UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

BUSTRACKER: SISTEMA DE RASTREAMENTO PARA TRANSPORTE COLETIVO

ALEXANDRE VICENZI

BLUMENAU
2015

2015/2-01

ALEXANDRE VICENZI

BUSTRACKER: SISTEMA DE RASTREAMENTO PARA TRANSPORTE COLETIVO

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Miguel Alexandre Wisintainer, Mestre - Orientador

BLUMENAU
2015

2015/2-01

BUSTRACKER: SISTEMA DE RASTREAMENTO PARA TRANSPORTE COLETIVO

Por

ALEXANDRE VICENZI

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Miguel Alexandre Wisintainer, Mestre – Orientador, FURB
Membro:	Prof. Francisco Adell Péricas, Mestre – FURB
Membro:	Prof. Matheus Carvalho Viana, Dr. – FURB

Blumenau, 7 de dezembro de 2015

Dedico este trabalho a todos que de alguma forma contribuem com o hardware livre e o software de código aberto.

AGRADECIMENTOS

Aos meus pais, Nelson e Tânia Vicenzi, pelo apoio durante a minha graduação.

Aos meus amigos, por terem estado ao meu lado sempre me motivando a continuar este trabalho e pelas ideias e informações compartilhadas.

Ao meu colega de trabalho, Germano Fronza, pelas ideias de melhoria propostas e pelas experiências compartilhadas.

Ao meu orientador, Miguel Alexandre Wisintainer, pela ideia proposta, pela confiança no desenvolvimento, pelas revisões e sugestões de como melhorar este trabalho.

Talk is cheap. Show me the code.

Linus Torvalds

RESUMO

Este trabalho apresenta o desenvolvimento de um sistema de monitoramento de transporte coletivo em tempo real com o uso da tecnologia GPS e o microcontrolador ESP8266. As informações coletadas são disponibilizadas em um sistema Web por meio de um painel com o tempo de chegada de cada ônibus até o terminal, ou por meio do mapa do Google Maps incorporado no sistema, ou ainda, consultar informações detalhadas de cada rota. O trabalho está dividido em três partes: a primeira parte é responsável por captar as informações foi desenvolvida em Lua para o microcontrolador ESP8266; a segunda parte é responsável por receber as informações, tendo sido desenvolvida em Python e faz uso do banco de dados MongoDB para armazenar as informações de localização; e a terceira parte é responsável por disponibilizar as informações ao usuário, tendo sido desenvolvida em Django e faz uso das informações disponibilizadas no MongoDB A partir dos testes realizados, comprovou-se que o sistema possui um bom grau de confiabilidade e de desempenho na atualização da informação.

Palavras-chave: Sistemas inteligentes de transporte. Internet das coisas. Transporte coletivo. Sistemas GPS.

ABSTRACT

This monograph presents the development of a public transport monitoring system in real time with the use of GPS technology and ESP8266 microcontroller. The information collected is made available on a Web system through a panel with the arrival time of each bus to the terminal, or via the embedded Google Maps, or even see detailed information for each route. The work is divided into three parts: the first part is responsible for capturing information was developed in Lua for ESP8266 microcontroller; the second part is responsible for receiving the information, having been developed in Python and uses MongoDB database for storing location information; and the third party is responsible for providing the information to the user, and was developed in Django and makes use of the information provided in MongoDB From the tests, it was found that the system has a good degree of reliability and performance update information.

Key-words: Intelligent transportation systems. Internet of things. Mass transit. GPS systems.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

2D - *2-Dimensional*

3D - *3-Dimensional*

API - *Application Programming Interface*

AP - *Access Point*

AVL - *Automatic Vehicle Location*

BCD - *Binary-coded decimal*

BHTRANS - Empresa de Transportes e Trânsito de Belo Horizonte

COMPASS - *Compass Navigation Satellite System*

CR - *Carriage return*

GGA - *Global Positioning System fix data*

GLONASS - *Globalnaya Navigazionnaya Sputnikovaya Sistema*

GNSS - *Global Navigation Satellite Systems*

GPIO - *General-purpose input/output*

GPS - *Global Positioning System*

IoT - *Internet of Things*

ITS - *Intelligent Transportation Systems*

I²C - *Inter-Integrated Circuit*

LAN - *Local Area Network*

LF - *Line feed*

MPSoC - *Multiprocessor System-on-Chip*

MQTT - *MQ Telemetry Transport*

NMEA - US National Marine Electronics Association

ORM - *Object-relational mapping*

PUC-Rio - Pontifícia Universidade Católica do Rio de Janeiro

REST - *Representational State Transfer*

RF - Requisitos Funcionais

RNF - Requisitos Não Funcionais

RTC - *Real Time Clock*

SIM - Sistema Integrado de Monitoramento

SIT - Sistemas Inteligentes de Transporte

SoC - *System on Chip*

SPI - *Serial Peripheral Interface*

SPIFFS - *SPI Flash File System*

SPTrans - São Paulo Transportes

SSE - *Server Sent Events*

UART - *Universal asynchronous receiver/transmitter*

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, o transporte urbano se tornou um tema recorrente nas cidades de médio e grande porte (MEIRELLES, 2010, p. 1), pelo fato do transporte coletivo se apresentar constantemente lotado e não atender os anseios da população (SANCHES, 2008, p. 7). O aumento acelerado da taxa de urbanização, aliado ao grande número de veículos nas vias urbanas, também contribuem para os problemas de deslocamento da população e de cargas (PILON, 2009, p. 17). Segundo Meirelles (2010, p. 2), esses problemas afetam diretamente a qualidade de vida das pessoas.

Com o grande crescimento da frota de veículos, não é possível solucionar esse problema com a simples expansão do sistema viário. Mesmo que com recursos para isto, os impactos ambientais e energéticos tornariam esse meio inviável (MEIRELLES, 2010, p 7). Para controlar tais problemas, a aplicação da tecnologia tem se mostrado viável em termos de custo-eficácia. Aliando a tecnologia ao planejamento, gestão e operação, é possível agregar sustentabilidade para o setor de transportes, trazendo redução do tempo em congestionamentos, consumo de energia e poluição ambiental (MEIRELLES, 2010, p 2).

A tecnologia que vem sendo mundialmente utiliza é denominada *Intelligent Transportation Systems* (ITS). Com ela é possível alcançar a melhoria do nível de serviços prestados, por meio da integração das tecnologias da informação e da comunicação (PILON, 2009, p. 18). Ezell (2010, p. 1, tradução nossa) propõe que "[...] a ITS traz significantes melhorias nos sistemas de transportes, incluindo a diminuição de congestionamentos, o aumento da segurança e a comodidade para quem viaja". Um dos sistemas mais populares no setor de transporte público é a bilhetagem eletrônica, que permite substituir o uso de dinheiro por um meio mais simples e seguro, reduzindo o número de fraudes e evasão de dinheiro (PEREIRA, 2007).

Em países com um alto nível de desenvolvimento, a ITS é utilizada em seu estágio mais avançado, como auxílio na navegação, aumento da segurança e gerenciamento de informações. Nos países em desenvolvimento, o foco da ITS está na fiscalização, controle de frotas e cobrança automática (PEREIRA, 2007). Apesar da ITS já estar avançada em países da Europa, Ásia e América do Norte, no Brasil esta só começou a ser abordada na última década. Porém, na maioria dos casos o seu uso ocorre de forma isolada e experimental. A disponibilização de informações estáticas já está presente na maioria das cidades brasileiras, a principal é o acesso de itinerários e rotas a partir da Internet. Porém, quando se trata de

informações em tempo real ainda existe a necessidade de implantação de um sistema para tal fim (ANTP, 2012).

Tendo em vista tal cenário, desenvolveu-se um sistema de monitoramento em tempo real que permita aos usuários de transporte urbano, bem como às empresas prestadoras do serviço, verificar qual a localização do ônibus, velocidade e tempo até a próxima parada, além de possuir um painel que permite aos usuários consultarem e serem alertados quando o seu ônibus está próximo ou chegou ao terminal.

1.1 OBJETIVOS

Este trabalho tem como objetivo o desenvolvimento de um sistema de monitoramento que alerte os usuários do transporte público quando seu ônibus está próximo ao terminal, bem como efetuar o rastreamento do ônibus e disponibilizar informações relacionadas a localização, velocidade e rotas para os usuários.

Os objetivos específicos do trabalho são:

a) desenvolver um aplicativo a ser utilizado no ônibus para o envio de suas paradas nos terminais;
b) desenvolver um web service que será responsável por receber e armazenar as informações enviadas pelos ônibus, bem como disponibilizar estas informações para uma aplicação web e uma *Application Programming Interface* (API) para sistemas de terceiros;
c) desenvolver uma aplicação Web que permitirá aos usuários a consulta da localização dos ônibus em tempo real.

1.2 ESTRUTURA

Este trabalho está dividido em quatro capítulos. O primeiro capítulo apresenta a introdução do trabalho e os objetivos. O segundo capítulo apresenta a fundamentação teórica sobre sistemas inteligentes de transporte, sistema de navegação por satélite, redes, a linguagem de programação Lua e o microcontrolador ESP8266. No terceiro capítulo é demonstrado o desenvolvimento do trabalho com requisitos, especificação, implementação, resultados e operacionalidade da aplicação. Por fim, no quarto capítulo são relatadas as conclusões e também as possíveis extensões.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está organizado em seis seções. A Seção 2.1 aborda os sistemas inteligentes de transporte. Na Seção 2.2 são apresentados os sistemas de navegação global por satélite. Na Seção 2.3 é abordado redes de comunicação. Na Seção 2.4 é apresentada a linguagem de programação Lua. Na Seção 2.5 é apresentado o dispositivo ESP8266. Finalmente, na Seção 2.6 são apresentados os trabalhos correlatos.

2.1 SISTEMAS INTELIGENTES DE TRANSPORTE (SIT)

Os SIT são uma combinação de tecnologia de ponta, sistemas de informação, comunicação, sensores e matemática avançada para aprimorar a atual infraestrutura de transporte (SUSSMAN, 2008, p. 3, tradução nossa). Segundo Williams (2008, p. 19, tradução nossa), "um sistema SIT não é necessário se não traz consigo algum benefício". Entretanto a variedade de benefícios com o uso dessas informações pode ser extensa.

O leque de aplicações voltadas para os transportes inteligentes é abrangente, e apesar de seu custo, esse tipo de tecnologia tem se mostrado uma alternativa viável em termos de custo e eficácia para a melhoria do sistema de transporte (MEIRELLES, 2010). Dentre os principais usos para os SIT, estão a diminuição dos congestionamentos, a redução da poluição, o gerenciamento e controle de frotas e a melhoria na qualidade do transporte para o usuário (SILVA, 2000, p. 6).

Nos países desenvolvidos este tipo de tecnologia vem sendo usada em grande escala. Por outro lado, no Brasil, o uso de SIT ainda está em fase inicial (MEIRELLES, 2010). Existem iniciativas de uso, na sua maioria pelas operadoras de transportes públicos e de cargas, nas quais é possível observar um investimento alto em sistemas de navegação, sensoreamento, localização automática e roterização para melhorar as demandas deste setor (SILVA, 2000, p. 10).

2.2 SISTEMAS DE NAVEGAÇÃO GLOBAL POR SATÉLITE

Segundo Rao (2010, p. 1, tradução nossa), "*Global Navigation Satellite Systems* (GNSS) é o termo genérico mais comum para sistemas de navegação por satélite que fornece cobertura global de posicionamento em espaço *3-Dimensional* (3D)". Os GNSS são capazes de calcular o posicionamento de objetos em termos de latitude, longitude, altitude, velocidade, direção e tempo usando um processo matemático denominado trilateração (RAO, 2010).

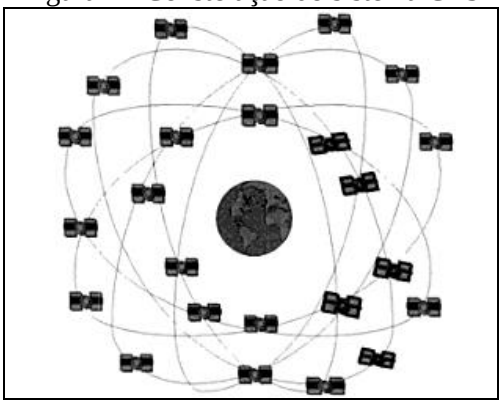
Atualmente existem dois sistemas GNSS em operação: o *Global Positioning System* (GPS) e o *Globalnaya Navigazionnaya Sputnikovaya Sistema* (GLONASS). Além desses,

mais dois sistemas estão em desenvolvimento: o *Galileo* e o *Compass Navigation Satellite System* (COMPASS) (BHATTA, 2011, p. 2).

O GPS é um sistema de navegação global desenvolvido pelo Departamento de Defesa dos Estados Unidos da América. O seu intuito inicial era ser o principal sistema de navegação das forças armadas americanas (MONICO, 2001). Como esse sistema proporciona alta precisão e um alto grau de desenvolvimento da tecnologia envolvida nos receptores GPS, uma grande comunidade usuária emergiu nas mais variadas aplicações civis (navegação, posicionamento geodésico, topográfico, etc.) (CEUB, 2006). Segundo Monico (2001, p. 21), "A concepção do sistema GPS permite que um usuário, em qualquer local da superfície terrestre, ou próxima a ela, tenha à sua disposição, no mínimo, quatro satélites para serem rastreados".

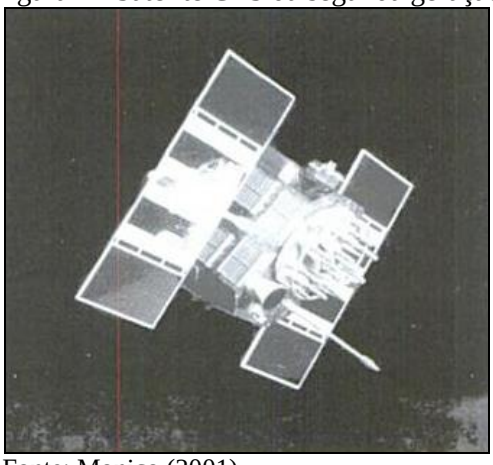
Os primeiros satélites do sistema GPS foram lançados em 1978 e a sua segunda geração de satélites iniciou-se em 1989. Em 1995, o sistema foi declarado como completamente operacional, após o lançamento dos 24 satélites estarem completas (HINCH, 2010). Na Figura 1 é possível observar a constelação do sistema GPS e na Figura 2 é possível observar um satélite GPS do Bloco II (segunda geração).

Figura 1 - Constelação do sistema GPS



Fonte: Monico (2001).

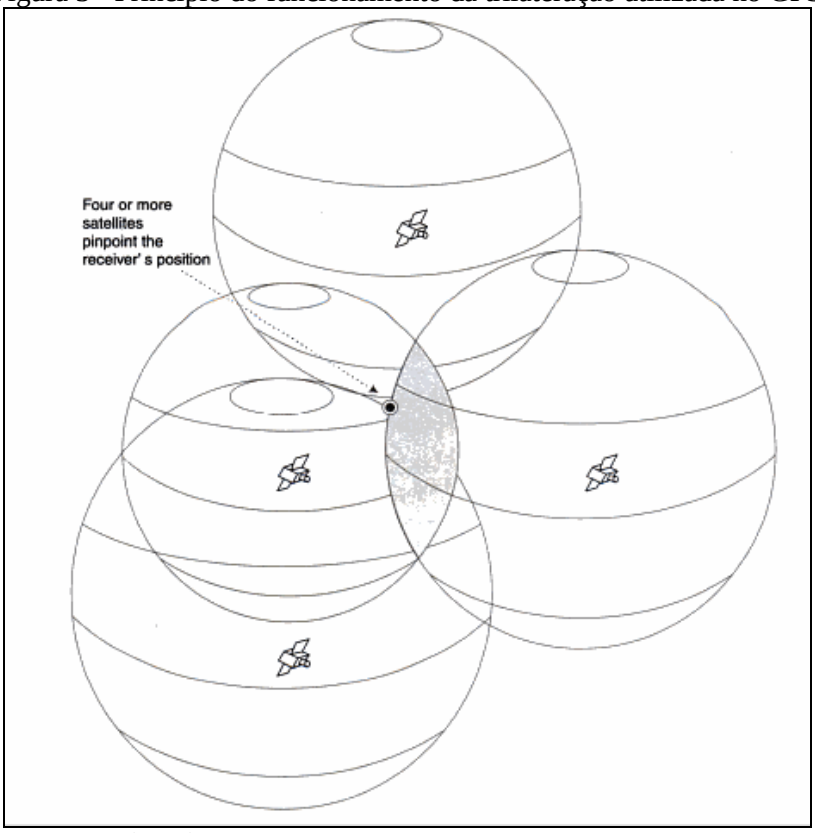
Figura 2 - Satélite GPS da segunda geração



Fonte: Monico (2001).

O GPS funciona utilizando o processo de trilateração, comumente chamado de triangulação. O processo de trilateração determina a posição de um objeto medindo a distância entre outros objetos que já se sabe a posição, no caso, pontos de referência. Este processo permite calcular a posição dos objetos em espaço *2-Dimensional* (2D) e 3D (RAO, 2010). O princípio básico deste cálculo consiste em medir a distância entre o objeto e quatro satélites. Como se conhece a coordenada dos satélites é possível determinar a posição do objeto. Do ponto de vista geométrico, apenas três satélites são necessários, porém o quarto é utilizado em razão do não-sincronismo dos relógios entre os satélites e o usuário (MONICO, 2001). Na Figura 3 é possível observar a determinação da posição no plano 3D utilizando quatro satélites.

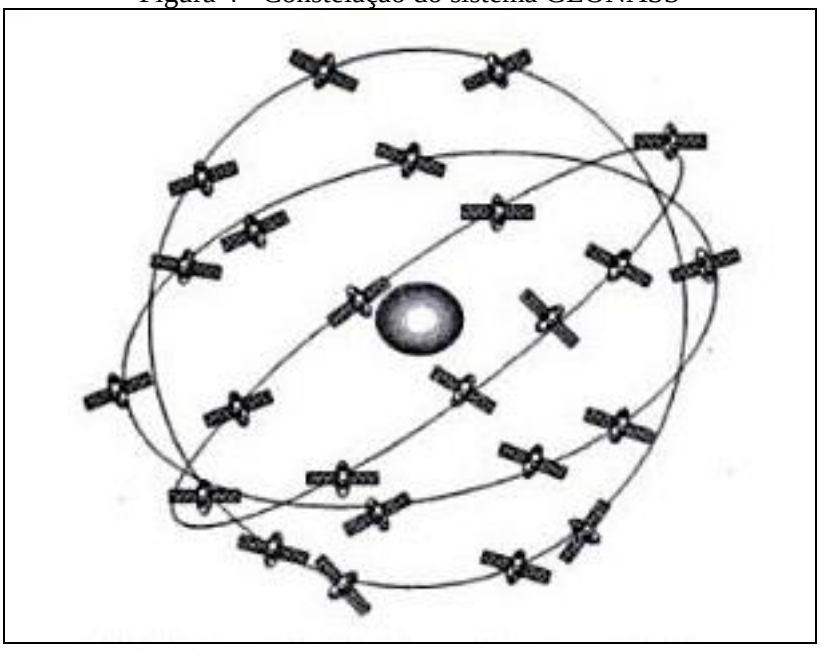
Figura 3 - Princípio do funcionamento da trilateração utilizada no GPS



Fonte: Rao (2010).

O GLONASS é operado pelo Departamento de Defesa Russo e teve seu início na União Soviética. Os primeiros três satélites foram lançados em órbita no ano de 1982. Para funcionamento completo são necessários 24 satélites, porém, devido a instabilidades políticas esse número nunca foi alcançado. Em 2009, o GLONASS, possuía 22 satélites com apenas 16 em operação (RAO, 2010). Na Figura 4 é possível observar a constelação do sistema e na Figura 5 um satélite GLONASS.

Figura 4 - Constelação do sistema GLONASS



Fonte: Rao (2010).

Figura 5 - Satélite GLONASS

Fonte: Rao (2010).

2.2.1 Formato NMEA

A US National Marine Electronics Association (NMEA) propôs, em 1983, o formato NMEA 0183, originalmente definido como uma interface para os dispositivos eletrônicos marítimos, mas que acabou se tornando o padrão industrial dos receptores GNSS. O formato especifica um padrão para posicionamento, contendo indicadores de qualidade, curso e velocidade (HOFMANN-WELLENHOF; LICHTENEGGER; WASLE, 2007).

Uma sentença NMEA consiste em uma cadeia de caracteres no formato ASCII 8-bit e contém no máximo 82 caracteres. Toda sentença inicia com `$`. Os próximos dois caracteres indicam o *talker* e os próximos três indicam o tipo da sentença. Os *talkers* mais comuns são `GP`, indicando GPS, e `GL`, indicando GLONASS. Os demais campos são separados por vírgula, seguidos de um asterisco e um *checksum* opcional na maioria das mensagens. Por fim a

mensagem termina com *carriage return* (CR) e *line feed* (LF) (EL-RABBANY, 2002). No Quadro 1 é possível observar um exemplo da sentença *Global Positioning System fix data* (GGA), responsável por prover informação de localização e correção.

Quadro 1 - Formato da sentença GGA

$GPGGA,hhmmss.ss,llll.ll,a,yyyyy.yy,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx*hh<CR><LF>
$GPGGA,115417.00,4338.123456,N,07938.123456,W,1,10,01.1,095.095,M,,M,999,0000

Fonte: El-Rabbany (2002).

2.3 REDES DE COMUNICAÇÃO

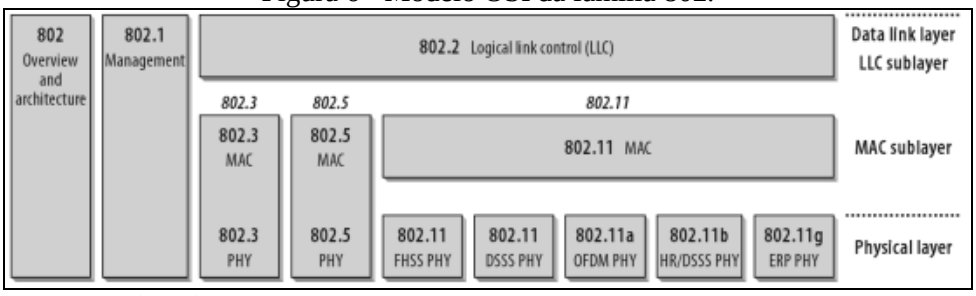
Nos últimos anos, houve um aumento na mobilidade. Devido a esse novo estilo de vida, os meios tradicionais de acesso à Internet se tornaram inadequados. Conexões sem fio surgiram para remover as restrições impostas pelas conexões cabeadas (GAST, 2005).

O termo Wi-Fi significa *wireless fidelity* e é o nome da aliança reguladora dos padrões e protocolos usados na comunicação sem fio, bem como o nome deste meio de comunicação. A Wi-Fi Alliance é uma organização sem fins lucrativos responsável por certificar a interoperabilidade dos dispositivos baseado no padrão 802.11 e os membros desta aliança são as empresas responsáveis por produzir os dispositivos 802.11 (DAVIS, 2004).

2.3.1 Redes IEEE 802.11

O padrão IEEE 802.11 pertence à família da IEEE 802, que é uma série de especificações para a *Local Area Network* (LAN). A IEEE 802 é focada nos dois níveis mais baixos do modelo OSI, pois eles são responsáveis por incorporar componentes físicos e *link* de dados (GAST, 2005). Na Figura 6 é possível observar a família 802 no modelo OSI.

Figura 6 - Modelo OSI da família 802.

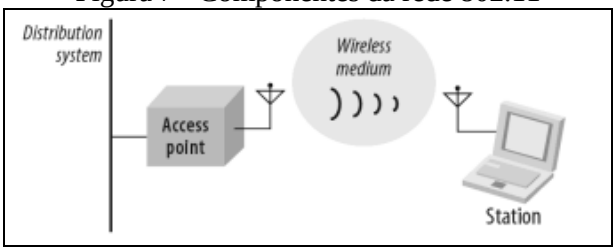


Fonte: Gast (2005).

Uma rede IEEE 802.11 possui quatro componentes básicos: um sistema de distribuição, *Access Point* (AP), um meio sem fio e estações móveis. O sistema de distribuição é responsável por enviar os pacotes para o destino quando um AP está conectado a uma rede. Os APs são os dispositivos que criam a ponte sem fio. Por fim, as estações

móveis são os usuários que utilizam o meio sem fio para transmitir dados (BURBANK et al., 2013). Na Figura 7 é possível observar os componentes que compõem a 802.11.

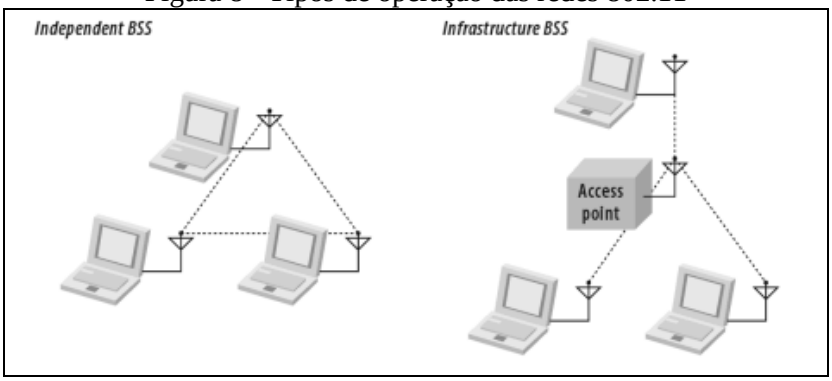
Figura 7 - Componentes da rede 802.11



Fonte: Gast (2005).

As redes 802.11 possuem dois tipos de operação: independente e infraestrutura. Nas redes independentes, comumente chamadas de *adhoc*, as estações comunicam-se diretamente. A menor rede possível no modo *adhoc* é entre duas estações. As redes de infraestrutura se distinguem das *adhoc* por dependerem de um AP, que é responsável por fazer a comunicação entre as demais estações da rede (GAST, 2005). Na Figura 8 está exemplificado os tipos de rede independente e infraestrutura.

Figura 8 - Tipos de operação das redes 802.11



Fonte: Gast (2005).

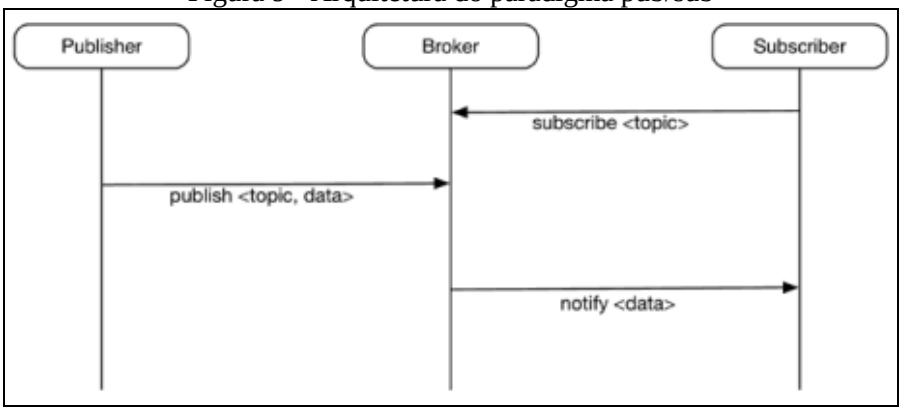
2.3.2 Protocolo MQ Telemetry Transport (MQTT)

O protocolo de mensagens MQTT é projetado para um baixo consumo de banda de rede e requisitos de hardware sendo extremamente simples e leve. O MQTT foi desenvolvido pela IBM e Eurotech e é projetado para enviar dados através de redes intermitentes ou com baixa banda de dados, para isto o protocolo é desenvolvido em cima de vários conceitos que garantem uma alta taxa de entrega das mensagens (CHEN et al., 2014).

O MQTT utiliza o paradigma *publish/subscribe* (*pub/sub*) para a troca de mensagens. O paradigma *pub/sub* implementa um *middleware* chamado de *broker*. O *broker* é responsável por receber, enfileirar e disparar as mensagens recebidas dos *publishers* para os *subscribers* (DESAI, 2015). O *publisher* é responsável por se conectar ao *broker* e publicar mensagens. Já o *subscriber* é responsável por se conectar ao *broker* e receber as mensagens

que ele tiver interesse (WAHER, 2015). Na Figura 9 é possível observar a arquitetura do paradigma *pub/sub*.

Figura 9 - Arquitetura do paradigma pub/sub

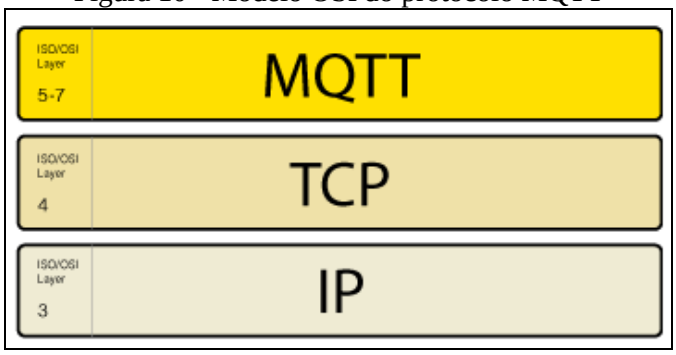


Fonte: Desai (2015).

O paradigma *pub/sub* utiliza o conceito de tópicos para processar as mensagens, em que cada mensagem é enviada para um determinado tópico. Diferente de outros protocolos de mensagem, o *publisher* não envia a mensagem diretamente ao *subscriber*, mas sim ao *broker*. O *publisher* envia a mensagem para o *broker* em um determinado tópico. O *broker* é responsável por receber a mensagem do *publisher* e fazer uma pré-filtragem das mensagens e enviá-las para os *subscribers* que estivem registrados em um determinado tópico (DESAI, 2015).

O protocolo MQTT é baseado no TCP/IP e ambos, cliente e *broker*, necessitam da pilha TCP/IP para o seu funcionamento (CHEN et al., 2014). Na Figura 10 é exibido o modelo do protocolo no padrão OSI.

Figura 10 - Modelo OSI do protocolo MQTT



Fonte: HiveMQ (2015).

2.4 LUA

Lua nasceu em 1993 na Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) no Brasil e é uma linguagem de código aberto escrita em C ANSI. Lua passou a ser usada em

diferentes setores, como robótica, sistemas distribuídos, processamento de imagens e web (IERUSALIMSCHY; FIGUEIREDO; CELES, 2007?).

A linguagem de programação Lua é projetada para dar suporte ao paradigma procedimental, porém também oferece suporte aos paradigmas de programação orientado a objetos, programação funcional e programação orientada a dados. Lua é uma linguagem planejada para ser utilizada em aplicações que necessitem de extensão, ou seja, ela necessita de um programa hospedeiro responsável por sua execução. A distribuição oficial de Lua inclui um programa hospedeiro, chamado `lua`, que funciona como um interpretador de linha de comando (IERUSALIMSCHY; FIGUEIREDO; CELES, 2008). No Quadro 2 é possível observar a implementação do cálculo de fatorial em Lua.

Quadro 2 - Função `fatorial` em Lua

```
function factorial(n)
  if n == 0 then
    return 1
  else
    return n * factorial(n - 1)
  end
end
```

2.5 PYTHON

Python é uma linguagem de programação de propósito geral, dinâmica, orientada a objetos e multiplataforma desenvolvida por Guido van Rossum no início de 1990. Python pode ser comparado com Ruby e Perl em nível de linguagem. Apesar de ser orientada a objetos, é possível utilizar os paradigmas de progração procedural e funcional (MARTELLI, 2006). Segundo Sweigart (2015), "o nome Python é proveniente do grupo surreal de comédia britânico Monty Python, e não do nome da cobra".

O código Python é projetado para ser legível, reutilizável e passível de manutenção. A uniformidade da linguagem permite que seja de fácil entendimento, mesmo para os que não possuem conhecimento na linguagem. Em muitos casos, o código escrito em Python é um terço ou um quinto do equivalente ao em Java ou C++ (LUTZ, 2013).

2.6 ESP8266

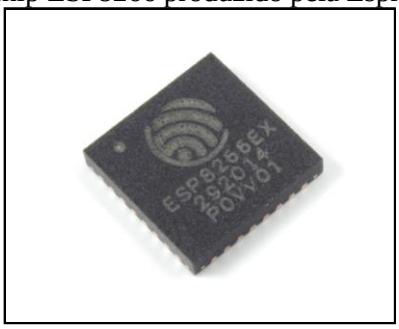
Microcontrolador é um circuito integrado capaz de executar programas (KOLBAN, 2015). Um dos mais conhecidos é o Arduino, por conta de sua arquitetura aberta que o permite ser utilizado em inúmeros tipos de projetos. Muitos desses dispositivos contam com capacidades de manipulação de *General-purpose input/output* (GPIO), *Inter-Integrated Circuit* (I^2C), *Serial Peripheral Interface* (SPI), *Universal asynchronous receiver/transmitter*

(UART) entre outros. Porém, assim como o Arduino, muitos microcontroladores não possuem acesso à internet sem a necessidade de um hardware complementar (KOLBAN, 2015).

É neste momento que entra o ESP8266, um micro controlador de baixo custo (KOLBAN, 2015). O ESP8266 é um *System on Chip* (SoC) de baixo consumo de energia e alto desempenho *wireless* que está de acordo com o padrão IEEE802.11bgn (ESPRESSIF SYSTEM, 2013, p. 4). Segundo Flynn e Luk (2011, p. 1, tradução nossa) "a arquitetura SoC é um conjunto de processadores, memórias e interconexões sob medida para um domínio de aplicação". Dependendo do domínio de aplicação, SoC pode ser classificado em duas categorias: (1) multiprocessador SoC de propósito geral (*Multiprocessor System-on-Chip* – MPSoC) e (2) SoC de aplicação específica (KUNDU; CHATTOPADHYAY, 2014, p. 1).

Destinado a plataformas móveis, o ESP8266 provê a capacidade de incorporar funções Wi-Fi em outros sistemas a um custo baixo e foi projetado para projetos de mobilidade, eletrônicos vestíveis e a Internet das Coisas (*Internet of Things* – IoT) (KOLBAN, 2015). Na Figura 11 é possível observar o chip produzido pela Espressif System e no Quadro 3 estão listadas as principais características do ESP8266.

Figura 11 - Chip ESP8266 produzido pela Espressif System.



Fonte: In-Circuit (2015).

Quadro 3 - Principais características do ESP8266

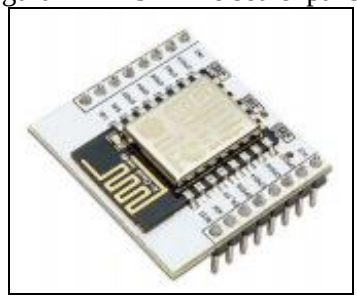
Voltage	3.3V
Current consumption	10uA – 170mA
Flash memory attachable	16MB max (512K normal)
Processor	Tensilica L106 32 bit
Processor speed	80-160MHz
RAM	32K + 80K
GPIOs	17 (multiplexed with other functions)
Analog to Digital	1 input with 1024 step resolution
802.11 support	b/g/n/d/e/i/k/r
Maximum concurrent TCP connections	5

Fonte: Kolban (2015).

2.6.1 ESP-12

O ESP-12 é uma das versões mais populares e flexíveis do ESP8266. Sua popularidade é devido ao grande número de GPIOs disponíveis, o que torna ele um grande candidato para projetos que necessitem de acesso a mais de um módulo externo. Para utilizar o ESP-12, é necessário soldá-lo a um expansor (KOLBAN, 2015). Na Figura 12 é possível observar o ESP-12 soldado no expansor.

Figura 12 - ESP-12 e seu expansor



Fonte: Kolban (2015).

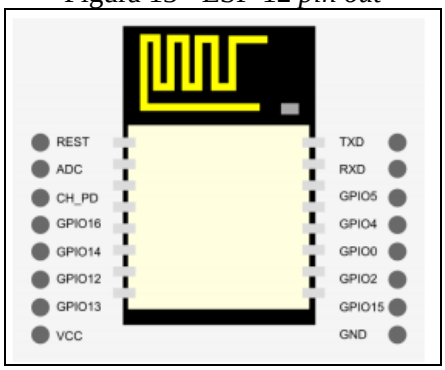
A Figura 13 exibe o *pin out* do ESP-12 após soldado no expansor. Cada pino possui um uso específico, como descrito no Quadro 4.

Quadro 4 - Descrição dos pinos do ESP-12

Name	Description
VCC	3.3V.
GPIO 13	Also used for SPI MOSI.
GPIO 12	Also used for SPI MISO.
GPIO 14	Also used for SPI Clock.
GPIO 16	
CH_PD	Chip enable. Should be high for normal operation. • 0 – Disabled • 1 – Enabled
ADC	Analog to digital input
REST	External reset. • 0 – Reset • 1 – Normal
TXD	UART 0 transmit.
RXD	UART 0 Receive.
GPIO 4	Regular GPIO.
GPIO 5	Regular GPIO.
GPIO 0	Should be **high** on boot, low for flash update.
GPIO 2	Should be **high** on boot.
GPIO 15	Should be **low** on boot and flash.
GND	Ground.

Fonte: Kolban (2015).

Figura 13 - ESP-12 *pin out*



Fonte: Kolban (2015).

2.6.2 NodeMCU

O NodeMCU é um *firmware* e kit de desenvolvimento que permite a programação de protótipos IoT. O *firmware* utiliza o paradigma *event-driven* para facilitar o desenvolvimento de aplicações que necessitem acesso à internet. Além disso, o *firmware* integra módulos de `GPIO`, `PWM`, `1-Wire`, `I2C`, `ADC`, entre outros, para facilitar o manuseio de módulos baseado no chip ESP8266 (NODEMCU TEAM, 2014).

O paradigma *event-driven* (ou programação assíncrona) consiste em um estilo de programação em que o fluxo de execução é determinado por eventos disparados por *callbacks*. Uma *callback* é uma função que é invocada quando algo acontece. Por exemplo, ao pressionar um botão um evento é disparado e a *callback* efetuará uma ação para esse evento (TEIXEIRA, 2013). No Quadro 5 é possível observar um exemplo de código de um servidor HTTP simples para o NodeMCU no estilo *event-driven*.

Quadro 5 - Código de um servidor HTTP simples em NodeMCU

```
srv = net.createServer(net.TCP)
srv:listen(80, function(conn)
    conn:on("receive", function(conn, payload)
    print(payload)
    conn:send("<h1>Hello, NodeMCU.</h1>")
    end)
end)
```

Fonte: NodeMCU Team (2014).

O NodeMCU é baseado no eLua, uma versão do Lua 5.1 para sistemas embarcados e seu sistema de arquivos é o *SPI Flash File System* (SPIFFS). O firmware substituiu algumas bibliotecas padrões do Lua por versões específicas para o ESP8266. Além dessas mudanças, algumas bibliotecas, como `math` e `debug`, foram completamente removidas para diminuir o consumo de memória do dispositivo (ELLISON, 2015).

2.7 TRABALHOS CORRELATOS

A seguir são apresentados três trabalhos com características semelhantes ao objetivo desse trabalho. O primeiro é o SIU (BHTRANS, 2008?), sistema que faz parte do SITBus, implantando em Belo Horizonte. O segundo é o Olho Vivo (SPTRANS, 2009), sistema que faz parte do SPTrans e está implantado em São Paulo. O terceiro é o Moovit (MOOVIT, 2014), aplicativo de navegação inteligente para *Smartphones*. Por fim, é feita uma breve análise sobre os trabalhos correlatos apresentados.

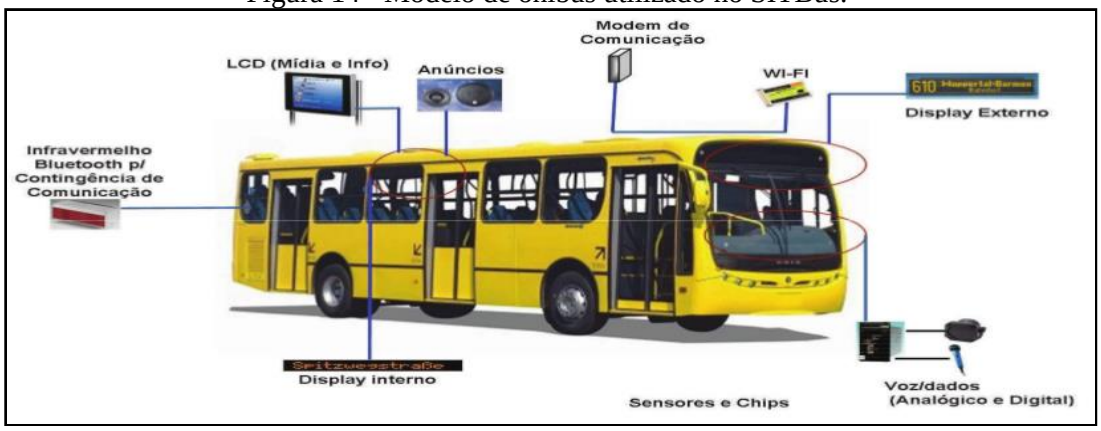
2.7.1 SIU – Sistema Informação ao Usuário

O SIU é um sistema que faz parte do SITBus e permite o envio, recebimento e prestação de informações aos usuários sobre os serviços de transporte público no município de Belo Horizonte. O SITBus é uma iniciativa da Prefeitura Municipal, em conjunto com a Empresa de Transportes e Trânsito de Belo Horizonte (BHTRANS), para a melhoria do transporte urbano, visando o bem-estar e maior mobilidade do usuário, bem como facilitar a gestão dos serviços de transporte coletivo (BHTRANS, 2008?, p. 4). De acordo com a BHTRANS (2008?, p. 60):

> Os sistemas de informação ao usuário, constituídos por equipamentos, sistemas, plataformas e serviços visam promover de forma extensiva, rápida, atualizada, objetiva e eficaz a disponibilização de informações visando ao aumento da conveniência, usabilidade e conforto do USUÁRIO na utilização dos SERVIÇOS, através da divulgação de horários, ITINERÁRIOS, tarifas e informações pertinentes ao sistema, em tempo real ou não.

Para fazer esse controle e distribuição das informações, o SITBus conta com dispositivos embarcados nos ônibus, dispositivos em terminais, pontos de embarque e desembarque, e mídias complementares. Contudo, apesar de o sistema estar no ar desde 2008, ele atende poucas regiões. A BHTrans prometeu, para 2011, instalar 900 painéis em pontos de ônibus da capital com informações em tempo real sobre os horários de chegada dos ônibus. No ano passado, com a implantação do Move, sistema de transporte rápido por ônibus, outros 600 equipamentos entraram na lista. O problema é que anos após a implantação, apenas 650 estão em funcionamento – 382 nos pontos convencionais e 268 nas estações do Move. Eles representam 7% do total de pontos de ônibus da capital (SUAREZ, 2015). Na Figura 14 demonstra-se um modelo de ônibus utilizado no SITBus, com seus dispositivos de controle e distribuição de informações.

Figura 14 - Modelo de ônibus utilizado no SITBus.

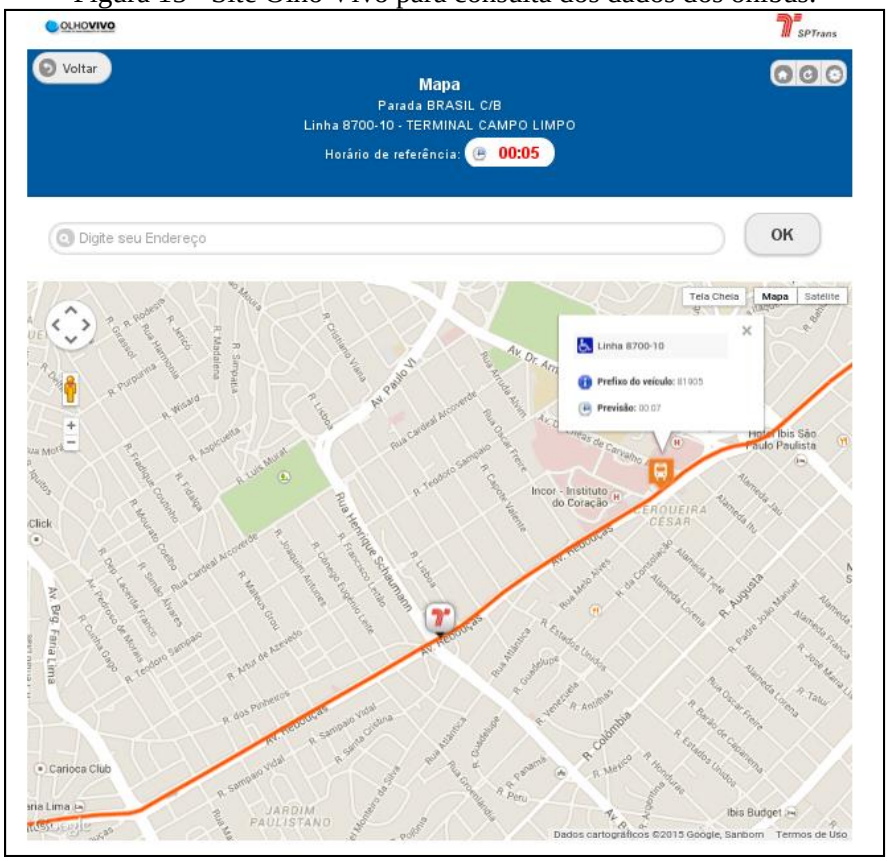


Fonte: BHTRANS (2008).

2.7.2 Olho Vivo

O Olho Vivo é um sistema que foi implantando em 2008 e faz parte da São Paulo Transportes (SPTrans). A partit de dados obtidos dos ônibus, é possível exibir informações referentes a tempo de viagem e velocidade dos veículos para os usuários, como mostra a Figura 15, bem como, melhorar a fluidez do transporte urbano. Essas informações são obtidas por meio do Sistema Integrado de Monitoramento (SIM), que, por sua vez, possibilita monitorar, controlar e fiscalizar as informações sobre os veículos, passageiros e o sistema viário de forma segura e contínua (SPTRANS, 2009).

Figura 15 - Site Olho Vivo para consulta dos dados dos ônibus.



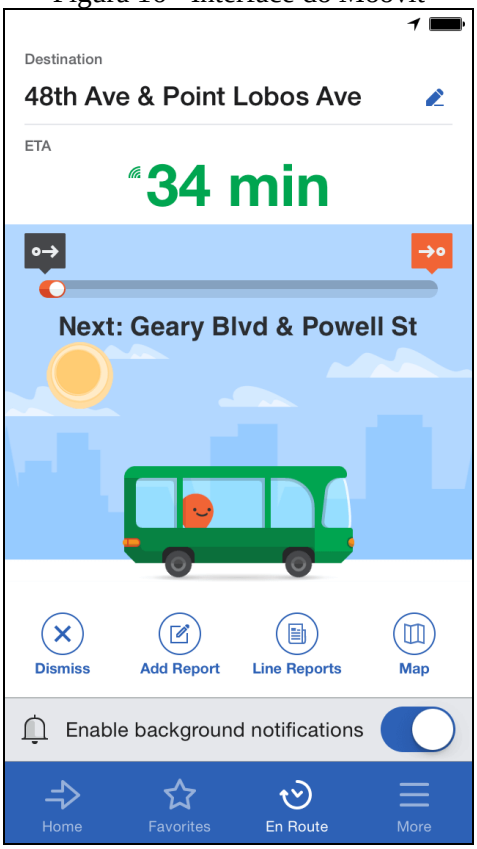
O equipamento utilizado para o monitoramento dos ônibus é chamado de *Automatic Vehicle Location* (AVL). Ele é composto por um GPS, um microprocessador e memória, sendo capaz de controlar e comunicar com veículos e máquinas através de uma conexão de rede (SPTRANS, 2009, p. 9).

É possível consultar informações pela Web ou por aplicativo para *Smartphones* com os sistemas operacionais Android, iOS e Windows Phone através do aplicativo Cadê o Ônibus. Além disso, existe uma API para a consulta dos dados da SPTrans, o que permite a integração do Olho Vivo com outros sistemas.

2.7.3 Moovit

O Moovit é um aplicativo para uso em viagens ou transporte público. Com ele é possível realizar escolhas de rotas e verificar qual o melhor caminho e tempo estimado de chegada. O Moovit oferece, em tempo real, qual a rota melhor e mais rápida até o destino. O seu funcionamento consiste em combinar informações do transporte público com informações de tráfico em tempo real com base na comunidade de usuários do aplicativo (MOOVIT, 2015a). Na Figura 16 é possível observar a interface do aplicativo.

Figura 16 - Interface do Moovit



Fonte: Moovit (2015b).

O Moovit já está disponível em mais de 600 cidades pelo mundo, inclusive algumas cidades brasileiras como São Paulo e Rio de Janeiro. O aplicativo está disponível para iOs, Android e Windows Phone. A empresa foi fundada em 2012 em Israel e sua base de dados já conta com mais de 20 milhões de usuários (MOOVIT, 2014).

2.7.4 Comparativo entre os trabalhos correlatos

O SITBus é um sistema de monitoramento com foco no usuário do transporte público, pois ele é projetado para disponibilizar informações em tempo real nos terminais e em pontos de ônibus. Suas principais desvantagens são a falta de integração com uma aplicação Web ou com Smartphone e o fato de atender a poucas regiões.

O Olho Vivo é focado no rastreamento dos ônibus, para identificar onde o veículo se localiza, qual sua velocidade média de deslocamento e o tempo de chegada em determinado ponto. É possível afirmar que o Olho Vivo é uma solução com um nível de maturidade superior ao SITBus, e diferente desse último, o Olho Vivo atende as linhas da cidade de São Paulo e conta com aplicações tanto Web quanto para Smartphones.

O Moovit é uma aplicação semelhante ao Olho Vivo, porém a grande diferença é que as suas informações são obtidas a partir dos usuários. Já no Olho Vivo as informações são coletadas a partir de um hardware instalado no ônibus, o que garante maior precisão nas informações obtidas. Apesar disto, o Moovit se mostra uma aplicação mais intuitiva e amigável para os usuários finais.

3 DESENVOLVIMENTO

As seções a seguir descrevem os requisitos, a especificação, a implementação e a operacionalidade do trabalho, abordando sucintamente as ferramentas e etapas utilizadas no desenvolvimento do projeto. Ao fim, são mostrados os resultados obtidos com este trabalho.

3.1 REQUISITOS

A seguir estão listados os Requisitos Funcionais (RF) e os Requisitos Não Funcionais (RNF):

a) o sistema deve permitir o cadastro da frota de ônibus (Requisito Funcional – RF);
b) o sistema deve permitir o cadastro dos terminais de ônibus (RF);
c) o sistema deve alertar os usuários do terminal quando um ônibus chegar (RF);
d) o sistema deve possuir controle de acesso para alterações das informações cadastrais (RF);
e) o sistema deve utilizar a linguagem Lua para programação do ESP8266 (Requisito Não Funcional – RNF);
f) o sistema deve utilizar HTML5 e JavaScript para a parte Web (RNF);
g) o sistema deve ser acessível por navegadores mais recentes, como por exemplo, o Chrome e o Firefox (RNF);
h) o sistema deve utilizar a linguagem Python para o Web Service (RNF);
i) o sistema deve utilizar o dispositivo ESP8266 (RNF);
j) o sistema deve utilizar as tecnologias GPS e Wi-Fi (RNF);
k) o sistema deve utilizar o banco de dados MongoDB (RNF).

Para atender estes requisitos, o trabalho está dividido em quatro partes, sendo elas:

a) um dispositivo físico, que é responsável por enviar as informações de posicionamento do ônibus utilizando a tecnologia de navegação GPS;
b) um sistema de mensagens responsável por receber, processar e armazenar as informações recebidas do dispositivo físico;
c) uma aplicação Web, responsável pelo gerenciamento da frota e das rotas, além de permitir aos usuários do sistema de transporte público efetuar consultas relacionadas a horários e localização do ônibus;
d) um serviço de Web Service, responsável pela integração de outros sistemas e aplicativos ao trabalho proposto.

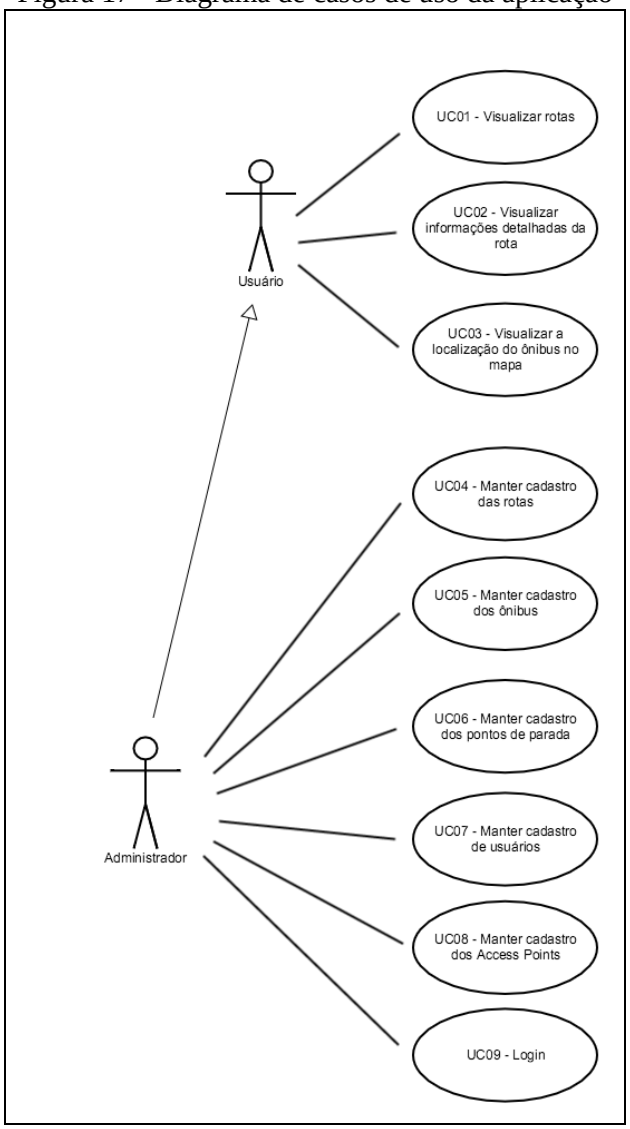
3.2 ESPECIFICAÇÃO

A especificação do trabalho foi criada utilizando as ferramentas Gliffy para gerar os casos de uso e o diagrama de topologia e o Fritzing para gerar o diagrama esquemático. Neste trabalho foram elaborados os diagramas de caso de uso, esquemático e de topologia.

3.2.1 Diagrama de casos de uso

A Figura 17 exibe o diagrama de casos de uso da aplicação Web para o ator `Usuário` e o ator `Administrador`.

Figura 17 - Diagrama de casos de uso da aplicação

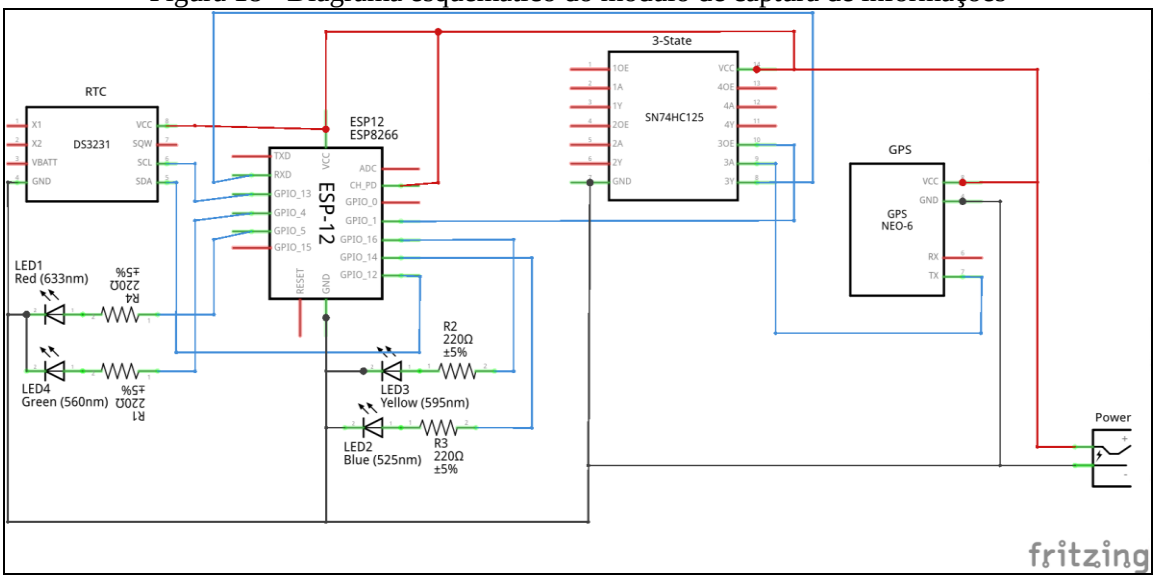


No diagrama exposto na Figura 17, os casos de uso 1 a 3 correspondem as operações que podem ser realizadas pelo usuário do transporte público; os casos de uso 4 a 8 correspondem as operações de manutenção do sistema e devem ser realizadas pelo administrador.

3.2.2 Diagrama esquemático

Na Figura 18 é especificado o diagrama esquemático do protótipo construído para a captura de informações de localização do ônibus.

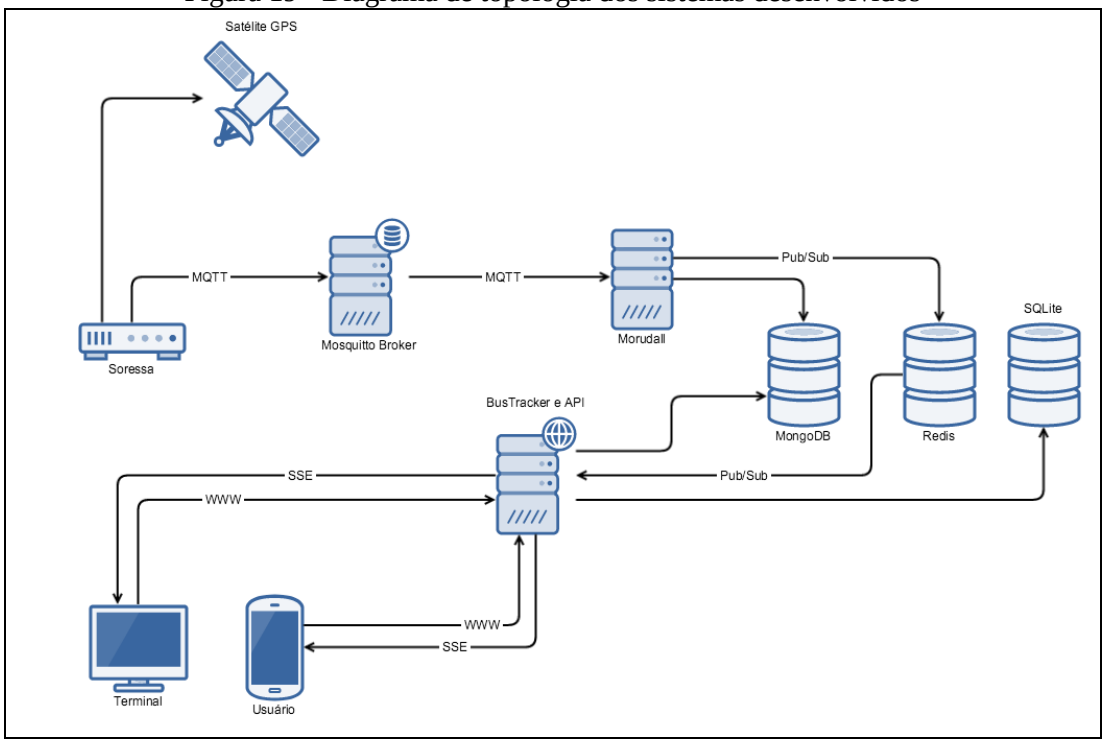
Figura 18 - Diagrama esquemático do módulo de captura de informações



3.2.3 Diagrama de topologia

A Figura 19 demonstra a topologia dos sistemas desenvolvidos funcionando em conjunto.

Figura 19 - Diagrama de topologia dos sistemas desenvolvidos



3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação. Para maior distinção entre as aplicações desenvolvidas elas serão nomeadas conforme a seguir:

a) BusTracker: aplicação Web para os usuários e *Web Service* para integração com outros sistemas;
b) Morudall: cliente MQTT responsável por receber as mensagens do *broker* MQTT e armazenar no banco de dados;
c) Soressa: aplicação do dispositivo embarcado no ônibus, responsável por enviar as informações ao *broker* MQTT.

3.3.1 Técnicas e ferramentas utilizadas

O trabalho foi desenvolvido na arquitetura cliente-servidor. A camada cliente é dividida em duas partes, uma responsável por informar a localização do veículo ao servidor via rede Wi-Fi (Soressa) e outra pela interface com o usuário (BusTracker), permitindo a consulta destas informações. A camada servidor é responsável pelo processamento e armazenamento destas informações (Morudall).

3.3.2 Partes do sistema

A seguir é detalhada a implementação de cada um dos sistemas que compõem este trabalho. Inicialmente é abordado o sistema responsável pelo envio da localização do ônibus (Soressa). Em seguida, é abordado o sistema responsável pelo armazenamento dos dados enviados pelo Soressa (Morudall). E, por fim, é abordado o sistema responsável pelo fornecimento das informações aos usuários do transporte público (BusTracker).

3.3.2.1 Sistema de posicionamento embarcado

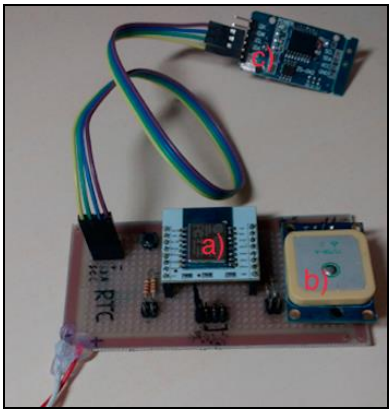
O Soressa, nome dado ao software que é executado pelo ESP8266, é responsável por controlar os módulos utilizados pelo microcontrolador e enviar os dados obtidos ao servidor MQTT. A partir deste protocolo, é possível fazer a comunicação com o servidor responsável pelo armazenamento das informações.

O servidor responsável por receber as mensagens do dispositivo é o Mosquitto. O Mosquitto é um *broker* de código aberto que implementa as versões 3.1 e 3.1.1 do protocolo MQTT usando o modelo *publish/subscribe* o que o torna adequado para a troca de mensagens entre dispositivos (MOSQUITTO, 2015?).

Para o desenvolvimento do software do dispositivo embarcado utilizou-se a linguagem de programação Lua. A utilização de Lua e o acesso aos GPIOs do dispositivo para controle dos módulos externos é feita por meio do NodeMCU.

O hardware do sistema desenvolvido pode ser observado na Figura 20 Ele pode ser dividido em 3 partes principais. a) um microcontrolador ESP12, responsável por executar o código Lua; b) módulo GPS; e c) módulo *Real Time Clock* (RTC). Além destes itens, o dispositivo conta com um circuito para controlar quando ativar o recebimento dos dados do GPS e LEDs indicadores de status.

Figura 20 - Hardware do sistema desenvolvido



No Quadro 6 são listados os valores das principais peças utilizadas no desenvolvimento do Soressa. A cotação foi feita em Dólar americano na loja virtual AliExpress.com.

Quadro 6 - Valores das peças utilizadas no projeto

PEÇA	**VALOR**
GY-NEO6MV2 GPS	US$ 12.00
ESP-12	US$ 2.50
SN74HC125	US$ 0.25
DS3231 AT24C32	US$ 1.00
LED 5mm Kit 100 peças colorido	US$ 1.50
Resistor 220ohm Kit 100 peças	US$ 1.00
Placa PCB	US$ 1.20

No trabalho optou-se pelo ESP12 pela sua quantidade de GPIOs disponíveis e interface I²C. Os GPIOs são necessários para controlar os LEDs de notificação, que são quatro no total. Cada cor representa um ou mais estados, conforme listado abaixo:

a) todos os LEDs acesos indica que o dispositivo está dando o boot;
b) vermelho acesa indica erro de conexão;
c) verde piscando indica que o dispositivo está se conectando a Wi-Fi;
d) verde acesa indica que o dispositivo se conectou com sucesso ao Wi-Fi;
e) amarela acesa indica que o cliente MQTT se conectou ao servidor com sucesso;

f) azul acesa indica que o GPS está ativo.

O módulo GPS utilizado no protótipo é o u-blox neo-6m, um GPS de baixo custo com boa precisão. Esse módulo é compatível com o protocolo NMEA 0183 versão 2.3 (U-BLOX AG, 2011). Para a conexão serial com o ESP12, foi utilizada a interface UART disponível em ambos os dispositivos. Como RTC foi utilizado o DS3132, um RTC de baixo custo, alta precisão e com interface I²C (MAXIN INTEGRATED, 2015).

O software desenvolvido para o dispositivo está divido em três módulos, sendo eles `led.lua`, `time.lua` e `soressa.lua`. O modulo `led` é responsável por controlar as notificações de status. No Quadro 7 são exibidas as funções `led.on` e `led.off`, responsáveis por controlar um LED. Como parâmetro é esperado o GPIO onde o LED está conectado.

Quadro 7 - Funções para ligar e desligar um LED

```
local function on(led)
    gpio.write(led, gpio.HIGH)
end

local function off(led)
    gpio.write(led, gpio.LOW)
end
```

No Quadro 8 são exibidas as funções `led.start_blink` e `led.stop_blink`, responsáveis por fazer o LED piscar. Para manter o LED piscando, utilizou-se a função `tmr.alarm`. Esta função cria um temporizador, que neste caso executa a cada 120 milissegundos.

Quadro 8 - Função responsável por fazer o LED piscar.

```
local function start_blink(led)
    gpio.write(led, gpio.LOW)

    -- temporizador de 120 ms.
    tmr.alarm(0, 120, 1, function ()
        tmr.wdclr()

        if (led_state) then
            -- ligar led.
            gpio.write(led, gpio.HIGH)
        else
            -- desligar led.
            gpio.write(led, gpio.LOW)
        end
        led_state = not led_state
    end)
end

local function stop_blink()
    tmr.stop(0)
end
```

O modulo `time` é responsável por controlar o RTC. No Quadro 9 é possível observar a conversão de *Binary-coded decimal* (BCD) para decimal.

Quadro 9 - Função para converter BCD para decimal

```
local function to_decimal(value)
    local hl = bit.rshift(value, 4)
    local hh = bit.band(value, 0xf)
    return string.format("%d%d", hl, hh)
end
```

No Quadro 10 é possível observar como é efetuada a leitura da data/hora. O endereço `0x068` representa o endereço I²C do DS3231, já o endereço `0x00` é a posição inicial de leitura. Primeiramente, é transmitido para o RTC qual a posição que será lida, em seguida são requisitados sete bytes, que nesse caso são: segundo; minuto; hora; dia da semana; dia do mês; mês e ano.

Quadro 10 - Função de leitura da hora do RTC

```
local function time()
    -- define qual posição inicial será lida.
    i2c.start(0)
    i2c.address(0, 0x68, i2c.TRANSMITTER)
    i2c.write(0, 0x00)
    i2c.stop(0)

    -- define qual o endereço do RTC.
    i2c.start(0)
    i2c.address(0, 0x68, i2c.RECEIVER)

    -- le a hora atual.
    local bytes = i2c.read(0, 7)
    i2c.stop(0)

    local second, minute, hour, day_of_week, day_of_month, month, year

    second = to_decimal(string.byte(bytes, 1))
    minute = to_decimal(string.byte(bytes, 2))
    hour = to_decimal(string.byte(bytes, 3))
    day_of_week = to_decimal(string.byte(bytes, 4))
    day_of_month = to_decimal(string.byte(bytes, 5))
    month = to_decimal(string.byte(bytes, 6))
    year = to_decimal(string.byte(bytes, 7))

    -- retorna data no formato ISO 8601.
    return string.format("20%s-%s-%s-T%s:%s:%s",
                         year, month, day_of_month,
                         hour, minute, second)
end
```

O módulo `soressa` é responsável por controlar o GPS, o MQTT, o Wi-Fi e os demais módulos. Além disso, esse módulo é o ponto inicial do software, ficando em execução no ESP8266 até que seja desligado. O módulo `soressa` também é responsável por conectar-se à Internet e ao MQTT. Após essas conexões estiverem estabelecidas, é iniciado o processo de envio de informações do GPS e da rede para que o servidor possa armazenar e processar esses dados. No Quadro 11 é possível observar como os dados são enviados ao servidor.

Quadro 11 - Função para enviar os dados ao servidor

```
local function send_data(channel, data)
    local ts = time.time()
    -- monta JSON a ser enviado ao Morudall.
    local msg = string.format(
        "{\"data\":\"%s\",\"ts\":\"%s\",\"id\":\"%s\"}",
        data, ts, DEVICE_ID)

    publish(channel, msg)
end
```

A mensagem contém o dado, que pode ser do GPS ou do AP, a hora de leitura destes dados, e o identificador do dispositivo. Cada dado é publicado em um canal diferente. Para o GPS, é utilizado o canal `/gpslocation`. Já para o AP é utilizado o canal `/accesspoint`. No Quadro 12 é possível observar como é criada a conexão com o MQTT, para que seja possível enviar as mensagens.

Quadro 12 - Inicialização do cliente MQTT

```
mq = mqtt.Client("soressa", 60, MQTT_USER, MQTT_PWD)
mq:lwt("/lwt", "offline", 0, 0)
mq:on("offline", function () led.off(led.MQTT)
   -- código omitido
end)

function start_all()
    mq:connect(MQTT_ADDR, MQTT_PORT, 0, function ()
        -- código omitido
    end)
end)
```

O modulo GPS envia mensagens no formato ASCII. Como o ESP8266 não possui memória suficiente, esta mensagem não é processada por ele, ficando a cargo do servidor processar a mensagem e descobrir se ela é valida ou não. A única validação feita é para identificar se a mensagem recebida é a esperada. Neste trabalho optou-se pelas mensagens que contêm latitude, longitude e velocidade do veículo. No Quadro 13 é possível observar o processamento da mensagem pelo ESP.

Quadro 13 - Função responsável por controlar os dados recebidos do GPS

```
uart.on("data", "\n", function (data)
    -- verifica se é uma sentença NMEA.
    if (string.sub(data, 1, 1) == "$") then
        local sentence = string.sub(data, 2, 6)

        -- verifica se é uma sentença aceita pelo Morudall.
        if sentence == "GPGLL" or
           sentence == "GPGGA" or
           sentence == "GPRMC" or
           sentence == "GPVTG"
        then
            -- envia a sentença para o Broker.
            send_data("/gpslocation",
                      string.sub(data, 1, string.len(data) - 2))
        end
    end
end, 0)
```

As sentenças `GPGGA` e `GPGLL` são responsáveis por informar a localização do receptor GPS no solo. A sentença `GPRMC` é responsável por fornecer a localização e a velocidade do receptor. Já a sentença `GPVTG` é responsável por fornecer a velocidade do receptor.

3.3.2.2 Processamento e Armazenamento das informações de posicionamento

A aplicação responsável por processar e armazenar as informações recebidas do dispositivo foi desenvolvida na linguagem de programação Python. Para fazer a implementação do cliente MQTT no Python utilizou-se a biblioteca Paho. O Paho é uma biblioteca desenvolvida pela Eclipse Foundation e implementa as versões 3.1 e 3.1.1 do protocolo MQTT (ECLIPSE FOUNDATION, 2015).

O Morudall, nome dado à aplicação, recebe as mensagens enviadas pelo Soressa e as armazena no MongoDB. Antes de salvar estas informações, é feito um pré-processamento para extrair as informações desejadas da sentença NMEA e persisti-las de uma forma estruturada que permita efetuar consultas sem a necessidade de decodificar a sentença novamente. Ao término do processamento da mensagem é notificado o BusTracker caso esta seja uma mensagem válida. Mesmo que a mensagem seja inválida ela, é armazenada no banco de dados.

Para fazer essa decodificação da mensagem foi implementado um *parser* parcial do protocolo NMEA 0183 v2.3. A implementação é parcial, pois trata apenas as mensagens envidas pelo Soressa, que são as sentenças `GGA`, `GLL`, `RMC` e `VTG`. Optou-se por desenvolver um *parser* próprio, pois a biblioteca disponível para Python não extraia as informações da sentença no formato utilizado pela aplicação.

No Quadro 14, pode-se observar a implementação base de um *parser* de uma sentença NMEA. Para implementar o *parser* de uma sentença (Quadro 15), basta informar o nome e a descrição da sentença, quais são os campos que compõem a sentença e o tipo Python corresponde ao valor do campo.

Quadro 14 - Parte da classe base de um *parser* de sentença

```
class Sentence(object):
    sentence = 'Unknown'
    sentence_description = 'Unknown Sentence'
    fields = ()

    def parse(self, data):
        data = str(data)
        # remove o checksum.
        sentence, checksum = data.split('*')
        raw_fields = sentence.split(',')

        # verifica se a sentença recebida tem a
        # mesma quantidade de campos que o esperado.
        if len(raw_fields) != self._fields_count:
            raise ParseException(
                'Field count mismatch. Expected %d fields, but found %d.' %
                (self._fields_count, len(raw_fields)))

        # para cada campo converte-se o valor.
        for index, field in enumerate(self.fields):
            field_name, _, field_type = field
            try:
                value = raw_fields[index]
                value = value.strip()

                if value:
                    setattr(self, field_name, field_type(value))
                else:
                    setattr(self, field_name, None)
            except:
                raise ParseException(
                    'Can\'t parse value into field "%s": %s' %
                    (field_name, value))

        return self
```

Quadro 15 - Implementação do *parser* da sentença `GLL`

```
class GLLSentence(Sentence):
    sentence_name = 'GLL'
    sentence_description = 'Geographic Position - Latitude/Longitude'
    # campos da sentença e seu respectivo tipo Python.
    fields = (
        ('latitude', 'Latitude', float),
        ('ns_indicator', 'N/S Indicator', str),
        ('longitude', 'Longitude', float),
        ('ew_indicator', 'E/W Indicator', str),
        ('utc_time', 'UTC Time', UTCTimeParser),
        ('status', 'Status', str),
        # ('mode', 'Mode', str), NMEA V 3.00
    )

    def __init__(self):
        super(GLLSentence, self).__init__()
```

O Soressa é responsável por publicar as mensagens no *broker* MQTT, já o Morudall é responsável por escutar estas mensagens e processá-las assim que recebidas. No Quadro 16 é possível observar como as mensagens do canal `/gpslocation` são tratadas. Como o Soressa publica mensagens em dois canais distintos, deve-se escutar ambos os canais. Todas as

mensagens são armazenadas de forma estruturada em dicionários, que no MongoDB é representado por um *JSON Object*. Ao armazenar as informações de forma estruturada, a obtenção dos dados por outras aplicações é facilitada.

Quadro 16 - Tratamento das mensagens recebidas

```
if msg.topic == '/gpslocation':
    try:
        # decodifica o JSON recebido do Soressa.
        payload = json.loads(msg.payload)
        device_id = payload['id']
        dt = dateutil.parser.parse(payload['ts'])
        nmea_sentece = payload['data']

        try:
            # decodifica a sentença NMEA.
            sentence = nmea.Parser().parse(nmea_sentece)
            extra = extract_gps_data(sentence)
            gps_data = sentence.to_dict()
        except:
            traceback.print_exc()
            gps_data = None
            extra = {}

        data = {
            'device': device_id,
            'time': dt,
            'raw_gps': nmea_sentece,
            'gps_data': gps_data
        }

        data.update(extra)
        # persiste a informação no banco de dados.
        self._save_gps_data(data)
    except:
        traceback.print_exc()
```

Todo o dado enviado pelo dispositivo é armazenado no MongoDB. Essa escolha foi feita, pois as informações são de cunho estatístico e de larga escala. Como o MondoDB é um banco de dados não relacional (NoSQL), ele oferece uma maior flexibilidade para o armazenamento de dado não estruturado. Outro ponto muito importante na escolha foi o desempenho em relação a um banco de dados relacional. Com uma frota muito grande de ônibus, os dados armazenados tendem a crescer exponencialmente, inviabilizando o uso de um banco de dados relacional.

3.3.2.3 Gerenciamento de rotas e frota

A aplicação Web e o *Web Service* são unificadas em uma única aplicação e foram desenvolvidas utilizando o *framework* Django. O Django é um *framework* Web para Python, e esta escolha foi feita pelo seu desempenho, segurança, possibilidade de escalabilidade e desenvolvimento ágil. O Django já possui ferramentas como Controle de Acesso,

Gerenciamento de Banco de Dados com *Object-relational mapping* (ORM) e *Template Engine*, importantes para o desenvolvimento do projeto (DJANGO SOFTWARE FOUNDATION, 2015).

A camada de apresentação faz uso do Bootstrap, Font Awesome, Javascrip e jQuery. Para melhorar a experiência com o usuário, entregando informações em tempo real, utilizou-se *Server Sent Events* (SSE) para receber as notificações enviadas pelo Morudall e a Google Maps Javascript API para o controle dos mapas.

O *Web Service* disponibilizado para integração com sistemas de terceiros utiliza a arquitetura *Representational State Transfer* (REST) e oferece apenas métodos de consulta. Desse modo, a alteração das informações apenas para as entidades reguladoras de trânsito ou empresas responsáveis pelo transporte público. A escolha da arquitetura REST se deve à sua simplicidade e facilidade de implementação em diversas linguagens de programação, permitindo que a aplicação BusTracker seja integrada com uma variedade de sistemas.

As informações utilizadas pela aplicação Web, bem como o cadastro de frotas, rotas e terminais é feita em um banco de dados relacional. A escolha de um banco de dados relacional nesse caso é pela garantia na consistência dos dados. Assim optou-se pelo SQLite pela sua facilidade de configuração e praticidade de uso.

Além do SQLite, o Redis, um banco estruturado em memória, é usado como fila de mensagens *pub/sub* para disparar as notificações enviadas pelo Morudall ao BusTracker. Optou-se pelo Redis por ele suportar o paradigma de mensagem *pub/sub*, evitando a necessidade de implementação de um outro servidor apenas para tratar distribuição de mensagens entre aplicações distintas.

O BusTracker possui seis módulos, além dos já disponibilizados pelo próprio Django, sendo eles:

a) app.api: módulo contendo a definição e o controle de acesso do *Web Service*;
b) app.core: módulo contendo os modelos ORM (representam as tabelas) da aplicação;
c) app.settings: módulo responsável por armazenar as configurações do sistema;
d) app.site: módulo contendo os *templates* do site;
e) bustracker: módulo que contém a configuração de aplicação Django;
f) util: módulo contento métodos e classes utilizados pela aplicação.

A interface administrador faz uso do Django Admin, um módulo que permite criar *Forms* dinâmicos com base nos modelos ORM. No Quadro 17 é possível observar a definição de um modelo e no Quadro 18 é exibido como esse modelo será renderizado na interface.

Quadro 17 - Modelo ORM da classe `Bus`

```
class Bus(models.Model):
    # campos da tabela.
    is_active = models.BooleanField(verbose_name=_(u'Ativo'),
                                    default=True)
    name = models.CharField(verbose_name=_(u'Nome'), max_length=100)
    details = models.TextField(verbose_name=_(u'Detalhes'),
                               max_length=250, blank=True, null=True)
    device_id = models.CharField(verbose_name=_(u'ID do Dispositivo'),
                                 max_length=17, help_text='MAC Address.')
    route = models.ForeignKey(BusRoute, related_name='bus_set',
                              verbose_name=_(u'Rota'))

    def __unicode__(self):
        return self.name

    class Meta:
        # descrição da tabela.
        verbose_name = _(u'Ônibus')
        verbose_name_plural = _(u'Ônibus')
```

Quadro 18 - Modelo administrados da classe `Bus`

```
class BusAdmin(admin.ModelAdmin):
    # campos da interface Admin.
    list_display = ('id', 'name', 'is_active')
    list_filter = ('is_active',)
    ordering = ('id',)
    raw_id_fields = ('route',)
    search_fields = ('name',)

admin.site.register(Bus, BusAdmin)
```

Para efetuar os cálculos de distância e tempo entre pontos foi utilizado o Google Maps API. Optou-se por este meio, pela possibilidade de calcular a distância entre dois pontos baseando-se em rotas e não em uma linha reta, aumentando assim a acurácia das informações disponibilizadas. No Quadro 19 é exemplificado como é feita a chamada a API do Google Maps e o processamento feito em cima do resultado.

Quadro 19 - Chamada a API do Google Maps

```
@lru_cache(maxsize=None, typed=True)
def get_distances(origins, destinations):
    try:
        origins = list(origins)
        destinations = list(destinations)

        gmaps = googlemaps.Client(key=API_KEY)
        # requisita para a API do Google Maps
        # qual a distância entre origem e destino.
        result = gmaps.distance_matrix(origins, destinations,
                                       mode='driving', units='metric')

        if result['status'] == 'OK':
            matrix = {}

            # monta uma matriz com os resultados
            # das distâncias entre os pontos.
            for i, origin in enumerate(origins):
                for j, destination in enumerate(destinations):
                    data = result['rows'][i]['elements'][j]
                    if data['status'] == 'OK':
                        matrix[(origin, destination)] = Distance(data)

            if matrix:
                return matrix
    except:
        traceback.print_exc()

    return None
```

Na função acima é utilizada a API do Google Maps para calcular a distância entre vários pontos. Após receber o resultado da API, o resultado é interpolado em uma matriz, em que a linha armazena os pontos de origem e de destino e o valor da coluna é a distância entre esses dois pontos.

A função do Quadro 19 é utilizada para descobrir qual o ponto de parada mais próximo do local do ônibus. No Quadro 20 é possível observar como a função é utilizada para efetuar tal cálculo.

Quadro 20 - Calculo do ponto mais próximo

```
@property
def next_stop(self):
    if not self.location_available:
        return None

    stops = self.route.stops.all()

    # se possuir paradas configuradas,
    # descobre qual a parada mais próxima.
    if stops.count() > 0:
        bus_lat, bus_lon = self.current_location_cached
        to_lat, to_lon = self.route.to_stop.location
        bus_distance = get_directions(bus_lat, bus_lon,
                                      to_lat, to_lon).meters

        pos_list = [(x.latitude, x.longitude) for x in stops]
        # calcula a distância do ônibus em relação
        # a todas as paradas.
        distances = get_distances([(to_lat, to_lon)], pos_list)

        if distances:
            d = distances.values()
            # retorna as paradas entre o ônibus
            # e a última parada.
            nexts = filter(lambda d: d.meters < bus_distance, d)

            if len(nexts) > 0:
                # retorna a parada de menor distância.
                distance = sorted(nexts, key=lambda item: item.meters)[0]
                key = distances.keys()[distances.values().index(distance)]
                index = pos_list.index(key[1])
                return stops[index]

    return self.route.to_stop
```

A função acima utiliza a API do Google Maps para calcular a distância do ônibus em relação a todos os pontos de parada da rota percorrida. Para se descobrir qual o próximo ponto, é calculada a distância entre o ônibus e o último ponto de parada. Depois, é calculada a distância entre todos os pontos da rota e o último ponto de parada. A partir da distância do ônibus em relação ao terminal e dos demais pontos de parada, é possível inferir o próximo ponto caso esse possua uma distância menor que a do ônibus e esta distância seja a maior distância até o último ponto.

3.3.3 Operacionalidade da implementação

Para melhor distinção entre as partes do sistema, este capítulo será dividido em três partes. Inicialmente será apresentado o sistema Web de consulta de informações. Em seguida será apresenta o sistema Web de cadastro (administrador) e, por fim, será apresentada a operacionalidade do dispositivo de rastreamento.

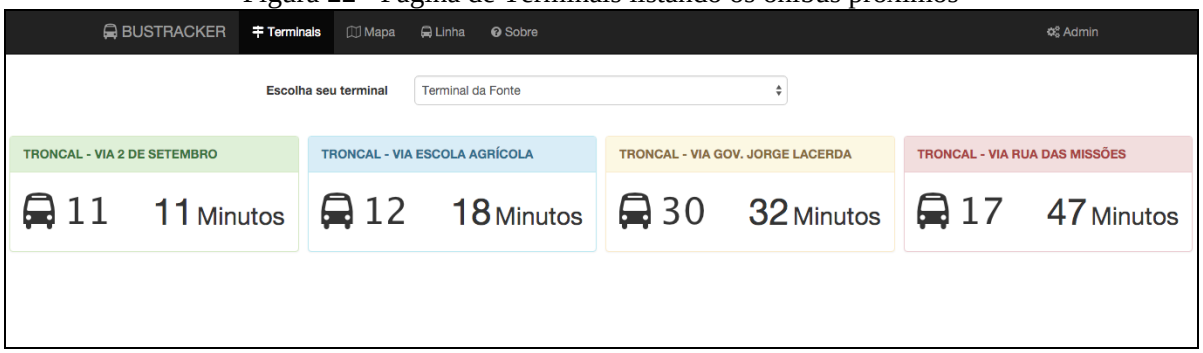
3.3.3.1 Sistema de consulta de informações

O sistema de consulta de informações possui foco no usuário do transporte público. Na Figura 21 é possível observar a página inicial do sistema.

Figura 21 - Página inicial da aplicação

Na parte superior da Figura 21 é demonstrado o menu principal da aplicação, disponível em todas as telas e que permite a navegação entre as demais páginas da aplicação. Na Figura 22 são exibidos os ônibus com paradas no terminal escolhido. A página é atualizada automaticamente quando um ônibus envia sua posição ao servidor.

Figura 22 - Página de Terminais listando os ônibus próximos

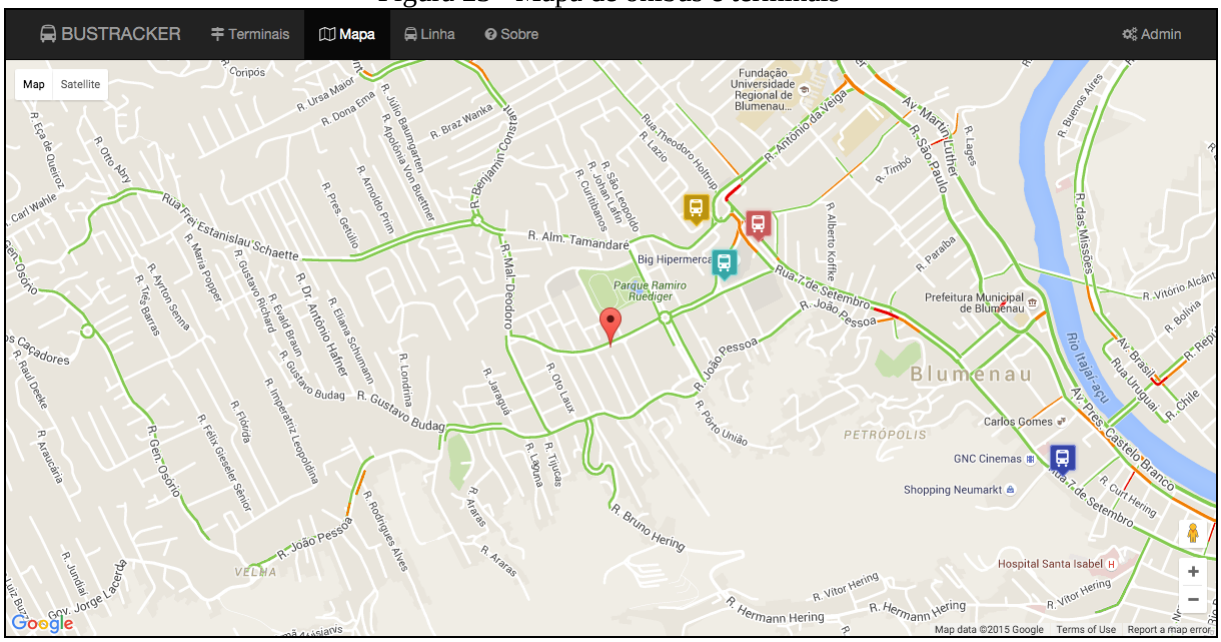


Na Figura 22 é possível observar que existe um campo para escolher qual terminal o usuário deseja visualizar. Por padrão, se for possível obter a localização atual do usuário, é exibido inicialmente o nome do terminal mais próximo de sua localização. Caso o usuário deseje visualizar outro terminal, basta alterar o nome neste campo.

Após a escolha, é exibido a lista de ônibus que fazem a próxima parada no terminal escolhido. Para cada ônibus é exibido o seu número, nome de rota e o tempo que falta para chegar no terminal escolhido. Ainda na Figura 22, é possível observar que a página apresenta quatro seções que representam uma determinada faixa de tempo até a chegada no terminal. A cor verde simboliza menos de 15 minutos; a cor azul simboliza de 16 a 30 minutos; a cor amarela simboliza de 31 a 45 minutos; por fim, a cor vermelha simboliza mais de 45 minutos até a chegada do ônibus.

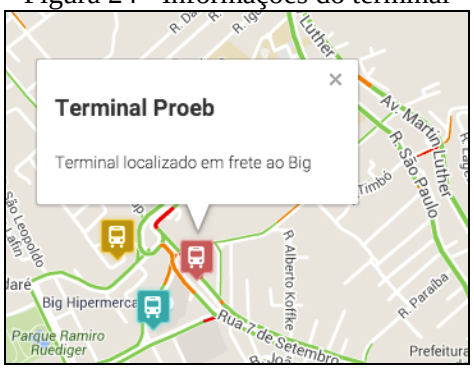
No sistema de consulta também é possível visualizar ônibus e terminais no mapa, conforme exibido na Figura 23. O mapa utilizado é o Google Maps, com uma customização para exibir os ônibus, terminais, pontos e garagens, além da localização atual do usuário, caso esteja disponível.

Figura 23 - Mapa de ônibus e terminais



Como se pode observar, existem diferentes ícones no mapa. O pino vermelho é a posição aproximada do usuário; o ícone amarelo representa a posição do ônibus; o ícone vermelho representa um terminal; o ícone azul-escuro representa um ponto de ônibus; por fim, o ícone azul-claro, representa uma garagem de ônibus. Ao se clicar nos ícones, é possível observar informações do local. Como se pode observar na Figura 24, é exibido o nome do terminal e uma descrição.

Figura 24 - Informações do terminal

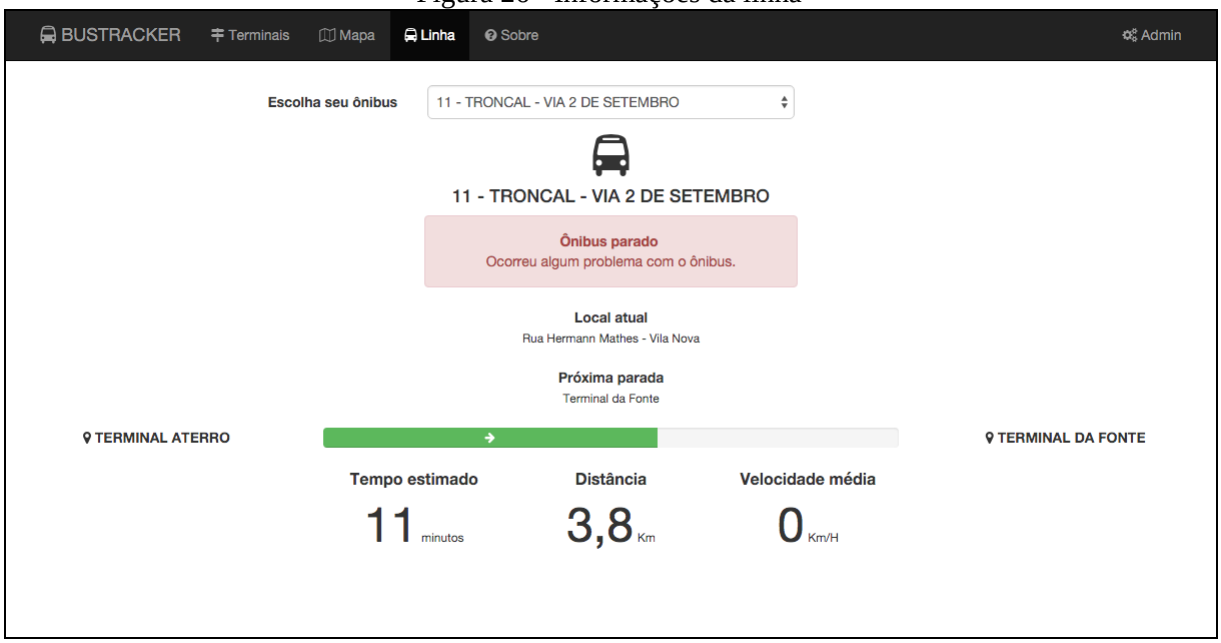


Na Figura 25 é exibido o número e nome da rota do ônibus, bem como o trajeto percorrido. Assim como a de terminais, essa página também é atualizada automaticamente.

Figura 25 - Informações do ônibus

As informações detalhadas do ônibus são exibidas na página da linha, representada na Figura 26. Assim como a página de terminais e o mapa, essa também é atualizada automaticamente.

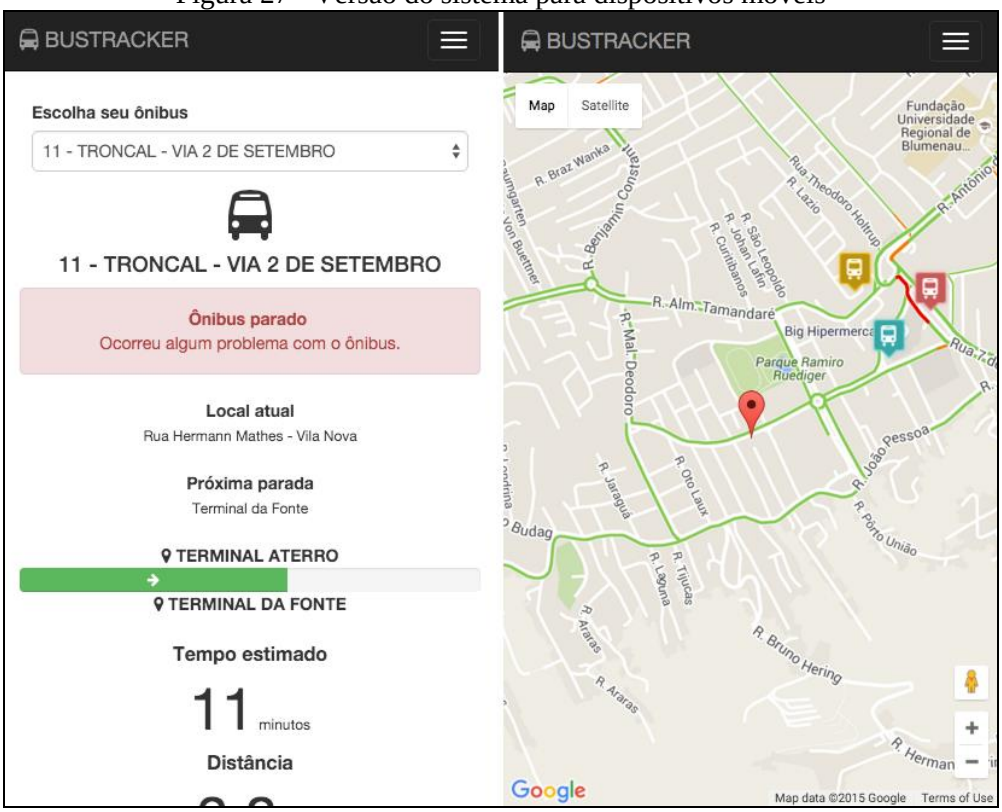
Figura 26 - Informações da linha



Nesta página é possível escolher o ônibus, porém o usuário também pode chegar até ela a partida da página de terminais ou do mapa. As informações são exibidas se estão disponíveis. Se, por algum motivo, o ônibus perder conexão com a Internet por mais de 5 minutos, é exibido um alerta na página informando que os dados podem estar inconsistentes. O mesmo ocorre quando o ônibus não altera sua posição por mais de 5 minutos. Nesse caso, é informado que pode ter ocorrido um problema com o ônibus.

Por fim, a Figura 27 mostra como o sistema é apresentado na versão para dispositivos móveis. Essa versão é importante para o usuário final, pois permite que ele consulte as informações de seu ônibus em qualquer lugar que ele esteja.

Figura 27 - Versão do sistema para dispositivos móveis



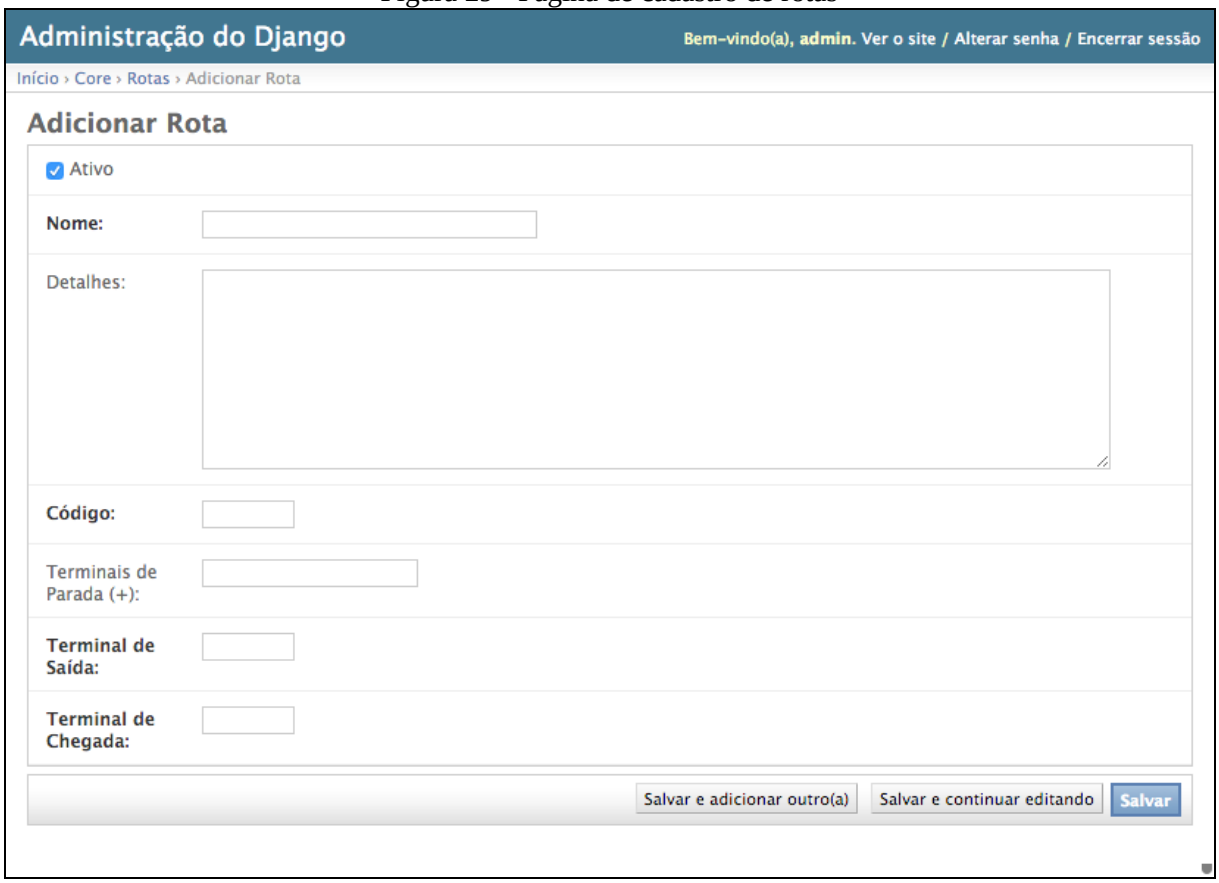
3.3.3.2 Sistema de administração

O sistema conta com uma interface de administrador para efetuar cadastro de ônibus, rotas, terminais, usuários, grupos, permissões, configurações e chaves de acesso a API. Na Figura 28 é possível observar a página principal do sistema.

Figura 28 - Página principal do sistema administrador

Para acessar esta parte do sistema, o usuário deve se autenticar com usuário/senha. Apenas quem é administrador deve possuir acesso a essas páginas, ou seja, a empresa que gerencia a frota de ônibus ou algum órgão relacionado a empresa. Na Figura 29 é exemplificado como é realizado o cadastro de uma nova rota.

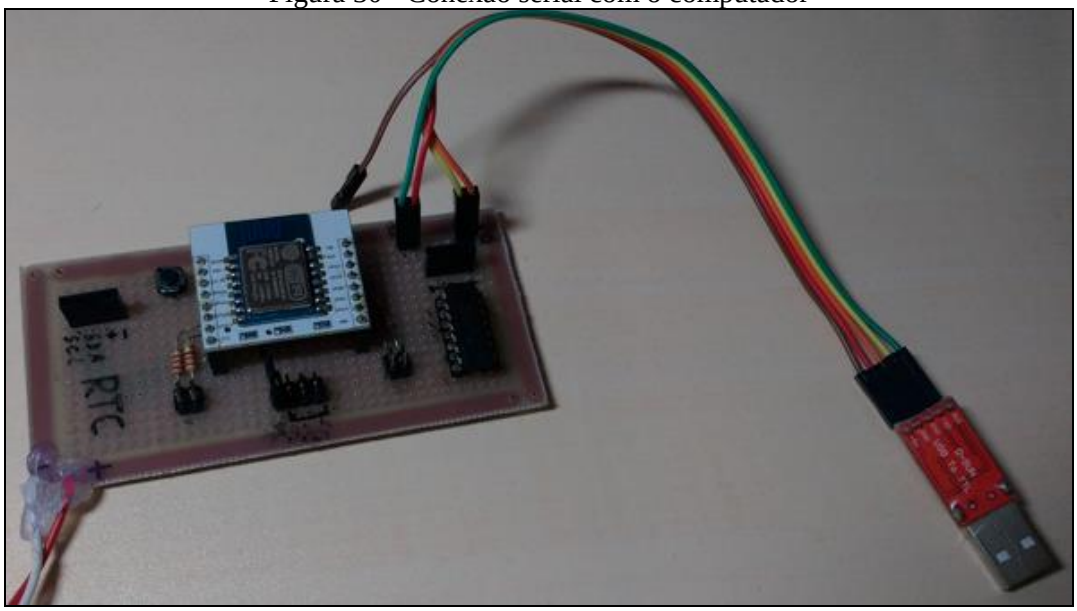
Figura 29 - Página de cadastro de rotas



3.3.3.3 Dispositivo de rastreamento

O dispositivo de rastreamento é único para cada ônibus, e sua identificação é feita através do *MAC address*. Para que o rastreamento funcione corretamente é necessário cadastrar o *MAC address* no momento que se adiciona um novo ônibus no sistema de administração. Como o dispositivo não possui nenhuma forma de exibição de informações em um visor, é necessário se conectar pela porta serial para obter o *Mac address*. Na Figura 30 é possível observar como conectar o dispositivo ao computador.

Figura 30 - Conexão serial com o computador



Para que o dispositivo se conecte a redes Wi-Fi e ao *broker* MQTT, é necessário adicionar dois arquivos de configuração. O arquivo `wpa.conf` é responsável por conter os dados das redes. Já o arquivo `mqtt.conf` é responsável por conter as informações de conexão com o MQTT.

Depois de configurado, basta ligar o dispositivo e aguardar alguns segundos até a sua inicialização. Depois de inicializado, o dispositivo enviará para o servido a cada 15 segundos os dados do GPS, a hora atual e o ID do dispositivo. O mesmo ocorre quando o dispositivo se conecta a uma rede nova. Nesse caso, ao se conectar a uma rede Wi-Fi, ele enviará os dados da rede.

3.4 RESULTADOS E DISCUSSÕES

Os testes foram realizados com um veículo com conexão (modo *online*) e sem conexão (modo *off-line*) Wi-Fi disponível. Ambos os modos de operação devem funcionar no sistema desenvolvido, porém, no modo sem conexão, o sistema se torna limitado. No modo *online*, foi utilizada a rede 3G/4G de um *Smartphone* servindo de *access point* para o dispositivo de rastreamento.

Nos testes sem conexão à Iinternet, não é possível obter a posição do veículo em tempo real, não sendo possível informar ao usuário o local atual e o tempo de chegada até o terminal. Porém, quando o veículo se aproxima de um roteador conhecido e se conecta a rede, ele dispara uma mensagem ao servidor informando que se conectou e, se esde roteador for de um terminal, ele alertará o usuário que seu ônibus chegou. Caso este roteador seja um outro qualquer, será assumido o fluxo de um veículo no modo *online*.

Nos testes com conexão à Internet, que é o modo ideal de operação, o sistema é capaz de informar a localização do veículo com uma precisão de 100-250 metros aproximadamente. Como o cálculo do tempo estimado de chegada depende de vários fatores, o valor é sempre aproximado, pois o cálculo considera apenas as melhores condições e rotas possíveis para o cálculo. Assim como no modo de operação *off-line*, o sistema notificará o usuário caso o ônibus esteja próximo, porém, nesse caso, será utilizada a localização atual do ônibus para inferir se este está próximo ou não do terminal.

Durante a fase de implementação e os testes iniciais foram encontrados alguns problemas em relação ao ESP8266 e ao NodeMCU. Para contornar um problema causado pelo GPS foi necessário o uso de um circuito *tri-state*. O módulo GPS, diferente dos outros módulos, envia mensagens constantemente para a porta serial do ESP8266. Um grande volume de dados pode causar estouro de memória no ESP8266, sendo assim, é necessário limitar o acesso ao serial do dispositivo. Outro problema encontrado foi o fato de o GPS iniciar antes do próprio Soressa configurar a entrada serial, isto em alguns casos também ocasionava problema no código em execução, pois as instruções NMEA se misturavam as outras instruções enviadas pela porta serial, dificultando o modo de depuração durante o desenvolvimento.

Ainda em relação ao GPS, como ele enviava várias mensagens por segundo, o circuito *tri-state* se fez necessário para limitar o tempo que o GPS fica enviando mensagens ao servidor. Se O GPS ficar enviando mensagens constantemente o módulo MQTT apresenta problemas com uma frequência considerável. O mais comum é o NodeMCU travar, demorando alguns segundos até reiniciar. Esse problema se apresentava sempre após uns 10 minutos enviando mensagem constantemente ao servidor. Com a limitação de execução do GPS para um intervalo de 15 segundos, esse problema não ocorreu mais.

Durante o desenvolvimento inicial do Soressa, foram encontrados problemas relacionados a falta de memória do ESP8266. Nas primeiras versões o software não conseguiu ficar executando por mais de um minuto. Após alguns estudos mais aprofundados sobre o NodeMCU, identificou-se no artigo *nodeMCU Unofficial FAQ* (ELLISON, 2015), que o Lua do NodeMCU funciona de um modo diferente, sendo necessário programar de uma maneira otimizada e orientada a eventos. Na versão inicial, havia-se utilizado o conceito de módulos, porém esta abordagem consumia muita memória no dispositivo. A solução foi diminuir drasticamente o código desenvolvido, resultando em 3 módulos sucintos, contendo apenas o necessário para o funcionamento.

Para diminuir ainda mais o consumo de memória, efetuou-se a compilação do código do Soressa para *bytecode* Lua e também a compilação de um *firmware* específico para este projeto, contendo apenas os módulos do NodeMCU necessários para o funcionamento da aplicação. A compilação em *bytecode* reduz em cerca de 60% o consumo de RAM, pois é removido todo o código de depuração e informações de variáveis (ELLISON, 2015).

Ainda assim ocorreram problemas de falta de memória ou travar com uma frequência menor. Porém, nos testes efetuados, o Soressa ficou em execução por mais de uma hora enviando mensagens ao servidor sem nenhum problema. Uma margem de falhas é aceitável, desde que esta não atrapalhe a operação do protótipo. Mesmo que ocorra algum erro inesperado no ESP8266 ou no NodeMCU, o NodeMCU reinicia e volta a funcionar normalmente, não sendo necessária nenhuma intervenção do operador do ônibus.

Uma limitação da aplicação Web é o seu uso em *Smartphones*. A aplicação não foi totalmente otimizada para estes aparelhos, sendo assim a experiência do usuário pode ser insatisfatória dependendo do tamanho e resolução da tela. Porém este item não descarta seu uso em tais aparelhos como visto na operacionalidade da implementação.

O cálculo das rotas entre um terminal e outro também é limitada, pois ela se baseia na melhor rota entre dois pontos. O ideal seria utilizar o trajeto que o ônibus percorre entre tais pontos. Baseando-se no trajeto percorrido é possível estimar o tempo de chegada com uma precisão maior. Porém, tendo em vista que as rotas traçadas pelo Google Maps são similares as rotas percorridas, isso não chega a ser um problema. Em cidades cujo Google Maps possui as rotas de ônibus disponíveis é possível informar no momento da consulta que o trajeto será percorrido via ônibus. Porém, no teste efetuado com a cidade de Blumenau, a API do Google não retornou nenhuma rota disponível para trajeto de ônibus, sendo assim, optou-se por utilizar o trajeto de trânsito.

O dispositivo não possui um meio de armazenamento para o modo de operação *off-line*, ou seja, mesmo que o dispositivo se conecte a Internet após ficar muito tempo desconectado, as informações obtidas neste meio tempo serão perdidas. No modo *online* as informações também não possuem garantia de entrega 100%, pois o cliente MQTT está configurado para entregar as mensagens até uma vez, porém se ocorrer erro, a mensagem é descartada. Como o dispositivo envia mensagens de 15 em 15 segundos, a perda de alguns pacotes não se torna algo problemático.

Uma limitação identificada quase no fim do desenvolvimento do projeto que pode vir a ser um problema no futuro, são os ônibus que mudam de rota durante o dia. Hoje é necessário substituir o aparelho por outro, pois cada aparelho está vinculado a um ônibus que está

vinculado a uma rota. Sendo assim, se o ônibus mudar de rota, o alerta de chegada não funcionará como esperado. Pois os tempos de chegada ao terminal são calculados com base nos terminais pelo qual ele passará. Caso a rota mude, ele pode nunca chegar ao terminal para qual está configurado.

4 CONCLUSÕES

No início do trabalho objetivou-se alertar os usuários do transporte público quanto tempo o ônibus levaria para chegar ao terminal e alertar quando este chegasse. Além destes objetivos, durante o desenvolvimento do trabalho percebeu-se que outras informações também poderiam ser de grande utilidade para o usuário, como por exemplo, a adição de um mapa na aplicação para permitir o usuário visualizar de forma simplificada em que rua o ônibus se encontra. Outra informação adicionada foi o estado do ônibus, a partir do qual é possível informar se o veículo está em movimento, sem conexão com a Internet ou parado por causa de algum problema técnico ou por estar na garagem. Essa informação ajuda o usuário a escolher uma rota alternativa caso o seu ônibus esteja com algum tipo de problema. Apesar dos problemas encontrados durante o desenvolvimento e das limitações do projeto, os objetivos foram alcançados.

A possibilidade de o usuário consultar informações de sua rota em qualquer lugar ou horário é de extrema importância, sendo assim, com a disponibilização da API no sistema desenvolvido, permite-se que este possa ir além da versão Web e atender uma gama maior de usuários com versões para dispositivos móveis.

As ferramentas e métodos de trabalho foram adequados para o desenvolvimento deste trabalho. A vantagem de utilizar um banco dados NoSQL permite que se possa executar cálculos estatísticos com grande facilidade utilizando as ferramentas do próprio MongoDB. O uso de Python, Django, sistema de mensageria MQTT e *pub/sub*, permite criar um sistema extensível e escalável. O banco de dados escolhido para a aplicação administradora não é recomendado para uso em aplicações de grande porte. O SQLite é simples, e atende os objetivos deste trabalho, porém ao utilizar com uma frota grande é recomendado configurar a aplicação para utilizar outro banco de dados relacional.

O ESP8266 e o NodeMCU apesar dos problemas apresentados são ferramentas com um grande potencial. Conforme novas versões são disponibilizadas, esse conjunto se torna mais estável, o que torna esse conjunto uma boa escolha para o desenvolvimento de aplicações relacionadas a internet das coisas. O ESP8266 já conta com um sucessor, que é o ESP32. Essa nova versão ainda está em fase de desenvolvimento, porém já é esperada uma melhoria em relação ao seu antecessor (SOUZA, 2015)

Conforme apresentado nos trabalhos correlatos, existem aplicações similares já em operação em algumas cidades do Brasil e do mundo. Infelizmente, todos os sistemas apresentados são proprietários. O SITBus é operado pela BHTrans apenas em Belo Horizonte,

já o Olho Vivo é operado pela SPTrans em São Paulo. O Moovit é o aplicativo que possui grande poder de crescimento pelo seu âmbito de aplicação social, porém sempre estará vinculado a adoção de uso pelos usuários do transporte público e não pela prestadora do serviço.

O trabalho desenvolvido é uma implementação que trabalha de forma similar ao SITBus e ao Olho Vivo, onde as informações são obtidas através de um hardware específico. A sua grande vantagem em relação aos demais, é a possibilidade de adoção por qualquer empresa prestadora de serviço público em qualquer cidade do Brasil ou do mundo. Ainda que não implementada uma forma de se operar o sistema por meio de coleta de dados dos Smartphone dos usuários, essa é uma possibilidade que deve ser analisada e que pode vir a ser desenvolvida caso este projeto seja estendido ou aderido por alguma empresa. Apesar de ser abordado apenas o uso em ônibus, a aplicação desenvolvida pode vir a ser adotada em outros meios de transporte.

4.1 EXTENSÕES

No decorrer do desenvolvimento do trabalho, foram observados alguns aspectos que podem ser aprimorados.

A implementação de um aplicativo para *Smartphones* é um aspecto a ser levado em consideração pelo grande uso desses dispositivos no dia a dia. Tal aplicativo não requer processamento algum, pois todo cálculo já é feito pela API. Seria necessário apenas uma interface de usuário.

O segundo aspecto é melhorar a conexão do dispositivo, pois hoje ele está limitado a Wi-Fi. Algumas empresas de transporte público disponibilizam Wi-Fi em seus veículos, porém seria importante contar com uma conexão *Global System for Mobile Communications* (GSM) ou 3G. Como o protocolo utilizado é o MQTT, é possível que com GSM já seja suficiente para enviar atualização da localização com um intervalo de alguns poucos minutos.

Como terceiro aspecto seria importante implementar um aplicativo de controle para atualizar de forma automatizada as informações contidas no dispositivo embarcado no ônibus. Tal aplicativo poderia ser uma aplicação para Smartphone, para facilitar a agilidade da atualização. O ESP8266 conta com modo de conexão AP, sendo assim o *Smartphone* poderia conectar-se a ele e efetuar as alterações das informações necessárias.

O quarto aspecto envolve a melhoria do sistema de posicionamento. Hoje o dispositivo conta apenas com o sistema GPS, porém a adição do sistema GLONASS melhoraria a precisão. Hoje é comum os *Smartphone* suportarem os dois ou mais sistemas de

posicionamento. Ainda no quesito de melhorar a precisão, se baseando no segundo aspecto, pode ser utilizado o sinal 3G para fazer a triangulação da localização e detectar a posição do ônibus dentro da cidade. A triangulação do sinal 3G seria importante em casos onde os satélites GNSS podem não estar visíveis, garantindo o envio contínuo da localização.

Como quinto aspecto seria recomendado implementar o sistema de rotas no projeto proposto, pois com a rota correta do ônibus é possível fazer cálculos mais assertivos em relação a posicionamento e tempo estimado de chegada.

Por fim, substituir o uso de Wi-Fi/GPS para detectar a distância do terminal por ZigBee, Beacons ou outro protocolo. No trabalho proposto utilizou-se a estrutura já disponível nos terminais. Porém, com a adição de uma nova tecnologia como ZigBee e 3G, seria possível identificar todos os pontos de parada e não apenas os terminais em que um ônibus passa, melhorando a acurácia do rastreamento.

REFERÊNCIAS

ANTP. Sistemas Inteligentes de Transportes. **Série Cadernos Técnicos**, São Paulo, v. 8, maio 2012. Disponível em: <http://www.antp.org.br/_5dotSystem/download/dcmDocument/2013/03/18/9AB9A3EB-97DC-4711-9751-162AD361D7F0.pdf>. Acesso em: 19 nov. 2015.

BHATTA, Basudeb. **Global Navigation Satellite Systems:** Insights into GPS, GLONASS, Galileo, Compass and Others. Boca Raton: CRC Press, 2011.

BHTRANS. **ANEXO VIII:** SITBus - Sistema Inteligente de Transporte do Município de Belo Horizonte. [Belo Horizonte], [2008]. Disponível em: <http://www.bhtrans.pbh.gov.br/portal/page/portal/portalpublicodl/Temas/Onibus/gestao-transporte-onibus-2013/080320_SITBus_Anexo_VIII.pdf>. Acesso em: 30 mar. 2015.

BURBANK, Jack L. et al. **Wireless Networking:** Understanding Internetworking Challenges. Hoboken: John Wiley & Sons, 2013.

CEUB. **Fundamentos GPS Sistema de Posicionamento Global.** [S.l.], mar. 2006. Disponível em: <http://www.ltc.ufes.br/geomaticsee/Modulo%20GPS.pdf>. Acesso em: 7 abr. 2015.

CHEN, Whei-Jen et al. **Responsive Mobile User Experience Using MQTT and IBM MessageSight.** Armonk: IBM Redbooks, 2014.

DAVIS, Harold. **Absolute Beginner's Guide to Wi-Fi Wireless Networking.** Indianapolis: Que Publishing, 2004.

DESAI, Pratik. **Python Programming for Arduino.** Birmingham: Packt Publishing Ltd, 2015.

DJANGO SOFTWARE FOUNDATION. **The Web framework for perfectionists with deadlines | Django**. [S.l], 2015. Disponível em: <https://www.djangoproject.com/>. Acesso em: 17 nov. 2015.

ECLIPSE FOUNDATION. **Paho - Open Source messaging for M2M.** [S.l], 2015. Disponível em: <https://www.eclipse.org/paho/clients/python/>. Acesso em: 16 nov. 2015.

EL-RABBANY, Ahmed. **Introduction to GPS:** The Global Positioning. Norwood, MA: Artech House, 2002.

ELLISON, Terry. **nodeMCU Unofficial FAQ.** [S.l.], jul. 2015. Disponível em: <http://www.esp8266.com/wiki/doku.php?id=nodemcu-unofficial-faq>. Acesso em: 10 nov. 2015.

ESPRESSIF SYSTEMS. **Espressif Smart Connectivity Platform:** ESP8266. [S.l.], out. 2013. Disponível em: <https://nurdspace.nl/images/e/e0/ESP8266_Specifications_English.pdf>. Acesso em: 31 mar. 2015.

EZELL, Stephen. **Intelligent Transportation Systems.** [Washington], jan. 2010. Disponível em: <http://www.itif.org/files/2010-1-27-ITS_Leadership.pdf>. Acesso em: 28 mar. 2015.

FLYNN, Michael J.; LUK, Wayne. **Computer System Design:** System-on-Chip. Hoboken: John Wiley & Sons, 2011.

GAST, Matthew. **802.11 Wireless Networks:** The Definitive Guide. 2nd ed. Sebastopol, CA: O'Reilly Media Inc., 2005.

HINCH, Stephen W. **Outdoor Navigation with GPS.** 3rd ed. Birmingham: Wilderness Press, 2010.

HIVEMQ. **MQTT Essentials Part 3:** Client, Broker and Connection Establishment. [S.l.], 2015. Disponível em: <http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment>. Acesso em: 17 out. 2015.

HOFMANN-WELLENHOF, Bernhard; LICHTENEGGER, Herbert; WASLE, Elmar. **GNSS – Global Navigation Satellite Systems:** GPS, GLONASS, Galileo, and more. Viena: Springer Science & Business Media, 2007.

IERUSALIMSCHY, Roberto; FIGUEIREDO, Luiz H.; CELES, Waldemar. **Manual de Referência de Lua 5.1.** [S.l.], 2008. Disponível em: <http://www.lua.org/manual/5.1/pt/manual.html>. Acesso em: 12 out. 2015.

_____. **The Evolution of Lua.** [S.l.], [2007]. Disponível em: <http://www.lua.org/doc/hopl.pdf>. Acesso em: 12 out. 2015.

IN-CIRCUIT. **Espressif Smart Connectivity Platform:** ESP8266EX. [S.l.], 2015. Disponível em: <http://shop.in-circuit.de/product_info.php?products_id=168&osCsid=943a17522877d254b4aa2c87b2b120f2>. Acesso em: 6 abr. 2015.

KOLBAN, Neil. **Kolban's Book on the ESP8266.** [S.l.], out. 2015. Disponível em: <http://neilkolban.com/tech/wp-content/uploads/2015/09/Kolbans-Book-on-the-ESP8266-October-2015.pdf>. Acesso em: 14 out. 2015.

KUNDU, Santanu; CHATTOPADHYAY, Santanu. **Network-on-Chip:** The Next Generation of System-on-Chip Integration. Boca Raton: CRC Press, 2014.

LUTZ, Mark. **Learning Python.** 5th ed. Sebastopol, CA: O'Reilly Media Inc., 2013.

MARTELLI, Alex. **Python in a Nutshell.** Sebastopol, CA: O'Reilly Media Inc., 2006.

MAXIN INTEGRATED. **DS3231:** Extremely Accurate I2C-Integrated RTC/TCXO/Crystal. [S.l], 2015. Disponível em: <http://datasheets.maximintegrated.com/en/ds/DS3231.pdf>. Acesso em: 12 nov. 2015.

MEIRELLES, Alexandre A. C. **Sistemas de Transportes Inteligentes:** aplicação da telemática na gestão do trânsito urbano. Belo Horizonte, abr. 2010. Disponível em: <http://www.ip.pbh.gov.br/ANO1_N1_PDF/ip0101meirelles.pdf>. Acesso em: 18 mar. 2015.

MONICO, João F. G. **Posicionamento pelo NAVSTAR-GPS.** São Paulo: UNESP, 2001.

MOOVIT. **At a Glance.** [S.l.], [2014]. Disponível em: <http://moovitapp.com/wp-content/uploads/2014/07/Moovit-At-A-Glance-USA-4-14.pdf>. Acesso em: 14 nov. 2015.

_____. **Moovit Embarks on Global Expansion, Picks Up $50M Series-C led by Nokia Growth Partners (NGP).** [S.l], [2015a]. Disponível em: <http://moovitapp.com/wp-content/uploads/2015/03/press-release-moovit-series-c-2015.pdf>. Acesso em: 14 nov. 2015.

_____. **Moovit Press.** [S.l], 2015b. Disponível em: <http://moovitapp.com/press/>. Acesso em: 14 nov. 2015.

MOSQUITTO. **An Open Source MQTT v3.1 Broker**. [S.l.], [2015]. Disponível em: <http://mosquitto.org/>. Acesso em: 28 out. 2015.

NODEMCU TEAM. **NodeMCU -- An open-source firmware based on ESP8266 wifi-soc.** [S.l.], 2014. Disponível em: <http://nodemcu.com/index_en.html>. Acesso em: 12 nov. 2015.

PEREIRA, Wallace F. **O uso de sistemas inteligentes para o aumento da eficácia do transporte público por ônibus:** o sistema de bilhetagem eletrônica. 2007. 111 f. Dissertação (Mestrado em Engenharia de Transporte) – Programa de Pós-graduação em Engenharia de Transporte, Universidade Federal do Rio de Janeiro, Rio de Janeiro.

PILON, José A. **Sistema de informação ao usuário do transporte coletivo por ônibus na cidade de Vitória-ES.** 2009. 152 f. Dissertação (Mestrado em Engenharia de Produção) - Programa de Pós-graduação em Engenharia de Produção, Universidade Tecnológica Federal do Paraná, Ponta Grossa.

RAO, G S. **Global Navigation Satellite Systems**. Tata McGraw-Hill Education: Nova Deli, 2010.

SANCHES, Alexandre M. **Planejamento de transporte urbano:** estudo de caso da linha verde em Curitiba. 2008. 131 f. Dissertação (Mestrado em Engenharia de Produção) - Programa de Pós-Graduação em Engenharia de Produção, Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2008.

SILVA, Danyela M. **Sistemas inteligentes no transporte público coletivo por ônibus.** 2000. 144 f. Dissertação (Mestrado em Engenharia de Produção) – Programa de Pós-graduação em Engenharia de Produção, Universidade Federal do Rio Grande do Sul, Porto Alegre.

SOUZA, Fábio. **ESP32:** o sucessor do ESP8266. [S.l.], nov. 2015. Disponível em: <http://www.embarcados.com.br/esp32-o-sucessor-do-esp8266/>. Acesso em: 18 nov. 2015.

SPTRANS. **Sistemas Informatizados para a Gestão Do Transporte Coletivo do Município de São Paulo.** São Paulo, maio 2009. Disponível em: <http://www.sptrans.com.br/pdf/biblioteca_tecnica/SISTEMAS_INFORMATIZADOS_PARA_A_GESTAO_DO_TRANSPORTE.pdf>. Acesso em: 30 mar. 2015.

SUAREZ, Joana. **Promessa para 2011, BHTrans 'engaveta' painéis eletrônicos.** Contagem, 2015. Disponível em: <http://www.otempo.com.br/cidades/promessa-para-2011-bhtrans-engaveta-pain%C3%A9is-eletr%C3%B4nicos-1.1003086>. Acesso em: 30 mar. 2015.

SUSSMAN, Joseph S. **Perspectives on Intelligent Transportation Systems (ITS).** Nova York: Springer Science & Business Media, 2008.

SWEIGART, Al. **Automatize tarefas maçantes com Python.** São Paulo: Novatec, 2015.

TEIXEIRA, Pedro. **Professional Node.js:** Building Javascript Based Scalable Software. Indianapolis: John Wiley & Sons, 2013.

U-BLOX AG. **NEO-6 u-blox 6 GPS Modules Data Sheet.** Thalwil, 2011. Disponível em: <https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf?utm_source=en%2Fimages%2Fdownloads%2FProduct_Docs%2FNEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf>. Acesso em: 4 nov. 2015.

WAHER, Peter. **Learning Internet of Things.** Birmingham: Packt Publishing Ltd, 2015.

WILLIAMS, Bob. **Intelligent Transport Systems Standards.** Norwood, MA: Artech House, 2008.