão geral sobre as bases da IoT. Foram discutidas diversas características dos objetos inteligentes permeando particularidades teóricas e artefatos existentes já empregados na IoT. Já esta seção traz uma perspectiva prática na IoT. Diante do que já foi discutido, o leitor está apto a por em prática os conceitos vistos. Os exercícios práticos visam que o leitor consolide e associe os conceitos teóricos e artefatos previamente discutidos. Além disso, uma das práticas servirá de âncora para assuntos das próximas seções, vinculando as redes de objetos inteligentes, até aqui apresentadas, com os desafios e próximos passos em relação ao futuro da IoT.

Os exemplos apresentados a seguir foram extraídos do conjunto de exemplos do sistema operacional Contiki. Presume-se que o leitor já tenha o Contiki instalado e operacional[48]. Todos os exemplos são de código livre e hospedados no repositório oficial do Contiki. Inicialmente será exemplificado como conectar os objetos inteligentes utilizando IPv6[49], em seguida, será pleiteado o Erbium que é uma das implementações do CoAP[50] para redes de objetos de baixa potência Contiki.

Os experimentos serão apresentados visam atender o maior publico possível. Para isto, ao longo do texto a prática é exemplificada através do uso de simulador. Entretanto, o minicurso também apresenta uma página web [51] e lá existem materiais extra construídos por nós autores ou referências de livre acesso. No site encontra-se vídeo tutoriais, textos, referências e outros conteúdos relacionados.

1.4.1. Rede objetos inteligentes IPv6

No primeiro experimento prático, será criada uma rede IPv6 de objetos inteligentes utilizando Cooja (veja a Seção 1.3.8). O Contiki oferece uma implementação do 6LoWPAN em conjunto com o protocolo de roteamento RPL [Hui 2012]. Além disso, o sistema operacional também oferece suporte a pilha de protocolos μIP TCP/IP (veja Seção 1.3.3). Para começar, inicie o Cooja através do atalho da área de trabalho ou execute o comando abaixo e, em seguida, crie uma nova simulação:

[48] http://www.contiki-os.org/start.html
[49] https://github.com/contiki-os/contiki/tree/master/examples/ipv6/rpl-border-router
[50] https://github.com/contiki-os/contiki/tree/master/examples/er-rest-example
[51] **http://homepages.dcc.ufmg.br/~bruno.ps/iot-tp-sbrc-2016/**

Inicie o Cooja executando os comandos abaixo:

```
$ cd <DIR>+contiki/tools/cooja/
$ ant run
```

Dando seguimento à prática, dispositivos *Tmote Sky* serão emulados. Um dos *motes* servirá como roteador de borda da rede de objetos IPv6. O roteador de borda é o dispositivo responsável por configurar um prefixo de rede e iniciar a construção da árvore de roteamento do RPL. Em outras palavras, o roteador de borda nada mais é que a raiz da árvore de roteamento e interlocutor entre a rede de objetos e a rede externa.

O código do roteador de borda está localizado em:

```
<DIR>+contiki/examples/ipv6/rpl-border-router/border-router.c
```

Com o Cooja aberto, vá até a aba de criação de *mote* e adicione um *Sky mote*. Na tela seguinte, compile e carregue, como *firmeware*, o código do roteador de borda. Finalmente adicione **1 *mote*** deste tipo.

Após configurar o roteador de borda, a próxima etapa é povoar a rede com objetos inteligentes. O código *sky-websense.c* ajudará nesta fase.

O código do *sky-websense.c* está localizado em:

```
<DIR>+contiki/examples/ipv6/sky-websense/sky-websense.c
```

Este código é uma aplicação que produz "dados sensoreados" e prover acesso a estas informações via *webserver*. Adicione alguns[52] *motes* desse tipo. É recomendável que o o leitor investigue o código *sky-websense.c* para entender o que ele faz. Retorne para o Cooja. Na aba chamada *"Network"*, localize o roteador de borda e no seu menu de contexto selecione a opção mostrada abaixo. O Cooja informará que o roteador de borda criou um *socket* e está escutando na porta local `60001`. Em seguida inicialize a simulação.

No Menu-Contexto do Roteador de Borda selecione:

```
"Mote tools for sky/Serial socket (SERVER)"
```

Abra um novo terminal e execute os seguintes comandos:

Comandos para executar o programa `tunslip6`

```
$ cd <DIR>+contiki/examples/ipv6/rpl-border-router/
$ make connect-router-cooja TARGET=sky
```

Estes comandos acabam por executar o programa `tunslip6`. O programa configura uma interface na pilha de protocolos do Linux, em seguida, conecta a interface ao *socket* em que o roteador de borda está escutando. O `tunslip6` fará com que todo o tráfego endereçado ao prefixo **aaaa:**[53] seja direcionado para a interface do Cooja.

[52] Adicione 2 ou 5 *motes* para manter a carga de processamento e consumo de memória baixos.

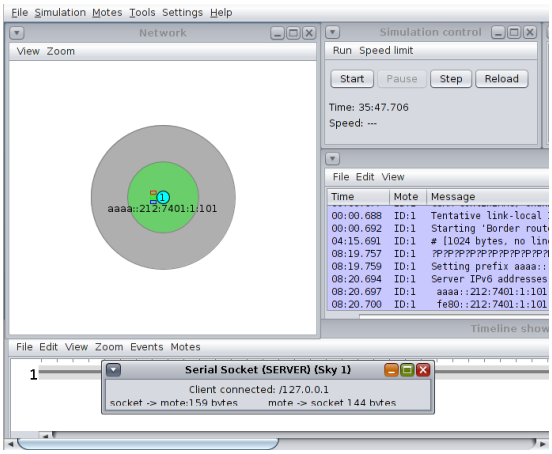
[53] Vale ressaltar que os endereços aqui apresentados serão expressos em modo comprimido. Por exemplo, `aaaa:0000:0000:0000:0212:7401:0001:0101` é o mesmo que `aaaa::212:7401:1:101`

Terminal 2 – `tunslip6`

```
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...
     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
     ...

Address:aaaa::1 => aaaa:0000:0000:0000
Got configuration message of type P
Setting prefix aaaa::
Server IPv6 addresses:
 aaaa::212:7401:1:101
 fe80::212:7401:1:101
```



O `tunslip6` apresentará uma saída semelhante a mostrada na caixa de título Terminal 2 e o Cooja apresentará algo como mostrado na figura ao lado[54]. Agora já é possível verificar se os *Tmotes Sky* emulados estão acessíveis através de ferramentas como o `ping6`.

Realize testes com os seguintes comandos:

`$ping6 aaaa::212:7401:1:101` (Roteador de borda)
`$ping6 aaaa::212:7402:2:202` (*Tmote* rodando *sky-websense*)

Em seu navegador acesse o endereço do roteador de borda http://[aaaa::212:7401:1:101]/

A rede de exemplo possui 3 *motes* e apresenta topologia exibida na figura abaixo, em que ~:101 é o roteador de borda.

O Roteador de Borda responderá com:

```
Neighbors

fe80::212:7402:2:202

Routes

aaaa::212:7402:2:202/128 via fe80::212:7402:2:202 16711412s
aaaa::212:7403:3:303/128 via fe80::212:7402:2:202 16711418s
```

A resposta a requisição feita ao roteador de borda conterá duas partes. A primeira, exibe a tabela de vizinhança, ou seja, os nós diretamente ligados na árvore de roteamento do RPL (*Neighbor Table*). Na segunda, são listados os nós alcançáveis (*Routes*) e o próximo salto na rota até aquele destinatário.

Agora acesse, pelo navegador, os recursos (luminosidade e temperatura) providos pelos *Tmotes* rodado *sky-websense* como mostrado abaixo:

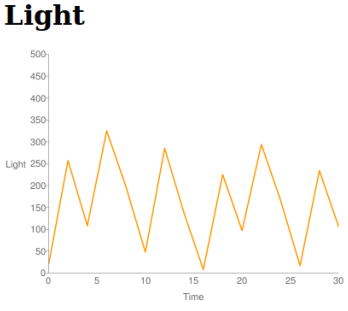
Acesse os *Tmotes* pelo browser com:

```
http://[IPv6 do Tmote]/
http://[IPv6 do Tmote]/l
http://[IPv6 do Tmote]/t
```

Exemplo de requisição:

```
http://[aaaa::212:7403:3:303]/l
```

Exemplo de resposta:
`http://[aaaa::212:7403:3:303]/l`



[54]Note que os *motes* executando *websense* não foram exibidos.

1.4.2. Erbium (Er) REST: uma implementação CoAP no Contiki-OS

Esta prática abordará o uso de *Representational State Transfer (REST)*, em português Transferência de Estado Representacional. Para tanto, será utilizado o Erbium (Er), uma implementação do *IETF Constrained Application Protocol (CoAP)*[55]. O Er foi projetado para operar em dispositivos de baixa potência [Kovatsch et al. 2011] e tem código livre disponível junto com o sistema operacional Contiki. Para iniciar será feita uma breve revisão da implementação e uso do CoAP no Contiki.

1.4.2.1. CoAP API no Contiki

No CoAP, os serviços disponibilizados pelo servidor são vistos como recursos, os quais são identificados por URIs únicas. O CoAP oferece diferentes métodos para realizar operações básicas CRUD sendo eles: **POST** para criar um recurso; **GET** para recuperar um recurso; **PUT** para atualizar algum recurso e; **DELETE** para apagar um recurso.

A implementação do CoAP no Contiki está localizada no diretório:

```
<DIR>+contiki/apps/er-coap<version>
```

Para cada recurso disponibilizado no servidor usando CoAP existe uma função (*handle function*), a qual a aplicação REST chama sempre que ocorre uma requisição de um cliente. Com a *handlefunction*, o servidor é capaz de tratar cada requisição e responder adequadamente com o conteúdo do recurso solicitado.

As macros[56] a seguir são providas pela implementação do CoAP/Erbium do Contiki para facilitar a criação de novos recursos para o servidor:

- **Normal Resource**: este tipo de recurso serve de base para todos as demais macros. Uma *Normal Resource* associa uma URI a uma *handle function*.

```
1 #define RESOURCE(name, attributes, get_handler, post_handler, put_handler,
      delete_handler)
```

- **Parent Resource**: destinado a gerenciar diversos sub-recursos de uma URI. Por exemplo, a URI test/parent/<sub-recursos>.

```
1 #define PARENT_RESOURCE(name, attributes, get_handler, post_handler, put_handler,
      delete_handler)
```

- **Separate Resource**: esta macro é bastante utilizada quando é necessário enviar informações em partes. Por exemplo, quando se tem algum arquivo na memória do dispositivo. Deste modo, o arquivo pode ser enviado em partes separadas para o cliente que o solicitou.

```
1 #define SEPARATE_RESOURCE(name, attributes, get_handler, post_handler, put_handler
      , delete_handler, resume_handler)
```

[55] CoAP RTF 7252 http://tools.ietf.org/html/rfc7252.

[56] Mais detalhes sobre as macros, atributos e estruturas são encontrados em: https://github.com/contiki-os/contiki/tree/master/apps/er-coap

- **Event Resource** e **Periodic Resource**: no primeiro, a ideia é que quando um evento ocorra o dispositivo envie uma mensagem para algum cliente que esteja "observando" aquele recurso, por exemplo, quando um botão no dispositivo é pressionado envia-se uma mensagem para o cliente. No segundo, existe uma periodicidade no envio das mensagens, assim se faz necessário um parâmetro extra indicando o período. Vale pontuar que os dois tipos de recursos são *observáveis*, isto é, um cliente ao "assinar" o *feed* daquele recurso receberá notificações sempre que ocorrer mudanças naquele recurso.

```
1 #define EVENT_RESOURCE(name, attributes, get_handler, post_handler, put_handler,
      delete_handler, event_handler)
```

```
1 #define PERIODIC_RESOURCE(name, attributes, get_handler, post_handler, put_handler
      , delete_handler, period, periodic_handler)
```

Para exemplificar a implementação de um recurso, será tomado como modelo o recurso *helloworld* do arquivo *er-example-server.c* do conjunto de exemplo do Er. Abaixo é exibido um fragmento do arquivo e alguns comentários traduzidos.

```
1 /*
2  * Assinatura do metodo: nome do recurso, metodo que o recurso manipula e URI.
3  */
4 RESOURCE(helloworld, METHOD_GET, "hello", "title=\"Hello world ?len=0..\";rt=\"Text\"");
5
6 /*
7  * Handleer function [nome do recurso]_handle (Deve ser implementado para cada recurso)
8  * Um ponteiro para buffer, no qual a conteudo (payload) da resposta sera anexado.
9  * Recursos simples podem ignorar os parametros preferred_size e offset, mas devem
10 * respeitar REST_MAX_CHUNK_SIZE (limite do buffer).
11 */
12 void helloworld_handler(void* request, void* response, uint8_t *buffer, uint16_t
       preferred_size, int32_t *offset) {
13   const char *len = NULL;
14   char const * const message = "Hello World!";
15   int length = 12; /*           |<---------->| */
16
17   /* A URI solicitada pode ser recuperada usando rest_get_query() ou usando um parser
       que retorna chave-valor. */
18   if (REST.get_query_variable(request, "len", &len)) {
19     length = atoi(len);
20     if (length<0) length = 0;
21   if (length>REST_MAX_CHUNK_SIZE) length = REST_MAX_CHUNK_SIZE;
22     memcpy(buffer, message, length);
23   } else {
24     memcpy(buffer, message, length);
25   }
26
27   REST.set_header_content_type(response, REST.type.TEXT_PLAIN);
28   REST.set_header_etag(response, (uint8_t *) &length, 1);
29   REST.set_response_payload(response, buffer, length);
30 }
31
32 /* Inicializa a REST engine. */
33 rest_init_engine();
34
35 /* Habilita o recurso. */
36 rest_activate_resource(&helloworld);
```

- Na linha 4 é declarado um *Normal Resource*, indicando o nome do recurso, bem como o método que o recurso aceita (pode aceitar mais de um método), seguidos da URI e de um texto de descrição.

- Na linha 12 é declarada a função que manipula as requisições para a URI do recurso. O padrão `[nome do recurso]_handler` deve ser mantido. Ao ser invocada, a função envia uma mensagem *"Hello World"* para o cliente. Além disso, se o parâmetro *"len"* for passado e o valor for menor que o comprimento da *string "Hello World"*, então a função retorna somente os caracteres `[0:len]`. Se o parâmetro contiver valor 0, então a função retorna uma *string* vazia.

- Entre as linhas 18 e 25 o processamento da requisição é realizado e o *buffer* de resposta é preenchido adequadamente.

- Nas linhas 27 e 29 o cabeçalho da mensagem de resposta é preenchido indicando respectivamente o tipo da mensagem (`TEXT_PLAIN`) e o comprimento da mensagem. Finalmente na linha 31 a carga da mensagem é preenchida.

- Uma vez que o recurso já foi declarado e implementado é preciso inicializar o *framework* REST e iniciar o processo CoAP, isto é realizado na linha 33. Além disso, é preciso habilitar o recurso para que ele esteja disponível para o acesso via URI, isto é feito na linha 36.

Este é um esboço da implementação de um recurso. É recomendável a leitura dos arquivos citados, bem como o material extra do minicurso para completo entendimento.

1.4.2.2. CoAP Erbium Contiki

Nesta etapa será apresentado o uso do Erbium Contiki.

Mude para o seguinte diretório:

```
<DIR>+/contiki/examples/er-rest-example/
```

Neste diretório existe uma conjunto de arquivos de exemplo do uso do Erbium[57]. Por exemplo, o arquivo *er-example-server.c* mostra como desenvolver aplicações REST do lado servidor. Já *er-example-client.c* mostra como desenvolver um cliente que consulta um determinado recurso do servidor a cada 10s. Antes de iniciar a prática é conveniente realizar algumas configurações. A primeira delas diz respeito aos endereços que serão utilizados. Deste modo, realize as operações abaixo:

Defina os endereços dos *Tmotes* no arquivo `/etc/hosts`

Adicione os seguintes mapeamentos:

```
aaaa::0212:7401:0001:0101 cooja1
aaaa::0212:7402:0002:0202 cooja2
...
```

Além disso, adicione a extensão Copper (CU)[58] CoAP *user-agent* no Mozilla Firefox.

[57] Para mais detalhes sobre os arquivo consulte: https://github.com/contiki-os/contiki/tree/master/examples/er-rest-example

[58] https://addons.mozilla.org/en-US/firefox/addon/copper-270430

No arquivo *er-example-server.c* verifique se as macros "REST_RES_[HELLO e TOGGLE]" possuem valor 1 e as demais macros em 0. Em caso negativo, modifique de acordo. Agora as configurações preliminares estão prontas.

Na pasta do exemplo Erbium Rest execute o comando abaixo:

```
make TARGET=cooja server-only.csc
```

Este comando iniciará uma simulação previamente salva. Esta simulação consta 2 dispositivos *Tmotes*, o dispositivo de ID 1 é um roteador de borda e o ID 2 emula um dispositivo executando *er-example-server.c* como *firmeware*.

Em um novo terminal execute o comando abaixo:

```
make connect-router-cooja
```

Como na primeira prática (Seção 1.4.1), este comando executará o programa `tunslip6` e realizará o mesmo procedimento anteriormente citado. Inicie a simulação, abra o Mozilla Firefox e digite: `coap://cooja2:5683/`[59].

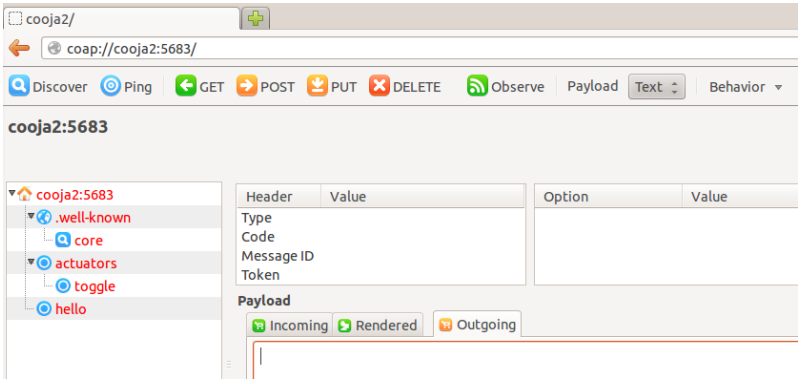


Figura 1.12. Resultado para `coap://cooja2:5683/`.

Como resultado o *Mozilla Firefox* deverá abrir o ambiente do Copper. A Figura 1.12 exemplifica a situação atual. No lado esquerdo é possível verificar os recursos disponíveis pelo servidor (cooja2). Para experimentar os recursos providos pelo cooja2, selecione o recurso "hello". Note que a URI foi alterada, em seguida, pressione o botão "GET" do ambiente Copper e observe o resultado. Já no recurso "toggle" pressione o botão "POST" e observe que o LED RED do cooja2 (no simulador) estará ligado, repita o processo e verifique novamente o LED.

É recomendável que os leitores alterem os recursos disponíveis pelo cooja2 no arquivo *er-example-server.c* modificando as macros "REST_RES_*" convenientemente. No material extra do curso, existem outras simulações e exemplos de uso. Vale ressaltar, que o mote emulado (*Tmote*) apresenta limitações de memória, como a maioria, dos objetos inteligentes e, por esse motivo, nem sempre todos os recursos poderão está ativos ao mesmo tempo. Em caso de excesso de memória, o simulador ou compilador para o dispositivo alertará tal problema.

Comentários no primeiro exercício foi possível empregar na prática os conceitos aprendidos na seções anteriores. Mostrou-se como funciona uma rede de objetos inteligentes utilizando os protocolos estado da arte para endereçar (6LoWPAN) e rotear (RPL) men-

[59] É importante frisar que o CoAP utiliza a porta 5683 como padrão.

sagens através de implementações disponíveis no Contiki. Na segunda prática, foi mostrado como acessar, de modo padronizado, os recursos dos objetos inteligentes por meio do protocolo CoAP, o qual emprega REST e URI para identificar unicamente os recursos disponíveis nos objetos inteligentes. No conteúdo *web* do capítulo[60] existem exemplos para implementação em dispositivos reais.

O conteúdo teórico e prático visto até o momento elucidaram as bases que sustentam o funcionamento da IoT hoje. Os conceitos e práticas visto são importantes para melhor compreensão das próximas seções, pois será assumido que uma rede de inteligentes funcional e que os recursos providos pelos dispositivos possam ser acessados utilizando, por exemplo, o CoAP ou similares.

1.5. Gerenciamento e Análise de Dados

Uma das principais características da IoT diz respeito à sua capacidade de proporcionar conhecimento sobre o mundo físico, a partir da grande quantidade de dados coletados pelos seus sensores. Por meio da mineração destes dados é possível descobrir padrões comportamentais do ambiente ou usuários e realizar inferências eles. Por exemplo, é possível concluir, apartir dos dados obtidos, sobre fenômenos naturais, de modo a permitir que aplicações possam antecipar condições meteorológicas e tomar decisões com base nisso. Obviamente, os maiores beneficiados com isto são os próprios usuários, que terão melhorias na qualidade de suas vidas.

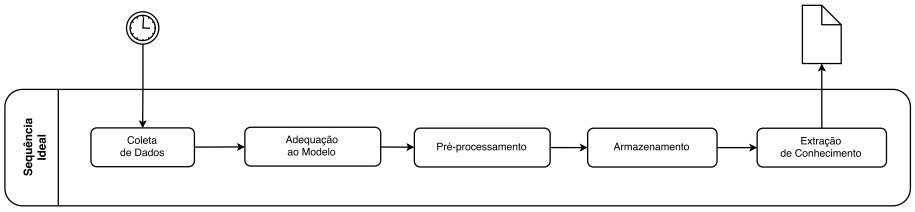
Contudo, para poder-se extrair todo o potencial contido nos dados da IoT é necessário que, primeiro, algumas medidas sejam tomadas. Nesta seção destacamos as principais características e desafios encontrados ao se lidar com dados oriundos destes sensores. Além disso, apresentamos algumas das principais técnicas utilizadas para modelagem e processamento destes dados, até a extração de conhecimento, para melhoria da qualidade dos serviços em cenários nos quais a IoT é empregada. Por fim, apontamos possíveis direcionamentos para aqueles que desejarem se aprofundar mais no assunto.

1.5.1. Descrição do Problema

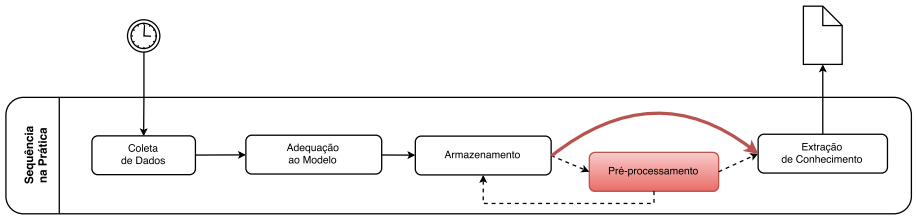
Do ponto de vista dos usuários e suas aplicações, a "*raison d'être*" (razão de ser) da IoT é, justamente, a extração de conhecimento a partir dos dados coletados pelos seus sensores. Extrair um conhecimento refere-se a modelar e analisar dados definindo uma semântica de forma a tomar decisões adequadas para prover um determinado serviço [Barnaghi et al. 2012]. Tomando como exemplo o cenário de *smart grids* [Yan et al. 2013], uma arquitetura de IoT pode auxiliar a controlar e melhorar o serviço de consumo de energia em casas e edifícios. Por meio da IoT, as fornecedoras de energia podem controlar os recursos proporcionalmente ao consumo e possíveis falhas na rede elétrica. Isso acontece por meio das diversas leituras que são coletadas por objetos inteligentes e são analisadas para prevenção e recuperação de falhas, aumentando, assim, a eficiência e qualidade dos serviços.

Para melhor entender o processo de extração de conhecimento, na Figura 1.13a são apresentadas as etapas ideais, que vão desde a coleta dos dados brutos até a extração

[60] http://homepages.dcc.ufmg.br/~bruno.ps/iot-tp-sbrc-2016/



(a) Representação da sequência ideal.



(b) Representação da sequência na prática.

Figura 1.13. Etapas para extração de conhecimento a partir de dados de sensores.

de conhecimento a partir deles. As etapas de adequação dos dados a um modelo, de pré-processamento e armazenamento são essenciais para que eles estejam aptos a serem processados e para que técnicas de inferências possam ser aplicadas sobre eles.

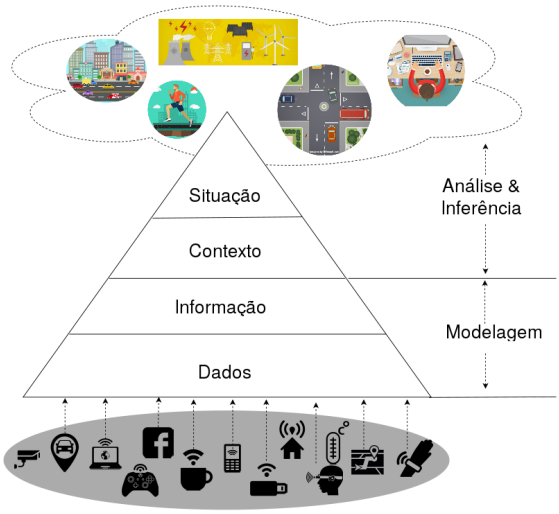


Figura 1.14. Hierarquia dos níveis de conhecimento a partir de dados brutos de sensores.

Do ponto de vista do conhecimento em si, uma outra abordagem que pode-se aplicar sobre este processo está ilustrada na Figura 1.14. Nela, a transformação da informação a partir de dados brutos de sensores pode ser entendida por meio de uma hierarquia de níveis de conhecimento. Estes níveis podem ser sub-divididos em dois momentos: (i) modelagem e (ii) análise e inferência. Na modelagem, cujo principal objetivo é o de adicionar semântica aos dados, depara-se com um grande volume de dados oriundos de diversos tipos de sensores. Uma particularidade desafiadora para manipular esses dados

é o fato de serem originados de múltiplas fontes e, em muitos casos, heterogêneas. Nesse sentido, algumas técnicas de pré-processamento, como fusão e representação de dados, podem ser adotadas, após isso, os dados são armazenados em uma representação adequada. O segundo momento consiste em interpretar tais dados visando obter contexto a partir deles e, com isso, realizar inferências para se extrair conhecimento quanto à situação de um determinado aspecto, seja ele um fenômeno natural, o estado de um usuário, entre outros.

Contudo, apesar de cada uma destas etapas ser de fundamental importância para o processo de extração de conhecimento como um todo, na prática não é bem isso o que acontece. Como ilustrado na Figura 1.13b, o que geralmente acontece é a ausência da etapa de pré-processamento dos dados antes que eles sejam armazenados. Os dados coletados dos sensores são simplesmente armazenados para posterior utilização, o que pode comprometer todas as inferências subsequentes. Assim, uma vez de posse dos dados armazenados, o que podemos fazer é realizar as atividades de pré-processamento em uma etapa adicional, antes que os dados sejam utilizados para a extração de conhecimento.

Considerando o cenário representado pelo que ocorre na prática, no restante desta seção aprofundamos nossas considerações sobre cada uma destas etapas.

1.5.2. Modelagem de dados

Dados brutos obtidos pelos sensores dificilmente possuem explicitamente uma organização hierárquica, relacionamentos ou mesmo um formato padrão para manipulação. Esta etapa de modelagem refere-se em definir uma representação para manipular e trabalhar com esses dados visando uma interoperabilidade e formatos padrões interpretáveis. Nesse sentido, aplica-se a modelagem que consiste em definir um conjunto de fatores tais como atributos, características e relações do conjunto de dados sensoriados. Tais fatores variam de acordo com o domínio da aplicação e consequentemente uma determinada solução de modelagem se torna mais adequada que outra. A modelagem aqui empregada consiste em representações conceituais de como representar a informação. As representações apresentadas são: *key-value*, *markup scheme*, *graphical*, *object based*, *logic based* e *ontology based modeling*. A aplicabilidade dessas representações pode variar de acordo com o domínio da aplicação. Portanto, cada representação é descrita abaixo considerando suas vantagens e desvantagens em uma perspectiva geral. Um estudo mais aprofundado pode ser encontrado em [Bettini et al. 2010].

- *Key-value based*: nesta representação os dados são modelados como um conjunto de chaves e valores em arquivos de texto. Apesar dessa ser a forma mais simples de representação da informação entre as apresentadas aqui, possui muitas desvantagens como a complexidade de organizar e recuperar quando o volume de dados aumenta, além da dificuldade de realizar relacionamentos entre os dados.

- *Markup scheme based*: um *markup scheme* utiliza *tags* para modelar os dados. Linguagens de marcação (e.g., XML[61]) e mecanismos de troca de dados (e.g, JSON[62]) podem ser utilizados para este fim e são bastante aplicados para armazenamento

[61] https://www.w3.org/XML/
[62] https://tools.ietf.org/html/rfc4627

e transferência de dados. A vantagem dessa técnica consiste na estruturação hierárquica dos dados e acesso aos dados utilizando as *tags*. Recentemente, o W3C adotou o EXI [REF] o qual consiste de uma representação compacta do XML. EXI pode ser uma relevante alternativa ao XML para IoT, pois é adequado para aplicações com recursos computacionais limitados. A SensorML[63] (Sensor Model Language) provê uma especificação exclusiva para sensores considerando diversos aspectos como localização e metadados.

- *Object based*: esta técnica permite a construção de uma modelagem considerando o relacionamento e a hierarquia entre os dados. Um *Markup scheme* apresenta limitações no sentido que as *tags* são palavras chaves de um domínio e não fornecem uma semântica para a interpretação pela máquina. Técnicas de modelagem como UML (*Unified Modelling Language*) e ORM (*Object Role Modelling*) são preferíveis à *Markup scheme*, pois permitem capturar relacionamentos em um contexto. Esse tipo de modelagem pode ser facilmente vinculada a uma linguagem de propósito geral e, consequentemente, integrável à sistemas de inferência e cientes de contexto. No entanto, não tem a mesma predisposição para realizar inferência das representações baseadas em lógica e ontologia.

- *Logic based*: neste tipo de modelagem expressões lógicas e regras são usadas para representar a informação. A representação lógica é mais expressiva que as anteriores do ponto de vista que possibilita a geração de novas informações a partir dos dados. Dessa forma, a partir de informações de baixo nível pode-se ter regras para compor informações de alto nível. Exemplo. A desvantagem dessa representação é a dificuldade em generalização das regras, dificultando a reusabilidade e aplicabilidade.

- *Ontology based*: nesta forma de representação as informações são modeladas como ontologia seguindo os conceitos da *Web* Semântica. Uma ontologia define um conjunto de tipos, propriedades e relacionamentos de entidades dentro de um tema considerando aspectos espaciais e temporais [REF]. Para a área de sensores e IoT, surgiu a *Sensor Semantic Web* que refere-se a aplicar o conhecimento da *Web* Semântica contemplando o domínio de sensores. Dessa forma, tecnologias consolidadas podem ser utilizadas, tais como RDF e OWL, para modelar o conhecimento de domínio e estruturar as relações definidas na ontologia. A desvantagem da representação baseada em ontologia concentra-se no consumo de recursos computacionais.

1.5.3. Armazenamento

Para que a grande quantidade de dados gerados pelos sensores da IoT possa ser posteriormente analisada e processada, a etapa de armazenamento faz-se essencial. Como apresentado na Figura 1.6(III), uma das principais formas de acesso a estes dados, e a mais comumente encontrada na prática, acontece por meio de servidores de armazenamento. Muitos destes estão disponíveis sob a forma de plataformas computacionais especificamente voltadas para prover serviços para a IoT. Na Tabela 1.4 apresentamos algumas

[63] http://www.sensorml.com/index.html

plataformas para IoT atualmente disponíveis, destacando algumas das suas principais características.

A maioria destas plataformas baseiam suas funcionalidades de acordo com os modelos de dados definidos, assim, logo após coletados, os dados quando adequados ao modelo serão armazenados de forma a possibilitar sua consulta subsequente. Porém, ao contrário do que foi discutido quanto os aspectos de modelagem mais adequados ao domínio de aplicação, o que geralmente ocorre, na prática, é a utilização de um modelo mais simples e genérico possível, que se adeque ao mais variado número de aplicações. Neste caso, os modelos que se encontram nas principais plataformas são baseados em *key-value* e *markup scheme*. Estes modelos são utilizados para que os usuários possam dar semântica aos seus dados, descrevendo coisas como os tipos dos dados, formatos, etc. Além disso, algumas outras meta informações também podem ser providas, como a localização dos sensores, uma descrição textual do que o sensor representa e algumas *tags* que poderão ser usadas como palavras-chave para consultas.

Além do armazenamento, algumas plataformas também disponibilizam outros serviços, como a marcação de tempo (*timestamp*) de todos os dados recebidos, algumas funcionalidades de processamento, que geralmente são pré-definidos e acessados sob a forma de *wizards* ou *dashboards*, definição de regras para a execução de atividades com base em eventos ou comportamento dos dados, entre outros. Para mais detalhes sobre o projeto e implementação de plataformas para IoT, ver o trabalho de [Paulo F. Pires 2015].

1.5.4. Pré-processamento

Para que os dados possam ser processados adequadamente eles devem possuir certa qualidade, que seja suficiente para os processos a serem empregados. Apesar de ser difícil de se conceituar, uma forma simples de entender a qualidade aqui discutida se refere aos dados que estão aptos ao uso [Bisdikian et al. 2013]. Nesse sentido, um passo fundamental para contornar possíveis problemas existentes nos dados coletados, de forma a torná-los aptos ao uso, é a de pré-processamento. Contudo, como visto na Figura 1.13b, apesar de sua importância, esta etapa de pré-processamento é muitas vezes inexistente, sendo os dados armazenados com suas imperfeições. Assim, diante do exposto, uma alternativa que se tem é a inserção desta etapa de pré-processamento logo antes dos dados serem utilizados para a extração de conhecimento.

A seguir discutimos alguns dos principais problemas que ocorrem no dados da IoT e algumas técnicas comumente utilizadas para reduzir seu impacto durante a etapa de extração de conhecimento.

1.5.4.1. Principais problemas dos dados

Para melhor compreendimento dos principais problemas que podem ocorrer nos dados provenientes dos sensores de IoT, [Khaleghi et al. 2013] definem uma taxonomia de classificação partindo de problemas básicos relativos aos aspectos dos dados, nos quais os principais são descritos abaixo.

- *Imperfeição*: refere-se aos efeitos causados por alguma imprecisão ou incerteza

Platforma	Endereço	Descrição	Tipo de conta
AWS IoT	aws.amazon.com/iot	Plataforma da Amazon voltada para empresas.	Possui conta gratuita, mas pede cartão de crédito para confirmar.
Arrayent	arrayent.com	Plataforma com foco empresarial.	Não disponibiliza conta gratuita.
Axeda	axeda.com	Plataforma com foco empresarial. Provê serviços de gerenciamento e comunicação entre dispositivos.	Não disponibiliza conta gratuita.
Beebotte	beebotte.com	Plataforma com interessantes recursos, incluindo a possibilidade de criação de dispositivos públicos.	Possui conta gratuita, mas com limite de 3 meses de histórico.
BugLabs	dweet.io	Permite *publish-subscribe* para disponibilização dos dados e integração entre sensores.	Possui conta gratuita, mas os dados são apagados após 24h.
Carriots	carriots.com	Plataforma com foco empresarial. Provê serviços de gerenciamento de e comunicação entre dispositivos	Possui conta gratuita, mas com várias limitações, *e.g.*, número de dispositivos e acessos.
Electric imp	electricimp.com	Plataforma em nuvem que integra o conjunto de soluções dos dispositivos *electric imp*.	Possui conta gratuita.
Evrythng	evrythng.com	Plataforma com foco empresarial. Permite *publish-subscribe* para disponibilização dos dados de sensores.	Disponibiliza conta gratuita para desenvolvedor.
Exosite	exosite.com	Plataforma com foco empresarial.	Possui conta gratuita, mas limitada.
Flowthings	flowthings.io	Plataforma com interessantes conceitos para a integração de sensores.	Possui conta gratuita para desenvolvedor.
IBM Bluemix	bluemix.net	Plataforma da IBM com foco empresarial.	Possui conta gratuita, mas pede cartão de crédito para confirmar.
Kaa	kaaproject.org	*Middleware open source* disponível para a criação de sua própria plataforma para IoT.	Gratuita.
Lelylan	lelylan.com	Projeto *open source* com interessantes conceitos de arquitetura que pode ser utilizado para a criação de sua própria plataforma.	Gratuita.
Linkafy	linkafy.com	Plataforma voltada para o controle de dispositivos residenciais.	Possui conta gratuita, mas limitada a apenas dispositivo.
mbed	mbed.com	Plataforma que integra o conjunto de soluções que suportam os dispositivos mbed da ARM.	Possui conta gratuita, com limitações.
Microsoft Azure IoT	microsoft.com/iot	Plataforma da Microsoft voltada a IoT com foco empresarial.	Possui conta gratuita de um mês para testes.
Open.sen.se	open.sen.se	Plataforma interessante para integração dos sensores e seus dados.	Conta apenas após requisição.
OpenSensors	opensensors.io	Plataforma robusta com o foco principal para sensores abertos. Permite *publish-subscribe* para disponibilização dos dados de sensores.	Conta gratuita disponível, paga-se apenas para ter sensores privados.
PubNub	pubnub.com	Plataforma robusta com diversas funcionalidades voltadas para IoT.	Possui conta gratuita com limitações.
SensorCloud	sensorcloud.com	Plataforma com foco empresarial.	Possui conta gratuita, mas limitada.
SensorFlare	sensorflare.com	Plataforma para integração de sensores, mas apenas suporta alguns fabricantes e modelos.	Possui conta gratuita.
Sentilo	sentilo.io	Projeto gratuito e disponível para a criação de sua própria plataforma. Voltada às arquiteturas de *smart cities*.	Gratuita.
Shiftr	shiftr.io	Interessantes conceitos para a integração de sensores de forma intuitiva.	Possui conta gratuita.
ThingPlus	thingplus.net	Plataforma sul koreana com o foco interessante sistema de definição de regras para integração entre sensores.	Possui conta gratuita, mas limitada.
ThingSquare	thingsquare.com	Plataforma voltada ao controle de dispositivos e integração via celular.	Possui conta de desenvolvedor gratuita.
ThingSpeak	thingspeak.com	Plataforma robusta, com várias funcionalidades, como sensores públicos e busca por histórico.	Conta gratuita disponível.
ThingWorx	thingworx.com	Plataforma com recursos interessantes, mas mais voltada a soluções empresariais.	Conta gratuita, mas com limitações.
Xively	xively.com	Plataforma robusta, disponibiliza várias funcionalidades como busca por sensores públicos e histórico.	Foco empresarial, com conta paga disponível e gratuita apenas via solicitação.

Tabela 1.4. Algumas das principais plataformas para IoT.

nas medidas capturadas pelos sensores. As causas destes problemas podem ser várias, desde a problemas nas leituras devido a falhas de *hardware* ou calibragem dos sensores, até a fatores externos ao sensor, como do seu posicionamento em locais que adicionam ruídos às suas leituras.

- *Inconsistência*: surge principalmente dos seguintes problemas: (i) dados fora de sequência, isto é, em que a ordem em que foram armazenados ou temporalmente demarcados difere da real ordem de ocorrência no mundo físico, (ii) presença de *outliers* nos dados, observações que estão bem distantes das demais observações realizadas, causadas por alguma situação inesperada, e (iii) dados conflitantes, quando diferentes sensores medindo um mesmo fenômeno geram dados diferentes, agregando uma dúvida sobre qual sensor seria mais confiável.

- *Discrepância*: este é um problema causado, principalmente, quando diferentes tipos de sensores são utilizados para coletar dados sobre um mesmo fenômeno. Por exemplo, sensores físicos coletando informações sobre o tráfego, comparados com câmeras de monitoramento, ou ainda, sensores sociais, que coletam informações dadas por usuários em seus aplicativos móveis [Silva et al. 2014].

Além destes problemas, também podemos destacar outros problemas dos dados, principalmente no cenário atual da IoT, onde usuários comuns também estão participando de forma colaborativa da geração destes dados [Borges Neto et al. 2015]. Devido à popularização e redução dos custos dos sistemas computacionais embarcados, muitos usuários estão criando seus próprios sensores, seguindo um modelo DIY (*Do It Yourself*). Neste cenário, eles são responsáveis pela implantação, coleta e distribuição dos dados dos sensores, geralmente tornando-os públicos através das principais plataformas de IoT. Contudo, por se tratarem de sensores "particulares", não há nenhuma garantia quanto à qualidade dos dados gerados, nem mesmo da disponibilidade dos sensores.

Assim, alguns problemas que podem surgir neste novo cenário são: (i) ausência de descrição dos dados gerados, quando os usuários submetem os dados coletados pelos seus sensores para uma plataforma sem descrever de que se trata o dado, dificultando qualquer posterior utilização deste dado por outros usuários, (ii) disponibilidade dos sensores, quando os sensores param de funcionar, temporária ou permanentemente, sem mais detalhes se voltaram a operar ou não, geralmente de acordo com as necessidades dos próprios usuários, (iii) imprecisão das leituras causadas pelo baixo custo e qualidade dos sensores "caseiros", geralmente utilizando-se de técnicas alternativas para medir um fenômeno, as leituras destes sensores dos usuários podem diferir em magnitude dos dados coletados por sensores mais profissionais quanto a um mesmo fenômeno.

Para exemplificar isto, a Figura 1.15 apresenta duas medições distintas para um mesmo fenômeno, a velocidade do vento, na região da Espanha, realizadas por um sensor público, disponibilizado por um usuário através da plataforma ThingSpeak[64] e pelo serviço meteorológico do Weather Underground[65], no período entre 12 de setembro a 31 de outubro de 2015. Observa-se que embora bem relacionadas, com um coeficiente de

[64] https://thingspeak.com
[65] https://www.wunderground.com

correlação equivalente a 0.915 entre eles, suas magnitudes são diferentes. Além disso, também pode-se notar a incompletude destes dados, quando há lacunas devido à falta de dados coletados pelos sensores.

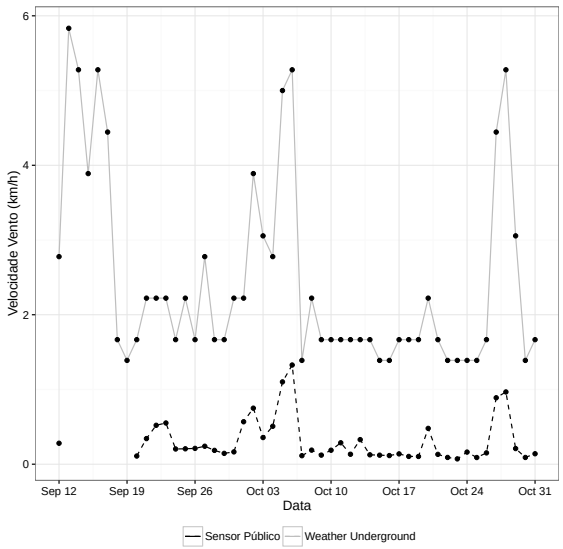


Figura 1.15. Velocidade do Vento (km/h) coletada na região de Madri, Espanha, medida por um sensor público, disponível na plataforma ThingSpeak (linha tracejada) e pelo serviço meteorológico do Weather Underground (linha sólida).

1.5.4.2. Principais técnicas de pré-processamento

O pré-processamento consiste na aplicação de técnicas, em sua maioria estatísticas e demais operações matemáticas, sobre os dados com o intuito melhorar a sua qualidade, contornando possíveis problemas existentes e removendo imperfeições. A complexidade desta etapa pode ser alta, principalmente neste cenário em questão, quando os dados são heterogêneos, oriundos de diversas fontes e com diferentes problemas. Nesse sentido, a decisão sobre qual técnica aplicar varia de acordo com o problema encontrado nos dados e, além disso, do que se espera fazer com estes dados. Assim, a seguir será apresentada uma visão geral sobre algumas das técnicas comumente utilizadas para alguns dos problemas citados, e apontamos algumas referências, para que leitores mais interessados possam se aprofundar no assunto.

- *Imprecisões e outliers*: A presença de imprecisões e *outliers* em dados é um problema muito comum em diversas áreas e um ponto bastante estudado pela comunidade científica. As soluções mais comumente utilizadas para contornar estes problemas são baseadas em técnicas de suavização dos dados. Por exemplo, (i) a utilização de médias móveis (*moving average*), onde as amostras são suavizadas por meio da média de amostras vizinhas, e (ii) de interpolação (polinomial ou *splines*), onde são definidas funções, *e.g.*, polinomiais, que melhor se ajustam às

amostras contidas nos dados. Com isto, é possível diminuir o efeito dos ruídos causados pelas imperfeições nos dados. Para mais detalhes, uma consulta pode ser [Morettin and Toloi 2006].

- *Lacunas nos dados*: O caso de lacunas nos dados acontece por fatores diversos e que podem causar diferentes níveis de prejuízo em sua extração de conhecimento. Um caso mais simples seria a lacuna causada por uma falha esporádica na operação dos sensores, por razões não associadas ao fenômeno monitorado, causando uma lacuna aleatória ou MAR (*missing at random*). Este tipo de falha pode ser contornada de forma mais fácil, por exemplo, por meio de interpolação para completar os dados faltantes. Mas, dependendo da forma como isso acontece, podem ser inseridos lacunas de forma sistemáticas nos dados, prejudicando a sua posterior análise. Por exemplo, um sensor que pára de coletar dados quando a temperatura atinge certos níveis de valores, ou quando um usuário desliga o sensor à noite quando deixa o escritório. Este tipo de lacunas que não são geradas aleatoriamente, ou MNAR (*missing not at random*) são mais problemáticas para o seu processamento. Outras técnicas também podem ser aplicadas, desde a simples remoção do período contendo lacunas até a imputação dos dados faltantes por meio da estimativa a partir dos demais dados existentes. Mais detalhes em [Schafer and Graham 2002].

- *Diferença de granularidade*: Devido à grade quantidade de sensores heterogêneos, um problema relevante que surge diz respeito às diferenças de amostragem que cada sensor possui. Isso pode causar diferenças significativas na granularidade dos dados de diferentes sensores. Por exemplo, para um mesmo fenômeno em um mesmo intervalo de tempo, podemos ter um sensor coletando dados a cada 5 segundos e outro a cada 1 minuto. Para que seja possível combinar os dados destes sensores, um primeiro pré-processamento seria igualar a quantidade de amostras de cada um deles. Para isto, algumas técnicas podem ser utilizadas e, uma simples mas eficaz é a PAA (*Piecewise Aggregate Approximation*), que é empregada pelo algoritmo de clusterização SAX (*Symbolic Aggregate approXimation*) [Lin et al. 2003], para a redução da dimensão do conjunto de dados por meio da agregação de dados, similar ao que é feito com médias móveis. Assim, pode-se transformar os dados de forma a possuírem a mesma dimensão. Ainda sobre a redução de dimensão, esta é uma técnica bem válida para o cenário de IoT, pois espera-se uma grande quantidade de dados sendo gerados. Outras estratégias também podem ser adotadas, como a transformação via *wavelets* ou Fourier, mais detalhes em [Rani and Sikka 2012, Warren Liao 2005].

- *Combinação de diversas fontes*: Para a combinação de dados de diversas fontes, como é o caso da IoT, técnicas de fusão de dados podem ser aplicadas de forma a compor uma única representação destes dados. Fusão de dados é o método de combinar dados de diversos sensores para produzir uma semântica mais precisa e muitas vezes inviável de ser obtida a partir de um único sensor. A utilização de fusão de dados nesta etapa de pré-processamento tem o intuito principal de cobrir os problemas dos dados de um sensor através dos dados coletados por outros sensores similares, gerando assim um novo dado combinado mais preciso. Dessa forma, são aplicados métodos automáticos ou semiautomáticos para transformar dados de dife-

rentes fontes em uma representação que seja significativa para a tomada de decisão e inferência. Em cenários de IoT nos quais existem milhares de sensores a fusão de dados é uma poderosa ferramenta tanto processo de extração de conhecimento quanto para reduzir recursos computacionais semelhante ao que ocorre em redes de sensores sem fio. Várias técnicas de fusão podem ser aplicadas, como, por exemplo, filtros de Kalman [Khaleghi et al. 2013]. Outra técnicas muito utilizadas para a combinação de sensores distintos se baseiam na clusterização de dados similares, com base em alguma métrica de distância, mas mais detalhes sobre isso algumas técnicas de clusterização vide [Rani and Sikka 2012, Warren Liao 2005].

1.5.5. Extração de conhecimento

Como visto na Figura 1.14 a modelagem consiste em explorar os dados brutos para estruturá-los em uma específica representação. Aumentando o nível de abstração na hierarquia, contexto refere-se a qualquer informação que pode ser utilizada para caracterizar a situação de uma entidade. Sendo essa entidade um objeto, lugar ou pessoa que é considerada relevante para interação entre um usuário e uma aplicação, incluindo o usuário e a própria aplicação [Dey 2001]. Enquanto que uma situação representa a interpretação semântica de contextos, geralmente derivado pela combinação de diversas informações contextuais, proporcionando conhecimento sobre o mundo físico [Dobson and Ye 2006]. Particularmente, a etapa de análise e inferência busca definir o contexto e a situação a partir dos dados coletados por sensores.

Para exemplificar um cenário, Bettini [Bettini et al. 2010] especifica uma situação "reunião está ocorrendo" considerando as seguintes informações contextuais: localização de várias pessoas e agenda de atividades delas, informação de sensores nos pisos, quais os dispositivos ativos na sala, o som ambiente e as imagens de câmeras. Ao longo do tempo essas informações contextuais vão se alterando, mas a situação ainda é a mesma. Além disso, outras informações contextuais poderão ser incorporadas, enquanto essas utilizadas podem se tornar obsoletas. Dessa forma, o objetivo da inferência refere-se a estratégia de extrair um conhecimento (i.e., semântica de uma situação) baseado nos dados coletados. Esta etapa é relacionada com a etapa de modelagem, pois algumas técnicas de modelagem são preferíveis de serem utilizadas com determinado tipo de técnica de inferência. A seguir, são apresentadas algumas categorias de técnicas úteis para inferência de situações. Uma revisão mais abrangente sobre tais técnicas pode ser encontrada em [Bettini et al. 2010, Perera et al. 2014].

- *Aprendizagem Supervisionada*: Nesse tipo de técnica, uma primeira coleção de dados é coletada para compor o conjunto de treinamento. Para tanto, esse dados devem ser rotulados em classes específicas. Em seguida, cria-se um modelo (ou função) que aprende a classificar dados de acordo com os dados que foram utilizados na etapa de treinamento. As árvores de decisão, redes neurais artificiais baseadas neste tipo de aprendizado e máquinas de vetores de suporte são exemplos de técnicas de aprendizagem supervisionada. Dois fatores importantes são a seleção de características significativas e uma considerável quantidade de dados para classificação.

- *Aprendizagem não supervisionada*: Nesta categoria de técnicas, não existe nenhum conhecimento a priori sobre que padrões ocorrem nos dados e o objetivo é encontrar um modelo (ou função) que descubra padrões implícitos em um conjunto de dados não rotulados. Dessa forma, é difícil uma confiável validação dos resultados obtidos e os resultados geralmente não são previsíveis. Exemplos clássicos desta categoria são as técnicas de clusterização tais como K-Means e clusterização hierárquica que agrupa os elementos em análise se baseando em métricas de similaridade entre eles.

- *Regras*: Este tipo de técnica consiste em definir regras condicionais. Apesar de ser a forma mais simples de inferência, ela apresenta dificuldades de generalização e pouca extensibilidade para diferentes aplicações.

- *Ontologias*: Nessa categoria, ontologias podem ser utilizadas tanto na modelagem como para a inferência. Assim, já se tem a vantagem de modelar um conhecimento sabendo o domínio da aplicação e o mapeamento das possibilidades de inferência. No entanto, apresenta a desvantagem de não tratar ambiguidades ou resultados inesperados. Essas desvantagens podem ser contornadas fazendo a combinação com regras.

- *Lógica probabilística*: Também conhecida como inferência probabilística refere-se a utilizar a teoria de probabilidade para lidar com as incertezas existentes em um processo dedutivo. Ao contrário das ontologias, esta categoria de técnicas pode lidar com ambiguidades considerando as probabilidades associadas para realizar a inferência. A desvantagem consiste em descobrir tais probabilidades. *Hidden Markov Models* (HMM) é um exemplo clássico e tem sido aplicado em cenários de reconhecimento de atividades de pessoas.

1.6. IoT como o meio para a Computação Ubíqua e pervasiva LOUREIRO (MAX 1 PG)

- Definição de entidades
  - Diferentes tipos
  - Diferentes contextos (Físico e Lógicos)
- Aquisição de contexto através dos dados dos dispositivos inteligentes
- Elementos para monitoramento (sensores)
- O papel da "*Cloud Computing*"
- Exemplos de Aplicações (Automação residencial, Smart Cities, Urban Networks, Monitoramento de Saúde)

1.7. Considerações Finais A DEFINIR (MAX 2 pg)

- Questões não comentadas no capítulo (essas questões devem entrar em alguma seção?)

  - Como tornar os dispositivos Plug & Play
  - Localização (quais os problemas existentes? como fazer?)
  - Segurança
  - Descoberta de Serviços
  - Desempenho X quantidades de acessos

- Revisão do texto e de forma tabular listar os problemas de pesquisa de cada seção.
- Agradecimentos

Referências

[Al-Fuqaha et al. 2015] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *Communications Surveys & Tutorials, IEEE*, 17(4):2347–2376.

[Alliance 2015] Alliance, W.-F. (2015). Who we are.

[Angell et al. 1983] Angell, J. B., Barth, P. W., and Terry, S. C. (1983). Silicon micromechanical devices. *Scientific American*, 248:44–55.

[Ashton 2009] Ashton, K. (2009). That 'internet of things' thing. *RFiD Journal*, 22(7):97–114.

[Bagula and Erasmus 2015] Bagula, B. and Erasmus, Z. (2015). Iot emulation with cooja. ICTP-IoT Workshop.

[Barnaghi et al. 2012] Barnaghi, P., Wang, W., Henson, C., and Taylor, K. (2012). Semantics for the Internet of Things. *International Journal on Semantic Web and Information Systems*, 8(1):1–21.

[Bettini et al. 2010] Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., and Riboni, D. (2010). A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2):161–180.

[Bisdikian et al. 2013] Bisdikian, C., Kaplan, L. M., and Srivastava, M. B. (2013). On the quality and value of information in sensor networks. *ACM Transactions on Sensor Networks*, 9(4):1–26.

[Borges Neto et al. 2015] Borges Neto, J. B., Silva, T. H., Assunção, R. M., Mini, R. A. F., and Loureiro, A. A. F. (2015). Sensing in the Collaborative Internet of Things. *Sensors*, 15(3):6607–6632.

[Boulis et al. 2011] Boulis, A. et al. (2011). Castalia: A simulator for wireless sensor networks and body area networks. *NICTA: National ICT Australia*.

[Chaouchi 2013] Chaouchi, H. (2013). *The internet of things: connecting objects*. John Wiley & Sons.

[Chlipala et al. 2004] Chlipala, A., Hui, J., and Tolle, G. (2004). Deluge: data dissemination for network reprogramming at scale. *University of California, Berkeley, Tech. Rep*.

[Clausen et al. 2013] Clausen, T., Yi, J., Herberg, U., and Igarashi, Y. (2013). Observations of rpl: Ipv6 routing protocol for low power and lossy networks. draft-clausen-lln-rpl-experiences-05.

[Cordero et al. 2013] Cordero, J. A., Yi, J., Clausen, T. H., and Baccelli, E. (2013). Enabling Multihop Communication in Spontaneous Wireless Networks. In H. Haddadi, O. B., editor, *Recent Advances in Networking*, volume 1 of *ACM SIGCOMM eBook*. ACM.

[Da Xu et al. 2014] Da Xu, L., He, W., and Li, S. (2014). Internet of Things in industries: A survey. *Industrial Informatics, IEEE Transactions on*, 10(4):2233–2243.

[Dey 2001] Dey, A. K. (2001). Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7.

[Dobson and Ye 2006] Dobson, S. and Ye, J. (2006). Using fibrations for situation identification. In *Pervasive 2006 workshop proceedings*, pages 645–651.

[Doddavenkatappa et al. 2012] Doddavenkatappa, M., Chan, M. C., and Ananda, A. L. (2012). *Testbeds and Research Infrastructure. Development of Networks and Communities: 7th International ICST Conference,TridentCom 2011, Shanghai, China, April 17-19, 2011, Revised Selected Papers*, chapter Indriya: A Low-Cost, 3D Wireless Sensor Network Testbed, pages 302–316. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Dunkels et al. 2004] Dunkels, A., Gronvall, B., and Voigt, T. (2004). Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, LCN '04, pages 455–462, Washington, DC, USA. IEEE Computer Society.

[Fasolo et al. 2007] Fasolo, E., Rossi, M., Widmer, J., and Zorzi, M. (2007). In-network aggregation techniques for wireless sensor networks: a survey. *Wireless Communications, IEEE*, 14(2):70–87.

[Forbes 2014] Forbes (2014). Internet of Things By The Numbers: Market Estimates And Forecasts.

[Gartner 2015] Gartner, I. (2015). Gartner's 2015 Hype Cycle for Emerging Technologies Identifies the Computing Innovations That Organizations Should Monitor.

[Gnawali et al. 2009] Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., and Levis, P. (2009). Collection Tree Protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*.

[Goldstein et al. 2005] Goldstein, S. C., Campbell, J. D., and Mowry, T. C. (2005). Programmable matter. *Computer*, 38(6):99–101.

[Hui 2012] Hui, J. W. (2012). The routing protocol for low-power and lossy networks (rpl) option for carrying rpl information in data-plane datagrams.

[Javaid et al. 2009] Javaid, N., Javaid, A., Khan, I., and Djouani, K. (2009). Performance study of ETX based wireless routing metrics. In *2nd IC4 2009*.

[Kelly et al. 2013] Kelly, S. D. T., Suryadevara, N. K., and Mukhopadhyay, S. C. (2013). Towards the implementation of IoT for environmental condition monitoring in homes. *Sensors Journal, IEEE*, 13(10):3846–3853.

[Khaleghi et al. 2013] Khaleghi, B., Khamis, A., Karray, F. O., and Razavi, S. N. (2013). Multisensor data fusion: A review of the state-of-the-art. *Information Fusion*, 14(1):28–44.

[Khan et al. 2012] Khan, R., Khan, S. U., Zaheer, R., and Khan, S. (2012). Future internet: the internet of things architecture, possible applications and key challenges. In *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, pages 257–260. IEEE.

[Kovatsch et al. 2011] Kovatsch, M., Duquennoy, S., and Dunkels, A. (2011). A low-power coap for contiki. In *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, pages 855–860. IEEE.

[Kurose and Ross 2012] Kurose, J. F. and Ross, K. W. (2012). *Computer Networking: A Top-Down Approach (6th Edition)*. Pearson, 6th edition.

[Kushalnagar et al. 2007] Kushalnagar, N., Montenegro, G., and Schumacher, C. (2007). Ipv6 over low-power wireless personal area networks (6lowpans): overview, assumptions, problem statement, and goals. Technical report.

[Levis and Lee 2010] Levis, P. and Lee, N. (2010). TOSSIM: A Simulator for TinyOS Networks.

[Levis et al. 2003] Levis, P., Lee, N., Welsh, M., and Culler, D. (2003). Tossim: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pages 126–137, New York, NY, USA. ACM.

[Levis et al. 2005] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., and Culler, D. (2005). *Ambient Intelligence*, chapter TinyOS: An Operating System for Sensor Networks, pages 115–148. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Levis et al. 2004] Levis, P., Patel, N., Culler, D., and Shenker, S. (2004). Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, NSDI'04, pages 2–2, Berkeley, CA, USA. USENIX Association.

[Lin et al. 2003] Lin, J., Keogh, E., Lonardi, S., and Chiu, B. (2003). A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery - DMKD '03*, pages 2 – 11, New York, New York, USA. ACM Press.

[Liu et al. 2013] Liu, V., Parks, A., Talla, V., Gollakota, S., Wetherall, D., and Smith, J. R. (2013). Ambient backscatter: wireless communication out of thin air. *ACM SIGCOMM Computer Communication Review*, 43(4):39–50.

[Loureiro et al. 2003] Loureiro, A. A., Nogueira, J. M. S., Ruiz, L. B., Mini, R. A. d. F., Nakamura, E. F., and Figueiredo, C. M. S. (2003). Redes de Sensores Sem Fio. In *Simpósio Brasileiro de Redes de Computadores (SBRC)*, pages 179–226.

[Mattern and Floerkemeier 2010] Mattern, F. and Floerkemeier, C. (2010). From the internet of computers to the internet of things. In *From active data management to event-based systems and more*, pages 242–259. Springer.

[Morettin and Toloi 2006] Morettin, P. A. and Toloi, C. M. C. (2006). *Análise de Séries Temporais*. Blucher, São Paulo, 2nd edition.

[Österlind and of Computer Science 2006] Österlind, F. and of Computer Science, S. I. (2006). *A Sensor Network Simulator for the Contiki OS*. SICS technical report. Swedish institute of computer science.

[Patel and Rutvij H. 2015] Patel, K. N. and Rutvij H., J. (2015). A survey on emulation testbeds for mobile ad-hoc networks. *Procedia Computer Science*, 45:581–591.

[Paulo F. Pires 2015] Paulo F. Pires, Flavia C. Delicato, T. B. (2015). Plataformas para a Internet das Coisas.

[Perera et al. 2014] Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2014). Context aware computing for the internet of things: A survey. *Communications Surveys & Tutorials, IEEE*, 16(1):414–454.

[Peterson and Davie 2011] Peterson, L. L. and Davie, B. S. (2011). *Computer Networks, Fifth Edition: A Systems Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition.

[Ramos et al. 2012] Ramos, H. S., Oliveira, E. M., Boukerche, A., Frery, A. C., and Loureiro, A. A. (2012). Characterization and mitigation of the energy hole problem of many-to-one communication in wireless sensor networks. In *Computing, Networking and Communications (ICNC), 2012 International Conference on*, pages 954–958. IEEE.

[Rani and Sikka 2012] Rani, S. and Sikka, G. (2012). Recent Techniques of Clustering of Time Series Data: A Survey. *International Journal of Computer Applications*, 52(15):1–9.

[Reiter 2014] Reiter, G. (2014). Wireless connectivity for the internet of things: One size does not fit all. page 11.

[RERUM 2015] RERUM (2015). Advanced techniques to increase the lifetime of smart objects and ensure low power network operation. *RERUM*.

[Ruiz et al. 2004] Ruiz, L. B., Correia, L. H. A., Vieira, L. F. M., Macedo, D. F., Nakamura, E. F., Figueiredo, C. M., Vieira, M. A. M., Bechelane, E. H., Camara, D., Loureiro, A. A., et al. (2004). Arquiteturas para redes de sensores sem fio.

[Saltzer et al. 1984] Saltzer, J. H., Reed, D. P., and Clark, D. D. (1984). End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 2(4):277–288.

[Santos et al. 2015a] Santos, B., Vieira, M., and Vieira, L. (2015a). eXtend collection tree protocol. In *Wireless Communications and Networking Conference (WCNC), 2015 IEEE*, pages 1512–1517.

[Santos et al. 2015b] Santos, B. P., Menezes Vieira, L. F., and Menezes Vieira, M. A. (2015b). CRAL: a centrality-based and energy efficient collection protocol for low power and lossy networks. In *Computer Networks and Distributed Systems (SBRC), 2015 XXXIII Brazilian Symposium on*, pages 159–170. IEEE.

[Schafer and Graham 2002] Schafer, J. L. and Graham, J. W. (2002). Missing data: Our view of the state of the art. *Psychological Methods*, 7(2):147–177.

[Silva et al. 2014] Silva, T., Vaz De Melo, P., Almeida, J., and Loureiro, A. (2014). Large-scale study of city dynamics and urban social behavior using participatory sensing. *Wireless Communications, IEEE*, 21(1):42–51.

[Sundmaeker et al. 2010] Sundmaeker, H., Guillemin, P., Friess, P., and Woelfflé, S. (2010). *Vision and challenges for realising the Internet of Things*, volume 20. EUR-OP.

[Tanenbaum 2002] Tanenbaum, A. (2002). *Computer Networks*. Prentice Hall Professional Technical Reference, 4th edition.

[Tanenbaum 2011] Tanenbaum, A. (2011). *Computer Networks*. Prentice Hall Professional Technical Reference, 5th edition.

[Tsvetko 2011] Tsvetko, T. (2011). Rpl: Ipv6 routing protocol for low power and lossy networks. In *Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS*.

[Varga et al. 2001] Varga, A. et al. (2001). The omnet++ discrete event simulation system. In *Proceedings of the European simulation multiconference (ESM'2001)*, volume 9, page 65. sn.

[Varga and Hornig 2008] Varga, A. and Hornig, R. (2008). An Overview of the OMNeT++ Simulation Environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 60:1–60:10, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[Vasseur et al. 2011] Vasseur, J., Agarwal, N., Hui, J., Shelby, Z., Bertrand, P., and Chauvenet, C. (2011). Rpl: The ip routing protocol designed for low power and lossy networks. Internet Protocol for Smart Objects (IPSO) Alliance.

[Vasseur and Dunkels 2010] Vasseur, J.-P. and Dunkels, A. (2010). *Interconnecting Smart Objects with IP: The Next Internet*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Wang et al. 2015] Wang, F., Hu, L., Zhou, J., and Zhao, K. (2015). A Survey from the Perspective of Evolutionary Process in the Internet of Things. *International Journal of Distributed Sensor Networks*, 2015.

[Warren Liao 2005] Warren Liao, T. (2005). Clustering of time series data - A survey. *Pattern Recognition*, 38(11):1857–1874.

[Weingärtner et al. 2009] Weingärtner, E., Vom Lehn, H., and Wehrle, K. (2009). A performance comparison of recent network simulators. In *Proceedings of the 2009 IEEE International Conference on Communications*, ICC'09, pages 1287–1291, Piscataway, NJ, USA. IEEE Press.

[Yan et al. 2013] Yan, Y., Qian, Y., Sharif, H., and Tipper, D. (2013). A Survey on Smart Grid Communication Infrastructures: Motivations, Requirements and Challenges. *IEEE Communications Surveys & Tutorials*, 15(1):5–20.

[Zachariah et al. 2015a] Zachariah, T., Klugman, N., Campbell, B., Adkins, J., Jackson, N., and Dutta, P. (2015a). The internet of things has a gateway problem. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pages 27–32. ACM.

[Zachariah et al. 2015b] Zachariah, T., Klugman, N., Campbell, B., Adkins, J., Jackson, N., and Dutta, P. (2015b). The internet of things has a gateway problem. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, HotMobile '15, pages 27–32, New York, NY, USA. ACM.