

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 (D2D), máquinas (M2M) e formar novas aplicações. Várias questões teóricas e práticas surgem no desenvolvimento de 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, e discutir desafios e questões de pesquisa. Através de uma abordagem crítica, é exposta 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.

Talvez fosse interessante começar esta seção estabelecendo a importância do tópico. Tipo uma motivação para o tópico...

1.1. Introdução BRUNO (MAX 5 PG)

Este capítulo aborda a Internet das Coisas (do inglês *Internet of Things* – IoT) 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 de interesses, expectativas e pesquisas na área. Logo em seguida, são introduzidos alguns dos desafios e questões de pesquisa que surgem ao trabalhar no contexto da IoT. Para iniciar a discussão, será retomada a atenção aos fundamentos das redes de computadores com uma questão elementar: o que são redes de computadores?

Desconsidere a ~~marcação~~ marcação

Esses desafios são diferentes dos desafios mencionados na linha 3?

Segundo [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 a grande quantidade de equipamentos e tecnologias não tradicionais que são usadas na Internet.

Para nós, os autores, diante da nova época tecnológica, as Redes de Computadores podem ser definidas como um combinado dos três pontos de vista anteriores. De modo mais formal, quando dois ou mais dispositivos com capacidade de processamento, armazenamento e comunicação trocam informações trata-se de uma rede. Atualmente, não só computadores convencionais estão conectados a grande rede, como também uma grande heterogeneidade de equipamentos tais como TV, Laptops, automóveis, smartphones, consoles de jogos, webcams e a lista parece não ter fim. Neste novo cenário, a pluralidade é crescente e previsões indicam mais de 40 bilhões de dispositivos estarão conectados até 2020 [Forbes 2014]. Concomitantemente uma gama de novas possibilidades de aplicações (cidades inteligentes (*Smart Cities*), saúde (*Healthcare*), casas inteligentes (*Smart Home*)) e desafios emergem (regulamentações, segurança, padronizações).

TVs

É importante notar que um dos elementos cruciais para o sucesso da IoT encontra-se nas padronizações das tecnologias. Isto permitirá que a heterogeneidade de dispositivos conectados cresça. 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.

1.1.1. O que é a Internet das Coisas ?

Na IoT, eventualmente, a unidade básica de *hardware* apresentará ao menos uma das seguintes características: i) unidade(s) de processamento; ii) unidade(s) de memória; iii) unidade(s) 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*). O termo objeto, aqui empregado, é genérico tal qual a palavra “coisa”. Deste modo, entende-se que qualquer coisa pode vir a possuir de modo inerente, acoplado ou ainda ser posto em proximidade a objetos inteligentes e viabilizar interações. Estes objetos ao estabelecerem

unidade(s) DE

➔ Acho que a heterogeneidade de dispositivos pode crescer independente se existe ou não padronização de tecnologias. Talvez eu tenha entendido errado..

comunicação com outros dispositivos, manifestam o conceito de estarem em rede, como discutido anteriormente. Ao passo que estes objetos se conectam com a Internet serão **conectados** considerados como parte da verdadeira Internet das Coisas. talvez não precise dessa "verdadeira" :)

A partir desta discussão inicial é possível enfim dar os primeiros passos em direção a entendimentos mais profundos sobre a IoT. Em [CASAGRAS 2009], são apresentados dois dos principais atores da IoT sendo eles: i) *Usuários*: os quais podem ser pessoas ou ativos digitais (ex: aplicativos, *software* agente, serviços) que tem por objetivo interagir com algum ambiente, sendo que esta interação se dá por meio da IoT; ii) *Entidades Físicas (EF)*: é a parte do ambiente físico em que os usuários possuem intenção de interagir para realizar suas tarefas. As EF podem ser qualquer objeto ("coisa") variando de objetos vivos (pessoas ou animais) até não vivos (carros, geladeira, mesa e outros). As EF necessitam de uma representação no mundo virtual, ou seja, uma *Entidade Virtual (EV)* que pode ser um modelo, um objeto de uma classe em uma linguagem de programação, uma conta em uma rede social entre outros modelos de representação. Além disso, as EV precisam ser associadas a uma EF exercendo assim uma relação entre os dois mundos.

Os objetos inteligentes possuem características e recursos necessários para monitorar ambientes de interesse dos usuários. Por esse motivo, os objetos inteligentes são os principais responsáveis por mediar as interações entre usuários, entidades físicas e suas representações digitais. Para prover tal mediação, os objetos inteligentes precisam ter competências nos dois mundos (físico e virtual), deste modo os recursos dos objetos são as ferramentas adequadas para estes requisitos. Em outras palavras, os objetos possuem: capacidades de monitoramento do ambiente através de sensores ou atuadores, poder de processamento, armazenamento e comunicação. A seguir será apresentada um breve **arcbouço histórico da IoT**.

1.1.2. Perspectiva histórica da IoT

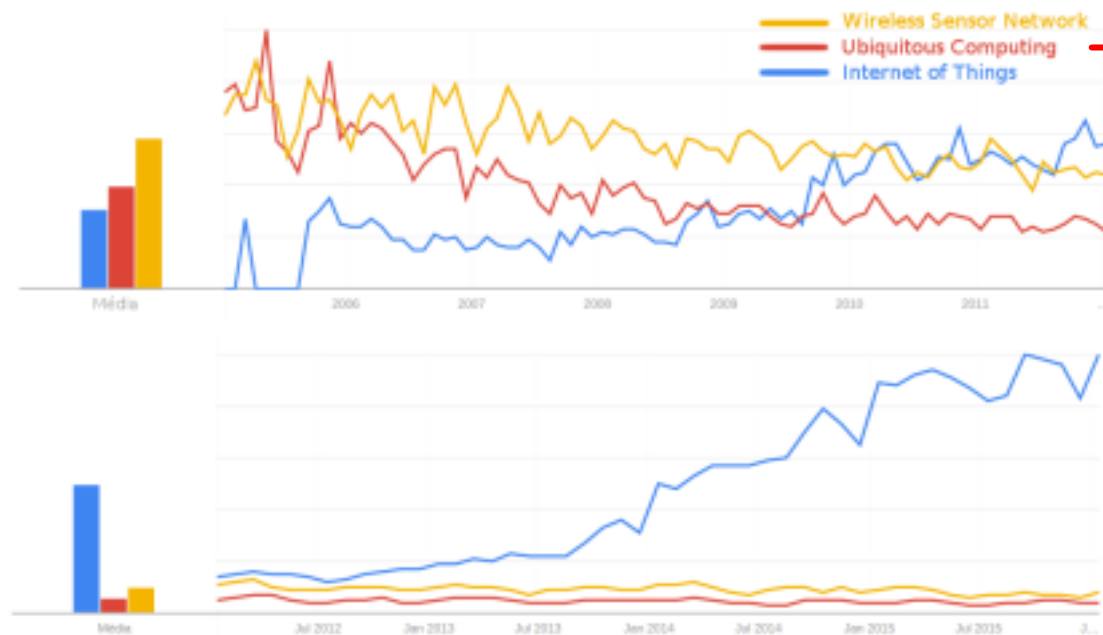
Kevin Ashton foi o primeiro a utilizar o termo *Internet of Things* em 1999 [Ashton 2009]. Na época, a IoT era altamente relacionada ao uso da tecnologia RFID e os principais consumidores da novidade eram empresas de logística e gestores de grandes quantidades de produtos. Como observado na Figura 1.1 superior, em 2005, surgem as Redes de Sensores Sem Fio (RSSF) que trazem avanços na automação residencial e industrial [Kelly et al. 2013] [Da Xu et al. 2014], bem como técnicas para explorar as diferentes limitações dos dispositivos (ex: memória e energia) e escalabilidade e robustez da rede. Naquele momento, objetos inteligentes já estavam espalhados em um ambiente para monitorar variáveis de interesse ou automatizar tarefas. Intuitivamente o próximo passo era preencher a lacuna entre RSSF e a Internet, a este estágio é dado o nome Sensor Web (SW). Projetos como [Microsoft 2015] [Delin 2002] são exemplos de SW, os quais possuem como principal característica a localização geográfica dos dispositivos e a vinculação entre os dados monitorados à web.

O surgimento da Web 2.0 é a próxima etapa da história que vale destacar. Na "segunda geração" da web não ocorrem modificações nas técnicas ou protocolos, mas sim olhar para a web como uma plataforma, um ambiente de interação e participação. Este marco fez com que inevitavelmente, a IoT também se transformasse. Emergiu o conceito de Web das Coisas (do inglês *Web of Things* - WoT), o qual é caracterizado pelos fato dos

Não entendi essa parte. A IoT já tinha sido definida e a web 2.0 a transformou?

Essas datas estão corretas?

imagino que tenha sido antes.



→ Por que computação ubíqua? Talvez os outros termos do texto, tipo web of things, sensor web..

Figura 1.1. Número de pesquisas no google sobre *Wireless Sensor Networks*, *Ubiquitous Computing* e *Internet of Things*.

dispositivos inteligentes poderem compartilhar informações com a Web. As buscas sobre IoT e temas relacionados dispararam por volta de 2012 como mostrado na Figura 1.1 inferior. Segundo [Wang et al. 2015], essa fase é caracterizada pela WoT empregar *gateways* executando um servidor Web, que é utilizado para prover acesso os dispositivos. É possível notar ainda, a popularização de *smartphones* que alavancou os estudos sobre computação móvel, bem como também a computação em nuvem. Atualmente a definição da IoT ainda não é exata e assim, diferentes autores apresentam diferentes perspectivas (veja [Pretz 2013] [Borgia 2014] [Al-Fuqaha et al. 2015]). Ainda assim, é possível notar que hoje é vivenciada a fase mais abrangente da IoT. Os objetos inteligentes estão cada vez mais populares e mais conectados entre si e com a Internet, além disso podem acessar e prover os serviços. A partir daí, novas aplicações serão uma consequência natural. Todavia, ainda existem questões a serem resolvidas tais como: explorar ainda mais a computação móvel, localização, técnicas de fusão de dados, análises preditivas sobre os dados oriundos da IoT para inferir sobre o contexto dos objetos e outros.

A IoT foi identificada como um tecnologia emergente em 2012 por especialistas da área [Gartner 2015]. A Figura 1.2 apresenta um *Hype Cycle* que é uma maneira de representar o surgimento, adoção, maturidade e impacto de diversas tecnologias. 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 tanto no âmbito acadêmico quando industrial. Também pode-se notar o surgimento das primeiras plataformas da IoT (discutidas mais adiante) que possuem expectativa de alta nos próximos anos. As setas em destacam na Figura 1.2 os momentos aqui comentados. Isto serve 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.

Talvez fosse importante diferenciá-las e definir qual se aproxima deste minicurso.

1.1.3. Motivação para manutenção da evolução da IoT



Figura 1.2. Tecnologias emergentes.

tras [Sundmaecker 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 tais como: 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 ao passo que mantém o baixo uso dos recursos limitados. Deste modo, fica evidente a necessidade da adaptação dos protocolos existentes (por exemplo o IP). Além disso, os paradigmas de comunicação e roteamento devem ser explorados para permitir prover os serviços fundamentais para formar novas aplicações.

Novos desafios surgem ao passo que 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 podemos empregar esses dados em determinados cenários. Deste modo, os desafios impostos por essas novas aplicações sobre a IoT devem ser explorados e soluções devem ser propostas. 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. Assim, uma discussão sobre o impacto da IoT na computação ubíqua é de grande importância, sendo assim esta discussão ser[á retomada nas próximas seções.

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 ~~(D2D)~~ e máquinas ~~(M2M)~~. Com isto emerge uma nova gama de aplicações, tais como coleta de dados de pacientes e monitoramento de idosos (*healthcare at home*), sensoria-mento de ambientes de difícil acesso e inóspitos, entre ou-

1.1.4. Objetivos do minicurso

Este capítulo tem por apresentar ao leitor uma visão geral da Internet das Coisas. Para tanto, se faz necessário conhecer os blocos básicos de construção deste novo conceito de redes. A seguir serão listados blocos básicos de construção dos quais alguns serão objetos de discussão ao longo do capítulo. 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 blocos na IoT.

Na Figura 1.3 são apresentados os blocos básicos de construção da IoT sendo eles:

I) **Identificação**: este bloco é de grande importância, visto que é primordial identificar os objetos inteligentes para então anexá-los à Internet. Existem basicamente dois principais tipos de identificação: 1) código de identificação ID (geralmente um inteiro de 32 bits) e 2) endereçamento IP geralmente empregado em conjunto com técnicas de compressão (assunto tratado mais adiante no texto); II) **Sensores**: existem diferentes tipos sensores, por exemplo, os de temperatura, umidade, atuadores, sensores embutido em vestimentas, sensores virtuais (aqueles que são combinações de sensores reais¹). Os objetos inteligentes coletam dados do seu contexto através dos sensores e, em geral, enviam as informações para estações de tratamento ou armazenamento de dados tais como banco de dados, a nuvem ou mesmo *data warehouse*; III) **Comunicação**: a comunicação também é um dos blocos fundamentais, pois através dela é possível conectar objetos inteligentes ~~heterogêneos~~. Além disso, este bloco desempenha papel importante no consumo de energia dos objetos, sendo portanto um fator 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;

os. Acho que a comunicação é fundamental para conectar heterogêneos ou não.



Figura 1.3. Blocos básicos da IoT.

IV) **Computação**: diz respeito a unidade de processamento, por exemplo, micro-controladores, processadores, FPGAs. Em outras palavras, tudo o que dá aos objetos inteligentes a capacidade de computar; V) **Serviços**: a IoT pode prover diversos serviços, dentre eles, destacam-se, por exemplo, *serviço de identificação*, isto é, prover acesso ao mapeamento entre uma entidade física e virtual. O *serviço de fusão de informações* sumariza os dados dos sensores para serem processados. Para finalizar, o *serviço de ubiquidade* permite a tomada de decisões e a reação adequada em face dos dados obtidos dos sensores ou serviços de fusão de dados; VI) **Semântica**: refere-se a habilidade de extração de conhecimento da diversidade de objetos na

IoT. Em outras palavras, a semântica trata da descoberta e uso inteligente dos recursos,

¹Um sensor de fogo físico é inexistente. O que se pode fazer é construir um sensor virtual através da combinação de sensores existentes, por exemplo, o de temperatura e concentração de CO₂.

para possibilitar o provimento de determinado serviço. Além disso, deve efetuar o reconhecimento e análise dos dados para a 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)*.

Após a breve apresentação dos blocos da IoT fica evidente o quão ampla é a área. Diante disto, serão discutidos somente os alguns destes blocos ao longo do capítulo sendo eles: **Identificação, Comunicação e os Serviços**. Sempre pontuando em momento oportuno questões que envolvem os demais blocos.

Por que foram escolhidos estes?

1.1.5. Desafios: uma breve provocação

Diversos desafios surgem no contexto da IoT. Ao nível de objetos inteligentes pode-se destacar os desafios do consumo de energia, tamanho físico dos objetos e a limitada quantidade de memória. Para exemplificar, o consumo de energia esta relacionado aos padrões de comunicação da rede, sendo assim como devem ser exploradas essa característica? Ainda sobre energia, como explorar a abundante quantidade de energia do ambiente (solar, cinética, eólica e outras) para fazer uso nos objetos inteligentes (IoT Energy Harvesting)?² Ao nível de *software* para os objetos, pode-se destacar a necessidade de adaptação dos protocolos existem para operar em severas limitações de memória e processamento.

Ao nível mais alto o de aplicações e serviços, destaca-se os problemas referentes aos dados oriundos da IoT. Estes dados apresentam diversas características peculiares tais como ambiguidade, são apresentados fora de ordem, duplicados ou ainda esparsos. Diante deste cenário, quais são as técnicas para lidar com esses problemas?

Desafios tal como os acima citados serão objetos de discussão ao longo do capítulo, sempre pontuando soluções totais ou parciais existentes na literatura.

1.1.6. Estrutura do minicurso

Na Seção **1.2** serão levantadas discussões sobre os objetos inteligentes e as principais tecnologias de comunicação utilizadas para interligar os objetos. Ao final da seção, será levantada um dos maiores desafios dos objetos da IoT que é a energia e a tecnologia *Energy Harvesting*. A Seção **1.3** abordará os blocos básicos diretamente relacionados a *software* tais como: Identificação e Serviços. Os quesitos de endereçamento, padrões de comunicação, arquitetura e outros são temas da seção que ao final chamará a atenção do leitor para o problema do *gateway*. Em seguida, na Seção **1.4**, uma abordagem prática será oferecida, para consolidar os conhecimentos das seções anteriores. Logo após, na Seção **1.5**, questões sobre os blocos básicos de Serviços e Semântica serão elencados e comentados, bem como algumas questões de pesquisa, por exemplo, a fusão e análise de dados adquiridos dos objetos IoT. A Seção **1.6** dará foco na perspectiva futura da IoT, apresentando um olhar diferenciado para a Internet das Coisas, em que a IoT é um meio para se atingir a Computação Ubíqua. Finalmente, na Seção **1.7**, serão realizadas as considerações finais e destacados os pontos que não foram cobertos no capítulo, mas fazendo referências que podem servir de ponteiro para leitores mais curiosos.

²*IoT Energy Harvesting* é a tecnologia que captura energia do ambiente para que objetos inteligentes possam operar de modo autônomo.

Será que tem como aumentar este parágrafo?

1.2. Dos Dispositivos e Tecnologias de Comunicação **LUCAS / LOUREIRO** (MAX 7 PG)

1.2.1. História dos objetos inteligentes **LUCAS**

Nesta seção é abordado os primórdios da Internet das Coisas (IoT), como surgiram e as principais tecnologias adotadas para sua implementação. No início as redes de computadores eram compostas apenas por poucas máquinas, presentes nas universidades ou instituições militares dos EUA. A única forma de conectar estes computadores era utilizando cabos.

← O que acha de agregar esses parágrafos?

Com os avanços tecnológicos dos últimos anos, hoje é possível se conectar à Internet de qualquer parte do mundo. O próximo passo é conectar qualquer dispositivo à Internet, criando indústrias completamente automatizadas, casas e até cidades. Este fenômeno é chamado de Internet das Coisas (IoT).

A definição de Internet das Coisas segundo o Cisco Internet Business Solutions Group (IBSG) é: IoT é simplesmente o momento em que existem mais coisas à Internet do que pessoas [Evans 2011]. Desde então a ideia se espalhou e existem uma grande série de aparelhos conectados à Internet como smartphones, tablets, smartwatches, dispositivos vestíveis (wearables) entre outros. Conceitos de novas aplicações como os dispositivos vestíveis foram introduzidos.

Redes de Sensores sem Fio eram anteriormente utilizadas para obter dados de um certo local. Como por exemplo temperatura, quantidade de Co2, umidade. Cada rede de sensores possuía a sua aplicação e sua motivação para monitorar um certo ambiente.

Com a evolução para Internet das Coisas a capacidade de monitorar ambientes ficou integrada à Internet. Smartphones e dispositivos vestíveis podem ser interpretados como sensores. A partir disso é possível coletar dados baseados nas experiências dos usuários, a utilização dos dispositivos e realizar a coleta de dados para obtenção de informações.

Um novo problema pode ser identificado é como gerenciar essa quantidade de dados gerada pelos sensores, antigamente, eram dados selecionados devido a Rede de Sensores sem Fio possuir uma aplicação bem definida. Com os inúmeros sensores espalhados pelo mundo, existem muitos dados, e o novo desafio é filtrar, tratar, gerenciar essa quantidade de dados enorme e retirar informações relevantes para aplicação.

1.2.2. Arquiteturas básica dos dispositivos

A arquitetura de sistema de um objeto inteligente é composta de cinco blocos principais: 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

Eu removeria essa sentença em destaque ou a deslocaria para a introdução da seção. Assim, eu começaria a subseção "No início as redes de..."

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.

1.2.3. Tecnologias de comunicação LUCAS

1.2.4. Tecnologias de comunicação

1.2.5. Ethernet

1.2.6. Wi-Fi

Hoje em dia, formas de comunicação sem fio se tornaram ubíquas. Elas estão presentes em quase todos os lugares, fazendo parte do nosso cotidiano nas casas, escritórios, indústrias e em até espaços públicos pela cidade. Uma tecnologia sem fio bastante conhecida é o padrão IEEE 802.11, também chamado de Wi-Fi, que hoje está 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, 54 Mbps foi alcançado nos anos seguintes. Hoje 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, tendo pouco, ou talvez nenhuma preocupação com dispositivos que possuem consumo energético limitado, como os utilizados em aplicações IoT. Portanto, não é esperado que muitos dispositivos utilizados em IoT adotem o padrão Wi-Fi como principal protocolo de comunicação. Contudo, Wi-Fi possui suas vantagens, como alcance de conexão e vazão, o que o torna adequado para navegação na Internet em dispositivos móveis, como smartphones e tablets. Esta utilização do Wi-Fi em dispositivos móveis permite a integração do Wi-Fi com outros protocolos de comunicação, permitindo a comunicação de Wi-Fi com aplicações IoT, ao mesmo tempo que as redes IoT continuam com o baixo consumo energético.

1.2.7. ZigBee

1.2.8. Bluetooth Low Energy

Bluetooth é um protocolo de comunicação inicialmente criado pela Ericsson para substituir a comunicação serial RS-232. Hoje, o órgão Bluetooth Special Interest Group (SIG) gerencia a tecnologia, sendo responsável por criar, testar e manter a tecnologia e os seus protocolos. 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.

A tecnologia Bluetooth se divide em: Bluetooth Clássico (ou BR/EDR), versão abaixo da 2.0; Bluetooth High Speed (ou HS), versão 3.0; e o Bluetooth Low Energy, versão acima da 4.0. No passado, as melhorias no Bluetooth eram focadas em aumentar a vazão, o que criou um protocolo complexo, o que fez o Bluetooth inadequado para dispositivos com consumo energético limitado. Indo contra as versões anteriores do Bluetooth, o Bluetooth Low Energy (BLE) possui especificação voltada para baixo consumo energético, permitindo dispositivos que usam baterias do tamanho de moedas (coin cell batteries) durarem anos.

Diferente do Bluetooth Clássico, BLE foi principalmente desenvolvido para transferir pequenas quantidade de dados, como dados que representam estado de sensores, fazendo com que o BLE não seja adequado para transferir grande quantidade de dados como vídeos ou áudio. Com essa limitação, o limite teórico máximo para a vazão do BLE é de 1 Mbps. Em comparação com a versão BR/EDR que possui a vazão de 3 Mbps e a versão HS possui 24 Mbps. Outro ponto importante para ser ressaltado é a falta de retrocompatibilidade entre as versões clássicas do Bluetooth com a versão BLE. Contudo, a especificação BLE define dispositivos capazes de implementar o Bluetooth Clássico e o BLE, permitindo a compatibilidade com diferentes versões do Bluetooth.

Hoje 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 MTU (maximum transmit unit) 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, qualquer dispositivo é capaz de atuar como Master ou Slave ao mesmo tempo, permitindo a criação de topologias em malha (Mesh). Atualmente cerca de 96% dos smartphones estão equipados com Bluetooth Low Energy [Woolley 2014].

1.2.9. 3G e 4G

1.2.10. LoraWan

1.2.11. Sigfox

1.2.12. Tabela Comparativa

1.2.13. Dispositivos: limitações, desafios e perspectivas LOUREIRO

Diante do acima mencionado fica claro que os objetos inteligentes apresentam limitações. A tendência dos “*Microelectromechanical systems - (MEMS)*” de tornar os dispositivos ainda menores (nano-escala) já encontra-se em andamento [REF]. Sendo assim, as restrições impostas pelos dispositivos variam desde de capacidades computacionais e comunicação até de energia (quando alimentados através de bateria). **Desenvolver +**

A poucos anos atrás era possível apontar o fator custo como limitante para adoção e desenvolvimento dos objetos inteligentes. Entretanto, hoje existem diversos produtos disponíveis no mercado com custo regular tais como: **listar produtos**. 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, modelo mais atual, custa 35 dólares.

Energia é um dos mais limitantes e desafiadores quesitos dos objetos inteligentes. Limitante no sentido de que dispositivos alimentados por baterias podem está em locais inóspitos ou as baterias não podem ser substituídas com facilidade. Nesta situação, tanto componentes de *hardware* quanto de *software* devem ser construídos de modo consciente, deste modo o consumo deve ser o mínimo consumo possível para estender ao máximo o tempo de atividade do dispositivo. A energia é desafiadora no sentido de que a rede de objetos inteligentes deve está consciente que a interrupção de um serviço provido por

dispositivos, por falta de energia, podem causar impactos de diferentes proporções. Por exemplo, a interrupção do serviço de temperatura do ambiente para estudo de uma disciplina sobre IoT pode não ser tão impactante, entretanto a suspensão de uma aplicação IoT *Health Care* pode ser vital para o usuário monitorado.

Uma possível estratégia para mitigar este problema é a técnica de colheita de energia (do inglês *Energy Harvesting*). 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 devido as limitações físicas dos dispositivos) é armazenada e deve ser utilizada para suprir as necessidades dos objetos como comunicação e processamento. Contudo, a proposta de solução também trás ao menos um outro desafios 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 está inserido 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 adquirida do ambiente.

- Limitações (processamento, memória, energia)
- Custo, qualidade do HW, tamanho e outros...
- Levantar a discussão sobre conhecimento prévio das redes de computadores (tanto de HW quanto de SW) e como devemos adaptá-los para esse novo mundo. O foco maior deve ser dado nas limitações do HW
- Energia como um grande desafio

– Energy Harvesting

- * O que é? Como fazer? Quais as direções?
- * Exemplo da reunião: Dado que temos os dispositivos com capacidade de adquirir energia do ambiente e armazenar. Como programar as atividades que o dispositivo deve desempenhar dado o orçamento de energia (*energy budget*), isto é, como gastar a energia em função das atividades que se deve fazer?

1.3. Da Teoria aos Softwares e Ambientes de Desenvolvimentos para IoT **BRUNO** **/ LUCAS / BRUNA (MAX 15 pg)**

Esta seção aborda assuntos referentes a camada de *software* que orquestra a lógica de operação dos objetos inteligentes, além disso serão pontuados os sistemas operacionais mais utilizados para IoT, bem como os principais ambientes de 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. O Software para redes convencionais e para rede de objetos inteligentes. **BRUNO**

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, muitos pesquisadores da área notam que uso dos protocolos da Internet (TCP/IP), sem modificações, é impraticável nos dispositivos com recursos limitados, os quais 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 ser ajustada 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. 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 diminuiu 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 (do inglês *Low-Power and Lossy Networks (LLN)*) relacionadas a seguir:

1. **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 a camada física e efetua o controle de acesso para redes sem fio pessoais

Pq para
redes convencionais?

Fiquei
em dúvida
o que seria
essa maior
liberdade de
desenvolvimento.

de baixas taxas potência de transmissão.

2. **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)*. O RPL é o protocolo estado-da-arte para IoT, o qual constrói uma topologia robusta com mínimo de estados necessários.

1.3.2. Paradigmas de comunicação sobre redes de dispositivos inteligentes **BRUNO/-BRUNA**

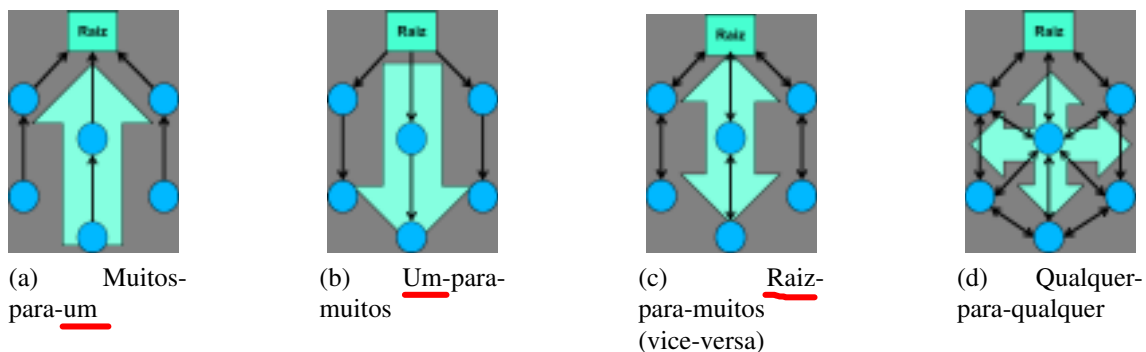


Figura 1.4. Paradigmas de comunicação que ocorrem nas redes de objetos inteligentes.

Os padrões de comunicação que ocorrem em uma rede de objetos inteligentes podem ser divididos em quatro paradigmas como mostrado na Figura 1.4. Os paradigmas diferem significativamente, sendo assim o modo de comunicação pode acarretar em maior ou menor impacto no uso dos recursos, em especial de memória e energia, e na viabilidade de aplicações sobre a rede. A seguir cada paradigma será detalhado:

- **Muitos-para-Um (*Many-to-One*):** em que os objetos inteligentes reportam informações, as quais são coletadas por estações base (Raiz na Figura 1.4). Este paradigma é conhecido como coleta de dados, sendo o mais comum dos paradigmas, visto que atende as demandas de comunicação de muitas aplicações. Para realizar a coleta de dados cria-se uma árvore de roteamento com raiz nas estações bases. 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 as raízes e os objetos na rede;
- **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

Aqui não ficou claro se o "Um" é a raiz ou qualquer nó na rede. Acho que os labels ajudaram isso

confirmar se os dados enviados foram entregues tal como ocorre na coleta de dados. O custo em número de mensagens também pode ser alto, caso sejam realizadas diversas inundações na rede;

- Raiz-para-muitos (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.
- Qualquer-para-Qualquer (*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.

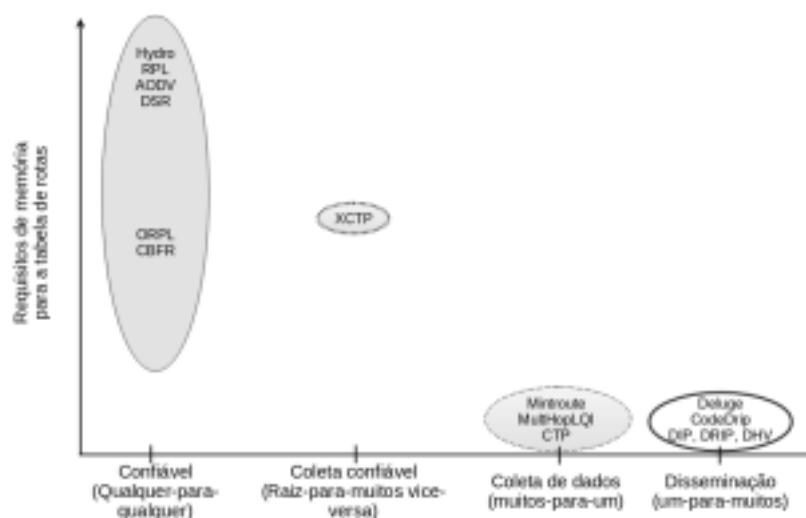


Figura 1.5. Comparativo entre os protocolos de roteamento.

Na Figura 1.5 são apresentados protocolos de roteamento classificados pelo paradigma que implementam e comparados, de modo qualitativo, com base no custo de memória para sua operação. Dos protocolos exibidos na Figura 1.5 Directed Diffusion [Chalermek et al. 2003], (DIP, DRIP e DHV) [Levis et al. 2005b] são para disseminação de pequenas quantidade de dados na rede. DIP, DRIP e DHV se baseiam no algoritmo Trickle [Levis et al. 2003b] para manter as rotas consistentes. DRIP trata cada informação como uma entidade separada, o que permite mais controle sobre como e o quão rápido os dados serão disseminados. Já DIP e DHV tratam os dados como um grupo, o que significa que os parâmetros de controle e disseminação são aplicados igualmente sobre os dados. Deluge é um protocolo de disseminação para grandes quantidades de dados, para tanto faz uso da métrica *Expected Transmission count (ETX)* [Javaid et al. 2009]

para encontrar rotas de alta qualidade. O protocolo CodeDRIP [Nildo et al. 2014] faz uso do *network coding*, uma sofisticada técnica, para recuperar pacotes perdidos através da combinação dos que foram recebidos.

Dos protocolos de coleta de dados, o *Collection Tree Protocol (CTP)* se destaca, pois apresenta resultados melhores que *MultiHopLQI* e *MintRoute* como argumentado pelos autores do CTP [Gnawali et al. 2009]. O CTP também emprega a métrica ETX e Trickle para criar e manter as rotas de qualidade a baixo custo. Atualmente, o único protocolo que contempla o paradigma raiz-para-muitos e vice-versa é o eXtend CTP [Santos et al. 2015a]. Os autores argumentam que o XCTP é uma extensão do CTP. O protocolo expande a gama de aplicações do CTP por permitir rotas reversas entre a raiz e os objetos da rede. XCTP realiza tal roteamento acrescentando pouca quantidade de estados quando comparado com CTP e significativamente menor quantidade de estados quando comparado com protocolos da categoria qualquer-para-qualquer.

O protocolo Hydro [Dawson-Haggerty et al. 2010] visa manter rotas de todos para todos os objetos da rede. Hydro se mostra robusto em casos de falhas na rede e apresenta uma sofisticada técnica para a construção das tabelas de rotas. Entretanto, este protocolo supõe que existam dispositivo com recursos não limitados e conexão consiste entre eles. *Ad hoc On-Demand Distance Vector Routing (AODV)* [Perkins et al. 2003] e *Dynamic Source Routing (DSR)* [Johnson 2003] os primeiros e famosos protocolos deste paradigma. Eles realizam inundações na rede com mensagens RREQ (requisição) para construir as tabelas de rotas de todos para todos. A principal diferença entre os dois é que DSR usa o cabeçalho do pacote para armazenar o caminho entre dois elementos na rede, enquanto o AODV mantém os pacotes enxutos, mas necessita de grande quantidade de memória para a tabela de rotas. CBFR é um protocolo de roteamento que se baseia em protocolos de coleta para permitir a comunicação ponto-a-ponto. Para isso, cada nó na árvore de coleta armazena os endereços de seus nós filho diretos e indiretos. Como a memória é um recurso escasso, estes dados são armazenados em filtros Bloom, uma estrutura de dados eficiente em relação ao consumo de espaço [Reinhardt et al. 2012]. ORPL é um protocolo que traz o encaminhamento oportunista para RPL, com objetivo de alcançar baixa latência, comunicação confiável em redes com ciclo de trabalho [Duquennoy et al. 2013]. Para encaminhar para cima, o ORPL utiliza *anycast*, o que faz com que os não precisam escolher um próximo salto e, então, não precisam das tradicionais tabelas de roteamento. O ORPL suporta tráfego qualquer-para-qualquer de modo que primeiramente encaminha a mensagem em direção à raiz até qualquer antecessor comum e então a mensagem é redirecionada para baixo até o destinatário. ORPL usa a mesma estrutura de dados para armazenar as informações de rota como CBFR, um filtro de Bloom. No entanto, no ORPL, em vez de escolher um próximo salto, os nós realizam *anycast* de pacotes. Dessa forma, ao receber um pacote, um nó decide transmiti-lo ou não, e envia uma confirmação apenas se optar por atuar como próximo salto. A seguir é dado enforque ao protocolo estado-da-arte para IoT.

Só tem uma subseção, talvez pudesse deixar 1.3.2 mesmo



1.3.2.1. IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) **BRUNA**

O RPL é um protocolo de roteamento para redes sem fio de baixa potência e com perdas que utiliza endereços IPv6. Dispositivos de rede executando o protocolo estão conectados de maneira acíclica, por meio de um Grafo Acíclico Direcionado Orientado ao Destino (DODAG). O grafo é construído com o uso de uma função objetivo (OF) que define como as métricas de roteamento são computadas de acordo com as restrições (usando o número esperado de transmissões (ETX) ou a quantidade atual de energia da bateria de um nó, por exemplo). Para iniciar o processo de construção do DAG a raiz anuncia informações sobre o grafo por meio de mensagens. Os nós vizinhos à raiz receberão essa mensagem e devem tomar decisões baseadas em sua OF. Uma vez que o nó se junta ao grafo, ele escolhe a raiz como seu pai e calcula seu *rank*. O *rank* é uma métrica que indica as coordenadas do nó na hierarquia da rede [Vasseur et al. 2011]. Os demais nós irão repetir esse processo de seleção do pai e notificação de informação do DAG, por meio de DIOS. Quando esse processo se estabiliza, a coleta de dados pode começar. O protocolo especifica dois modos de operação para o roteamento descendente: *storing* e *non-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. No modo *non-storing* cada nó gera e envia mensagens para a raiz do DODAG. 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*).

1.3.3. Arquitetura TCP-UDP/IP para IoT. Ou o que não pôde ser reutilizado talvez possa ser adaptado **LUCAS**

1. IP para Objetos Inteligentes? (Arquitetura TCP-UDP/6LoWPAN)
2. Adaptações do IPv6 para chegar ao 6LoWPAN
3. Pilhas TCP-UDP/6LoWPAN reduzidas
 - μ IP e lwIP

1.3.4. Modelos de conectividade Redes de Objetos inteligentes X IoT **BRUNO**

Até o momento foram discutidas questões sobre os paradigmas de comunicação, os protocolos de roteamento que implementam tais paradigmas, bem como a pilha de protocolos TCP/IP utilizada na IoT de hoje.

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 inteligente 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 grande importância, pois inevitavelmente estes dois quesitos serão as chaves para construir novas aplicações sobre a IoT.

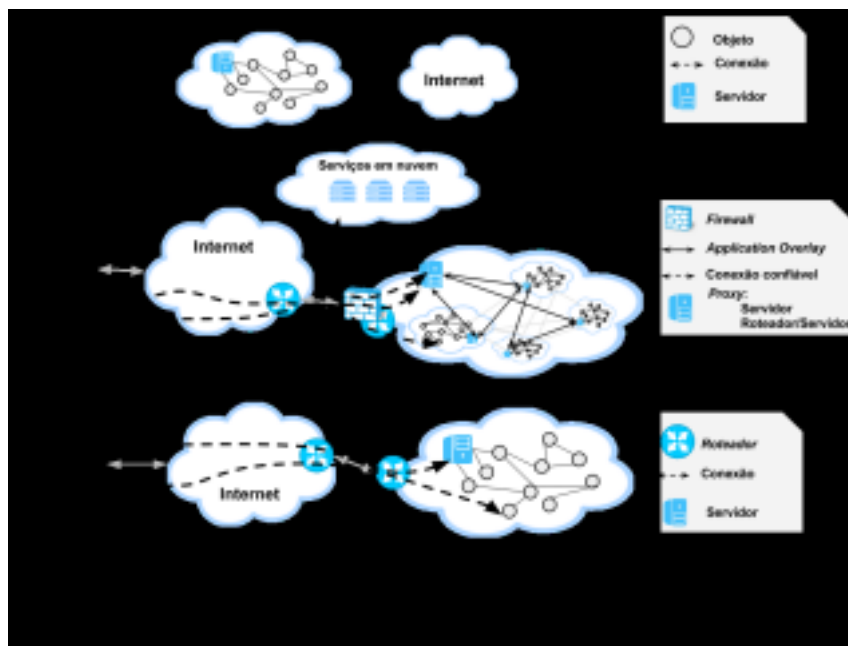


Figura 1.6. Modelos de conectividade dos objetos inteligentes. (I) Rede autônoma em que objetos inteligentes não possuem conexão com a Internet pública; (II) Rede de objetos inteligentes limitada, pois o acesso aos dispositivos é restrito; (III) IoT autêntica em que os objetos estão conectados a Internet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

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 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. Na Figura 1.6 (I) apresenta uma abordagem esquemática deste modelo de conectividade.

Uma questão pode ser levantada aqui é: “Neste modelo de conectividade é necessário que os objetos inteligentes sejam endereçados com IP?” Um argumento válido como resposta para esta questão é que o espaço de endereçamento provido pelo IPv6 pode ser bastante adequado, mesmo para redes sem conexão com a Internet. Também é válido alegar que ao usar IP neste modelo a rede manterá compatibilidade com a arquitetura que está em desenvolvimento pelos grupos RoLL e 6LoWPAN.

- **Internet Estendida:** o termo “Internet Estendida” refere-se ao posicionamento “central” deste modelo de conectividade em relação a 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]. É notório que a evolução das cidades inteligentes produzirá 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 a rede de objetos se dá via *firewall*, que por sua vez tem como tarefa controlar o acesso de modo seguro as redes de objetos privadas (IP). Deste modo, este modelo de conectividade “estende” a Internet por permitir o acesso aos objetos inteligentes que até então estavam isolados.

- **Internet of Things:** 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 atualmente se dá ou diretamente conectando-se com o objeto ou através de servidores que coletam informações dos dispositivos. Nesta segunda modalidade, a Internet conecta o usuário ao servidor e assim, técnicas de processamento dentro da rede (*in-network processing*) podem ser utilizadas para preservar os recursos limitados dos dispositivos.

1.3.4.1. Discussão sobre os uso de *Proxies* nos modelos de conectividade

Sem dúvidas o modelo de conectividade apresentado na Figura 1.6 (II) é o que levanta maior discussão. Porque o modelo necessita da existência de *proxies* que, neste caso, são roteadores ou computadores capazes de realizar processamento ao nível de aplicações para melhorar a escalabilidade e eficiência da Internet Estendida.

O modelo faz uso de *proxies* em detrimento dos *gateways*, para manter o princípio fim-a-fim da arquitetura da Internet. Com os *proxies* ao serem realizadas requisições não é necessário traduzir o IPv6 ao longo logo do caminho. Portanto, o IP é usado realmente de modo fim-a-fim, além disso assegura-se o suporte a *Quality of Service (QoS)*, gerenciamento, roteamento e demais funcionalidades da arquitetura da Internet. Sendo assim, o uso dos *proxies* pode ser realmente desejáveis em algumas situações, por exemplo, estes elementos podem realizar tarefas de coleta, agregação, pré-processamento ou *cache*. Considere ainda o caso em que uma rede com centenas de milhares de objetos, o uso de *proxies* pode mitigar problemas de consumo de energia próximo de uma estação base única (*Energy Hole problem*) nos casos em que os objetos usam baterias, bem como aumentar a escalabilidade e minimizar o gargalo de tráfego na estação base. Neste sentido, surge a ideia de aplicações em camadas (*applications overlay*) como mostrado na Figura 1.6 (II) que consiste em inserir elementos (*proxy*) interconectados para permitir *in-network processing*. Deste modo, é possível distribuir a carga de processamento e tráfego, bem como melhora a escalabilidade.

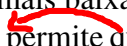
1.3.5. Ambientes de desenvolvimento **BRUNA**

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].

1.3.5.1. 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, o sistema operacional para esses objetos devem ser muito menores e consumir menos recursos. Nesta Seção, serão descritos brevemente os dois principais sistemas operacionais para objetos inteligentes: Contiki e TinyOS, além do sistema operacional Android que opera em grande parte dos dispositivos de sensoriamento participativo (*smartphones*). Ambos sistemas operacionais são código aberto e os respectivos códigos fonte podem ser encontrados na Web.

1. **Contiki:** 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 uIP TCP/IP e, em 2008, o sistema incorporou o uIPv6, 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 portátil [Dunkels et al. 2004].
2. **TinyOS:** 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. 2005a] 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:** Android é uma plataforma de software de código aberto para dispositivos móveis que inclui um sistema operacional, middleware e aplicativos [Developers 2011]. 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  permite que ao dispositivo manusear diferentes tipos de dados. Na mesma camada que as bibliotecas está também o *Android Runtime*,

Talvez seja
melhor
Google 2011

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 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.

1.3.5.2. 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*). 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:

- **A lista não apresenta nenhum EMULADOR**
- **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 scripts 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 (Contiki OS Java) 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 [Coo 2015] [citação estranha]. 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. 2003a].
- **OMNet++/Castalia:** OMNeT++ é um simulador de eventos discretos baseado em C++ para modelar redes de comunicação, multiprocessadores e outros sistemas distribuídos 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 (Simulator for Network Algorithms) 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 visão 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. Além disso, 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* [Group 2011].
- **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 [iot 2015].
- **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].

caberia uma tabela comparativa dos Emuladores X Simuladores

Não vi a referência
a tabela no texto.
Pode ter passado

Tabela 1.1. Comparação entre simuladores

Nome	Suporte GUI	Licença	Linguagem	Sistema Operacional
ns-2	Não	Código aberto	C++ e oTcl	GNU/Linux, FreeBSD, Mac OS, Windows
ns-3	Limitado	Código aberto	C++ e python	GNU/Linux, FreeBSD, Mac OS, 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++	Windows, Linux, Mac OS
Castalia	Sim	Código aberto	C++	Linux, Windows
Sinalgo	Sim	Código aberto	Java	Linux, Windows, Mac OS (?)

Acho que podemos alterar o nome dessa subseção

1.3.6. SW suas limitações e desafios que geram LUCAS/BRUNA

- Problema do Gateway

Os Gateways surgiram há cerca de duas décadas com a necessidade de interconectar redes IP, permitindo a comunicação entre diferentes arquiteturas e ambientes. Eles oferecem uma alternativa de adaptar o princípio fim a fim do IP, fazendo com que objetos inteligentes não baseados em IP possam se conectar em uma rede IP [Vasseur and Dunkels 2010]. Entretanto, existem alguns motivos para se evitar o uso dos gateways, que são tratados a seguir.

No texto, não vi referência a inteligência. Acho que o texto ainda está em edição, neh?

– Onde a “inteligência” deve ficar?

No estado da arte atual, é necessário a presença de um gateway de camada de aplicação, tanto no software quanto no hardware, que forneça conectividade de aplicação específica para os dispositivos da Internet das Coisas. Em [Zachariah et al. 2015], os autores comparam essa necessidade à utilização de um novo *web browser* para cada página da internet, o que é difícil de se imaginar quando se considera a escalabilidade da Internet das Coisas.

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, gateways de tradução de protocolo não escalam e se tornam gargalos da rede. Esses gateways também impactam na confiabilidade geral da rede pontos únicos de falha [Vasseur and Dunkels 2010].

Segundo [Zachariah et al. 2015], esse problema de gateway existe pois os gateways atuais unem as tarefas de conectividade de rede, processamento interno da rede e funções de interface de usuário. Para solucionar tal problema, é proposta uma solução que separa essas funções, com o objetivo de melhorar a conectividade dos dispositivos da IoT. Para isso, eles 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.

* **Se no Gateway outras questões surgem: se a conexão for perdida? e se for uma queda temporária? como implementar confirmações entre os dispositivos?)**

* **Se nos dispositivos: como enfrentar o trade-off com as limitações?**

– Gateway fixo ou diferentes gateways?

– Privacidade e Segurança

* Ex: Se o gateway é um dispositivo de terceiros como manter a troca de informações de modo seguro? Se for um dispositivo de terceiros quais seriam os incentivos alguém transmita seus dados?

– IP móvel

* **Mobility Support in IPv6:** O suporte à mobilidade no IPv6 é importante pois grande parte dos dispositivos que utilizam e irão utilizar o IPv6 são móveis. Sem o suporte à mobilidade, pacotes que deveriam ser entregues a um dispositivo móvel não seriam capazes de alcançá-lo

caso ele estivesse longe de sua fonte de conexão. Graças ao IPv6 Móvel [Perkins et al. 2011], um nó móvel pode se mover de um enlace a outro sem a necessidade de trocar seu endereço e sem perder a conectividade. Um nó móvel deve ser endereçado com seu *home address*, um endereço IP dentro de seu prefixo de subrede ou enlace. Enquanto um nó móvel está em casa, os pacotes enviados ao seu endereço são encaminhados utilizando mecanismos de roteamento da Internet convencionais. Quando um nó móvel está conectado externamente, fora de casa, ele recebe um *care-of address*, um endereço IP que tem o prefixo de subrede de uma ligação externa particular. Enquanto o nó móvel permanecer neste local, os pacotes enviados a este endereço será encaminhado para o nó móvel, que pode também aceitar pacotes de vários endereços *care-of*, caso ele esteja se movendo, mas ainda acessível na ligação anterior, por exemplo.

1.4. IoT na Prática BRUNO/BRUNA (MAX 5 pg)

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 operacional³. Todos os exemplos são de código livre e hospedados no repositório oficial do Contiki. Inicialmente será exemplificado como conectar os objetos inteligentes utilizando IPv6⁴, em seguida, será pleiteado o Erbium que é uma das implementações do CoAP⁵ para redes de objetos de baixa potência Contiki.

Os experimentos serão apresentados visam atender o maior público possível. Para isto, ao longo do texto a prática é exemplificada através do uso de simulador. Entretanto, o minicurso também apresenta uma página web⁶ 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 1.3.5). 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 μ IP TCP/IP [REF].

³ <http://www.contiki-os.org/start.html>

⁴ <https://github.com/contiki-os/contiki/tree/master/examples/ipv6/rpl-border-router>

⁵ <https://github.com/contiki-os/contiki/tree/master/examples/er-rest-example>

⁶ <http://homepages.dcc.ufmg.br/~bruno.ps/iot-tp-sbrc-2016/>

Para começar, inicie o Cooja através do atalho da área de trabalho ou execute o comando abaixo e, em seguida, crie uma nova simulação:

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 ⁷ *motes* desse tipo. É recomendável que 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:**⁸ seja direcionado para a interface do Cooja.

⁷ Adicione 2 ou 5 *motes* para manter a carga de processamento e consumo de memória baixos.

⁸ 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`

```

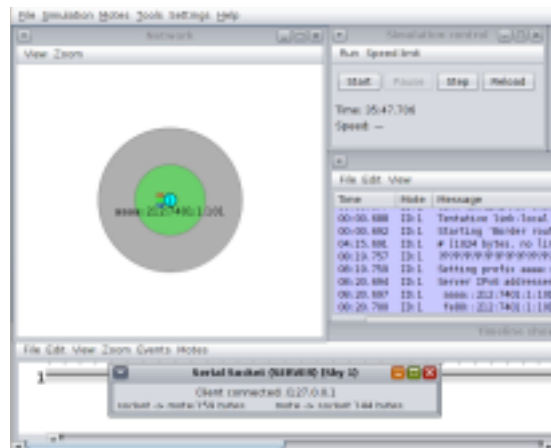
Terminal 2 – tunsliip6

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 `tunsliip6` apresentará uma saída semelhante a mostrada na caixa de título Terminal 2 e o Cooja apresentará algo como mostrado na figura ao lado⁹. Agora já é possível verificar se os *Tmotes Sky* emulados já 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* rodando *sky-websense* como mostrado abaixo:

Acesse os *Tmotes* pelo browser com:

```

http://[IPv6 do Tmote]/
http://[IPv6 do Tmote]/1
http://[IPv6 do Tmote]/t

```

Exemplo de requisição:

```

http://[aaaa::212:7403:3:303]/1

```

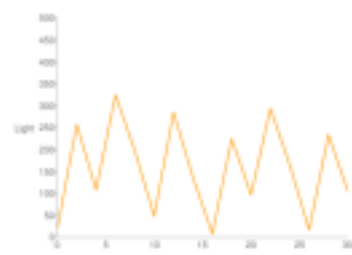
Exemplo de resposta:

```

http://[aaaa::212:7403:3:303]/1

```

Light



⁹Note 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)*¹⁰. 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 *handle function*, o servidor é capaz de tratar cada requisição e responder adequadamente com o conteúdo do recurso solicitado.

As macros¹¹ 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 os 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)
```

¹⁰CoAP RTF 7252 <http://tools.ietf.org/html/rfc7252>.

¹¹Mais 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 conteúdo (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(&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 Word” para o cliente. Além disso, se o parâmetro “len” for passado e o valor for menor que o comprimento da string “Hello Word”, 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. Exemplo 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 Erbium¹². 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)¹³ CoAP *user-agent* no Mozilla Firefox.

¹²Para mais detalhes sobre os arquivos consulte: <https://github.com/contiki-os/contiki/tree/master/examples/er-rest-example>

¹³<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 *firmware*.

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/`¹⁴

Como resultado o Mozilla Firefox deverá abrir o ambiente do Copper. A Figura 1.7 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.

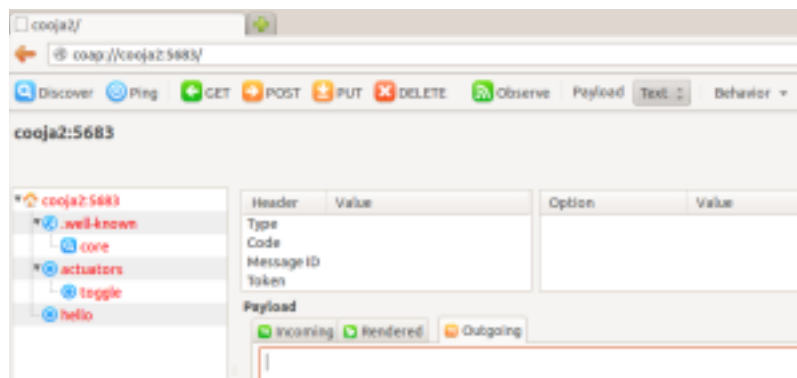


Figura 1.7. Resultado para `coap://cooja2:5683/`.

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 estar ativos ao mesmo tempo. Em caso de excesso de memória, o simulador ou compilador para o dispositivo alertará tal problema.

- Definir quais serão os experimentos.

¹⁴É importante frisar que o CoAP utiliza a porta 5683 como padrão.

- A proposta inicial seria:
 - * Instalar os códigos RPL/6LowPAN de um ou dos SOs TinyOS e Contiki, porém existe o problema de como explicar tal instalação no texto que pode ser extenso. Além da instalação, será apresentada uma demonstração consultas aos sensores dos nós através de requisições CoAP
 - * Realizar uma consulta em uma plataforma middleware tal como o João utiliza a Xyvely
 - * OBS: vale notar que as demonstrações serão realizadas ao longo da apresentação e não em um momento específico.
 - * OBS: podemos criar um vídeo para exemplificar de forma mais rápida e confiável, mas ainda assim seria interessante levar os motes reais para validar o conteúdo do vídeo.

1.5. Gerenciamento e Análise de Dados **LUCAS/JOÃO/CLAYSON (MAX 15 pg)**

1.5.1. Técnicas para abstrair a heterogeneidade dos dispositivos **LUCAS**

- CoAP, MQTT...
 - Um exemplo de abstração (RESTful)
- Ferramentas existentes (Plataformas de middleware)* **JOÃO**
 - Vai ocorrer alguma sobreposição com o minicurso passado (Plataformas para Internet das Coisas)

Esta subseção deve ser particionada.

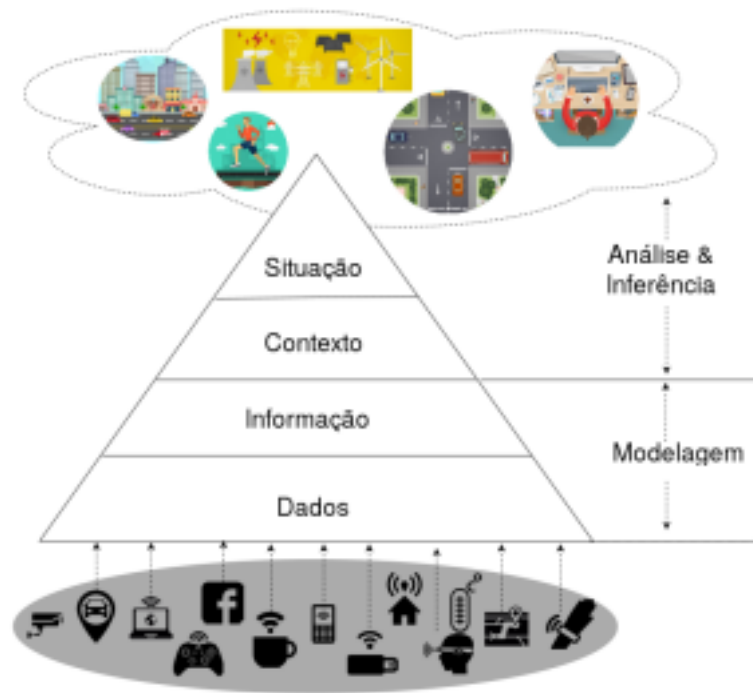
1.5.2. Técnicas de modelagem e inferência **CLAYSON**

Técnicas de modelagem e inferência são relevantes na IoT para a extração de conhecimento a partir de dados brutos. Extrair um conhecimento refere-se a modelar e analisar dados definindo uma semântica de forma a tomar decisões adequadas para prover um determinado serviço [1]. Por exemplo, em cenários de *Smart grids* [2] uma arquitetura de IoT pode auxiliar a controlar e melhorar o serviço de consumo de energia em casas e edifícios. Empregando a IoT em *Smart grids*, 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. Nesta seção, apresentamos as principais técnicas para modelagem e inferência desde a manipulação de dados brutos até a extração de conhecimento para melhoria da qualidade de serviço em cenários nos quais a IoT é empregada.

A figura **1.8** mostra uma hierarquia com os níveis de conhecimento a partir de dados brutos de sensores. Esses níveis podem ser sub-divididos em duas etapas: modelagem e análise & inferência. Na etapa de modelagem, nos deparamos 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 muitas situações, heterogêneas. Nesse sentido, algumas técnicas de fusão de dados e representação

dos dados são adotadas, tais técnicas são discutidas no restante desta seção. Além disso, visto que temos os dados armazenados em uma representação adequada, a segunda etapa consiste em interpretar esses dados visando um conhecimento de uma situação.

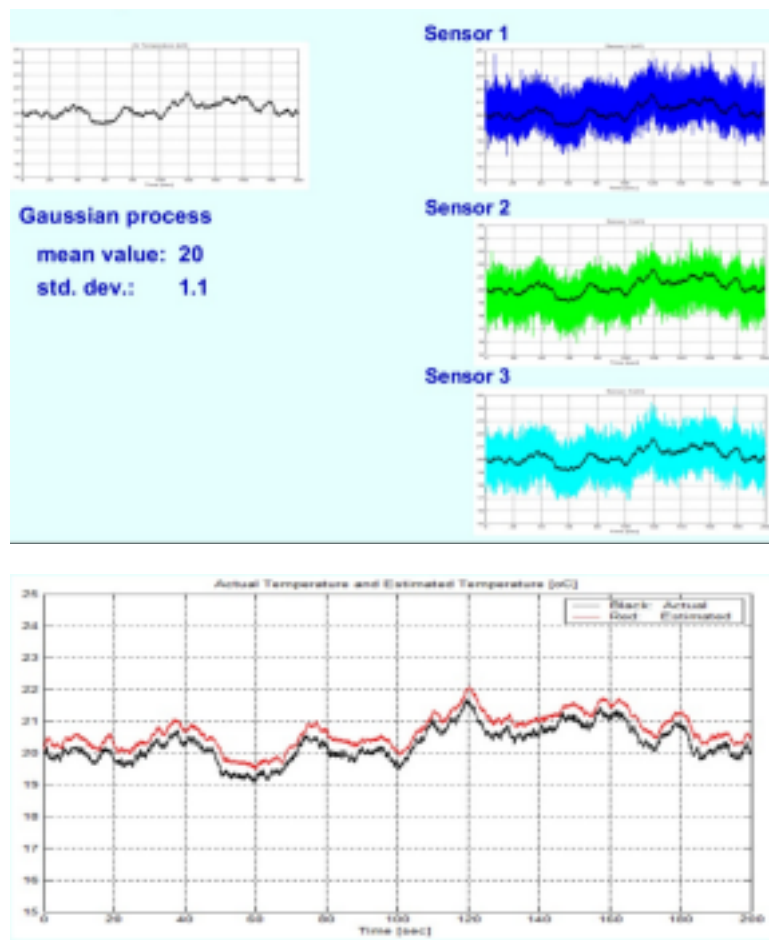
Figura 1.8. Hierarquia dos níveis de conhecimento a partir de dados brutos de sensores.



Para que os dados possam ser explorados adequadamente, eles devem possuir uma qualidade suficiente [REF]. Nesse sentido, um passo fundamental para contornar possíveis problemas existentes nos dados coletados é a de pré-processamento. O pré-processamento consiste em melhorar a qualidade dos dados contornando alguns problemas existentes devido à erros ou imprecisões de leitura, falhas de comunicação, etc. Portanto, algumas técnicas precisam ser aplicadas para lidarmos com a imperfeição dos dados, remover *outliers*, tratar conflitos, entre outros problemas que podem existir nos dados coletados. A complexidade desse processo pode complicar principalmente quando os dados são oriundos de diversas fontes de dados. Nesse sentido, a fusão de dados é aplicada de forma a termos uma combinação de informações de diversas fontes compondo uma única representação.

Klageghi et. al. definem uma taxonomia de classificação partindo de quatro problemas básicos relativos aos aspectos dos dados, são eles: imperfeição, correlação, inconsistência e discrepância. Cada um desses problemas pode ser dividido em subproblemas, mas por questões de escopo iremos focar somente nessas quatro categorias. Imperfeição refere-se aos efeitos causados por alguma imprecisão ou incerteza na medidas capturadas pelos sensores. Correlação envolve a dependência. Inconsistência surge dos problemas relacionados a disordem, conflitos e *outliers* nos dados. Discrepância

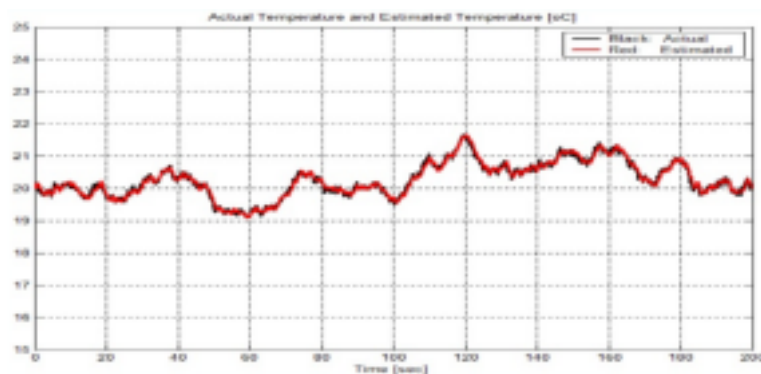
Fusão de dados é o método de combinar dados de diversos sensores para produzir uma semântica mais precisa e muitas vezes inviável de ser obtida a partir de um único



sensor [REF]. Dessa forma, são aplicados métodos automáticos ou semi-automáticos para transformar dados de diferentes 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 [REF].

A modelagem consiste em definir um conjunto de fatores tais como atributos, características e relações do conjunto de dados sensorizados. Tais fatores variam de acordo com o domínio da aplicação e consequentemente uma determinada solução de modelagem se torna mais adequada que outra. Nossa discussão quanto a modelagem consiste em técnicas conceituais de como representar a informação. As técnicas apresentadas são: *key-value*, *markup scheme*, *graphical*, *object based*, *logic based* e *ontology based modeling*. A aplicabilidade dessas técnicas pode variar de acordo com o domínio da aplicação. Portanto, cada técnica é descrita abaixo considerando suas vantagens e desvantagens em uma perspectiva geral.

- **Key-value:** Nesta técnica 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*: Um *markup scheme* utiliza *tags* para modelar os dados. Linguagens de marcação (e.g., XML¹⁵) e mecanismos de troca de dados (e.g., JSON¹⁶) podem ser utilizados para este fim e são bastante aplicados para armazenamento 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.
- *Graphical*: Esta técnica permite a construção de uma modelagem considerando o relacionamento entre os dados. UML e ORM
- *Object based*: Modelagem orientada a objetos são utilizadas para estabelecer hierarquia de classes e relacionamentos.
- *Logic based*: Neste tipo de modelagem fatos, expressões lógicas e regras são usadas para representar a informação.
- *Ontology based*:

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 determinada tipo de técnica de inferência, como discutido a seguir. A seguir, vamos discutir sobre o pre-processamento dos dados, a fusão dos dados e, por fim, sobre algumas técnicas de inferência.

Por fim, temos a geração da situação pela utilização de modelos de decisão e mineração de dados.

- *Aprendizado supervisionado*:
- *Aprendizado não supervisionado*:

¹⁵<https://www.w3.org/XML/>

¹⁶<https://tools.ietf.org/html/rfc4627>

- *Regras:*
- *Lógica fuzzy:*
- *Ontologias:*
- *Lógica probabilística:*
- Formato dos dados (JSON, XML ...)
- Aspectos dos dados
 - Espaços, Correlatos, Diferentes fontes, Imprecisos...

1.5.3. Questões de pesquisa **JOÃO**

- Qualidade dos dados (Estudo de caso)
- Fusão de dados (uma questão e 2 níveis de problemas)
 - com o artigo que o professor passou para Bruno e João, ou seja fusão (Estudo de caso).
 - in-networks
 - ITS

1.6. IoT como o meio para a Computação Ubíqua e pervasiva **LOUREIRO (MAX 2 PG)**

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

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

- [Coo 2015] (2015). An introduction to cooja. <https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja>
- [iot 2015] (2015). Iot-lab. <https://www.iot-lab.info/>
- [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.
- [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.
- [Borgia 2014] Borgia, E. (2014). The Internet of Things vision: Key features, applications and open issues. *Computer Communications*, 54:1 – 31.
- [Boulis et al. 2011] Boulis, A. et al. (2011). Castalia: A simulator for wireless sensor networks and body area networks. *NICTA: National ICT Australia*.
- [CASAGRAS 2009] CASAGRAS, P. (2009). Final Report, RFID and the Inclusive Model for the Internet of Things.
- [Chalermek et al. 2003] Chalermek, I., Govindan, R., Estrin, D., Heidemann, J., and Silva, F. (2003). Directed diffusion for wireless sensor networking. *Networking, IEEE-ACM Transactions*.
- [Chaouchi 2013] Chaouchi, H. (2013). *The internet of things: connecting objects*. John Wiley & Sons.

- [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.
- [Dawson-Haggerty et al. 2010] Dawson-Haggerty, S., Tavakoli, A., and Culler, D. (2010). Hydro: A hybrid routing protocol for low-power and lossy networks. In *Smart Grid Communications (SmartGridComm)*. IEEE.
- [de França et al. 2011] de França, T. C., Pires, P. F., Pirmez, L., Delicato, F. C., and Farias, C. (2011). Web das coisas: conectando dispositivos físicos ao mundo digital.
- [Delin 2002] Delin, K. A. (2002). The Sensor Web: A macro-instrument for coordinated sensing. *Sensors*, 2(7):270–285.
- [Developers 2011] Developers, A. (2011). What is android.
- [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.
- [Duquennoy et al. 2013] Duquennoy, S., Landsiedel, O., and Voigt, T. (2013). Let the tree bloom: Scalable opportunistic routing with orpl. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems, SenSys '13*, pages 2:1–2:14, New York, NY, USA. ACM.
- [Evans 2011] Evans, D. (2011). The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1:14.
- [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*.
- [Group 2011] Group, D. C. (2011). Sinalgo - simulator for network algorithms. <http://dcg.ethz.ch/projects/sinalgo/>

- [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*.
- [Johnson 2003] Johnson, D. B. (2003). The dynamic source routing protocol for mobile ad hoc networks. *draft-ietf-manet-dsr-09. txt*.
- [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.
- [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.
- [Levis and Lee 2010] Levis, P. and Lee, N. (2010). TOSSIM: A Simulator for TinyOS Networks.
- [Levis et al. 2003a] Levis, P., Lee, N., Welsh, M., and Culler, D. (2003a). 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. 2005a] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., and Culler, D. (2005a). *Ambient Intelligence*, chapter TinyOS: An Operating System for Sensor Networks, pages 115–148. Springer Berlin Heidelberg, Berlin, Heidelberg.
- [Levis et al. 2005b] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., et al. (2005b). TinyOS: An operating system for sensor networks. In *Ambient intelligence*. Springer.
- [Levis et al. 2003b] Levis, P. A., Patel, N., Culler, D., and Shenker, S. (2003b). Trickle: A self regulating algorithm for code propagation and maintenance in wireless sensor networks.
- [Li et al. 2014] Li, S., Xu, L. D., and Zhao, S. (2014). The internet of things: a survey. *Information Systems Frontiers*, 17(2):243–259.
- [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.
- [Microsoft 2015] Microsoft (2015). SenseWeb.

- [Nildo et al. 2014] Nildo, J., Vieira, M., Vieira, L., and Gnawali, O. (2014). CodeDrip: Data Dissemination Protocol with Network Coding for Wireless Sensor Networks. In Krishnamachari, B., Murphy, A., and Trigoni, N., editors, *Wireless Sensor Networks*, volume 8354 of *Lecture Notes in Computer Science*, pages 34–49. Springer International Publishing.
- [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.
- [Perkins et al. 2003] Perkins, C., Belding-Royer, E., and Das, S. (2003). Ad hoc on-demand distance vector (aodv) routing. Technical report.
- [Perkins et al. 2011] Perkins, C., Johnson, D., and Arkko, J. (2011). Mobility support in ipv6.
- [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.
- [Pretz 2013] Pretz, K. (2013). The next evolution of the internet. *IEEE Magazine The institute*, 50(5).
- [Reinhardt et al. 2012] Reinhardt, A., Morar, O., Santini, S., Zöller, S., and Steinmetz, R. (2012). Cbfr: Bloom filter routing with gradual forgetting for tree-structured wireless sensor networks with mobile nodes. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*, pages 1–9.
- [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.
- [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.
- [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 OM-NeT++ 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.
- [Vieira et al. 2010] Vieira, L., Loureiro, A., Fernandes, A., and Campos, M. (2010). Redes de Sensores Aquáticas. *XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, Gramado, RS, Brasil*, 24.
- [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.
- [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.
- [Woolley 2014] Woolley, M. (2014). Range limitation? what range limitation? introducing mesh networks.
- [Zachariah et al. 2015] Zachariah, T., Klugman, N., Campbell, B., Adkins, J., Jackson, N., and Dutta, P. (2015). 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.