Solução para Instrumentação Ubíqua Utilizando Computação em Nuvem e Servidor de Borda WiFi

Douglas Adalberto Scheunemann[1]

[1]Programa de Pós-Graduação em Eletrônica e Computação
Centro de Politécnico - Universidade Católica de Pelotas - UCPEL.

Resumo. *Os avanços em tecnologias para IoT e computação em nuvem permitem que diversas áreas da engenharia sejam aperfeiçoadas, dentre elas a instrumentação, tornando emergente o termo Instrução Ubíqua. Nesta área porém, a necessidade de uso de protocolos de comunicação sem fio com servidores e o tratamento de uma rede heterogênea de sensores torna a criação de instrumentos ubíquos uma tarefa desafiadora para os projetistas. Neste cenário é adequado considerar que as soluções para sensoriamento possam ser criadas a partir de dispositivos eletrônicos que ofereçam APIs com funções em alto nível de abstração, que facilitem esse trabalho. Nesse sentido, uma solução que tem se mostrado atraente é o uso do SoC ESP8266 em conjunto com o firmware open source NodeMCU, o qual suporta a programação com a linguagem de scripts Lua. Outro aspecto relevante é o uso de sistemas para computação em nuvem para armazenar e prover acesso às informações. Nesse trabalho é apresentado o desenvolvimento de um instrumento ubíquo para monitoramento e registro de pressão atmosférica, abordado etapas desde a coleta dos dados pelo servidor de borda até o armazenamento e visualização dos dados através de um servidor em nuvem.*

1. Introdução

A computação ubíqua (UbiComp) é um paradigma para concepção de infraestruturas computacionais que tem por meta a criação de sistemas inteligentes, de operação intuitiva e que se integram de forma natural no cotidiano dos usuários, em alguns casos sem que sejam percebidos estando disponíveis em qualquer lugar e a qualquer momento.

Modelos arquiteturais para UbiComp são amplamente discutidos na literatura. Abordagens baseadas em Internet of Things (IoT) têm sido difundidas nos últimos anos. Isso ocorre pela necessidade intrínseca dos sistemas ubíquos de interagir com sensores e atuadores. Tecnicamente, a IoT pode ser definida como uma tecnologia criada sob redes Web, baseada em protocolo TCP/IP, permitindo a comunicação entre dispositivos físicos e virtuais que possuem identidade única, atributos e interfaces inteligentes [Vermesan et al. 2009]. Ainda relacionado com a conectividade, são atribuídas características de generalização espaço-temporais à IoT, buscando a conectividade em qualquer lugar e a qualquer momento [Perera et al. 2013].

As redes de sensores são elementos fundamentais em IoT, pois é através delas que são gerados os pontos de acesso para os elementos monitorados ou controlados. Muitas aplicações possuem seu funcionamento baseado em rotinas de coleta, processamento e tomada de decisões baseadas em informações de sensores. Como resposta às decisões,

podem ser gerados comandos para atuadores [Perera et al. 2013]. Os dispositivos responsáveis pela comunicação com sensores e atuadores e interface com redes TPC/IP são comumente denominados como servidores de borda.

A possibilidade de sensoriamento remoto e distribuído provido em IoT, permite a criação de dispositivos inovadores para instrumentação, fazendo com que ferramentas de análise, armazenamento e visualização já consolidadas possam ser aplicadas. Nesse cenário alguns autores utilizam o termo Instrumentação Ubíqua, para caracterizar essa classe de dispositivos para medição de monitoramento que aplicam conceitos de IoT [Ferreira et al. 2013].

Neste trabalho são apresentadas as etapas de desenvolvimento de dispositivo para monitoramento de pressão atmosférica, que aplica os conceitos de Instrumentação Ubíqua. O trabalho consistiu no desenvolvimento de um servidor de borda e de um servidor para armazenamento dos dados em uma plataforma para computação em nuvem.

O texto foi dividido em quatro capítulos além deste de introdução. No Capítulo dois são apresentados os componentes e ferramentas de desenvolvimento utilizadas. A modelagem da arquitetura e suas especificações funcionais são apresentadas no capítulo 3. No capítulo 4 são apresentados os testes realizados e os resultados obtidos. No capítulo 5 tem-se as considerações finais do trabalho.

2. Materiais e métodos

A seguir estão descritas as ferramentas utilizadas para criação dos servidores de borda e em nuvem abordados no trabalho.

2.1. Módulo ESP8266

O módulo WiFi ESP8266 é um SoC com protocolo TCP/IP integrado e acesso a redes WiFi. O core de processamento é um processador com arquitetura ARM de 80 MHz. Foi utilizado no trabalho o firmware open source NodeMCU[1], o qual disponibiliza uma API com funções acessíveis por scripts em linguagem Lua. Os scripts são gravados na memória flash do ESP8266. Na figura 1 pode ser vista uma imagem do kit de desenvolvimento utilizado no trabalho e sua respectiva disposição de pinos.

Além da facilidade de utilizar na programação uma linguagem de scripts de alto nível como Lua, pode citar-se ainda como vantagem da utilização do firmware NodeMCU a possibilidade de parametrização remota do servidor de borda, visto que isso pode ser feito através da substituição dos arquivos de scripts, uma vez implementado o protocolo de rede adequado.

[1] https://github.com/nodemcu/nodemcu-firmware

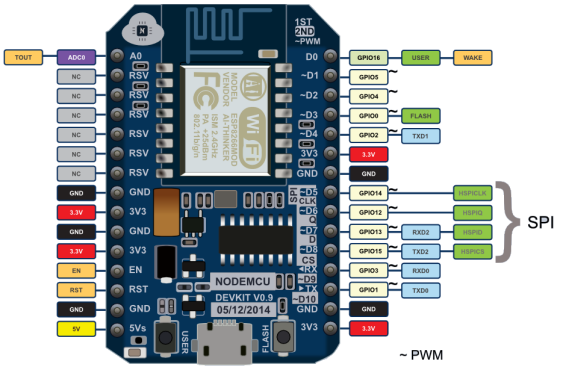

Figure 1. kit de desenvolvimento NodeMCU.

2.2. Sensor de pressão atmosférica MPL3115

Através da API para comunicação I2C do NodeMCU foi desenvolvido uma driver para o sensor de pressão atmosférica MPL3115, fabricado pela empresa Freescale. O sensor foi escolhido em função do interesse do grupo de pesquisa G3PD por aplicações que envolvem o monitoramento de condições climáticas.

A medição de pressão é feita pelo sensor com um conversor AD de 24 bis, sendo que a taxa de amostragem máxima é de 2 Hz. Além da medição de pressão atmosférica o sensor pode retornar também dados de altitude em relação ao nível do mar, neste caso a resolução é de 10 cm. O chip conta ainda com um sensor de temperatura de 12 bits que pode operar na faixa de -40 ºC a 85 ºC. Na figura 2 pode ser vista uma imagem do encapsulamento do sensor, o qual tem as dimensões de 3x5 mm.

Figure 2. Sensor de pressão atmosférica MLP3115.

2.3. Linguagens e ferramentas de programação utilizadas

O firmware NodeMCU disponibiliza uma API baseada na linguagem Lua. Esta foi desenvolvida e é mantida por um grupo de pesquisa da PUC-Rio. Lua combina programação procedural com poderosas construções para descrição de dados, baseadas em tabelas associativas e semântica extensível. É tipada dinamicamente, interpretada a partir de bytecodes, e tem gerenciamento automático de memória com coleta de lixo [2].

Na figura 3 pode ser visto um trecho de código extraído da aplicação desenvolvida, no qual é feita a passagem dos dados de um buffer de memória RAM para um arquivo na memória flash do ESP8266.

A gravação dos arquivos de scripts no kit NodeMCU e o monitoramento de mensagens enviadas via porta USB foram feitos utilizando os softwares NodeMCU Studio 2015 e ESPLorer.

[2]http://www.lua.org/portugues.html

```lua
-- Adiciona amostras em buffer temporário para o arquivo de log
try_file_open(log_file, "a+")
for i = 1, table.getn(amostras_temp_bf) do
    if(try_file_writeline(amostras_temp_bf[i]) == nil)then
        print("Erro ao escrever no arquivo de log de amostras.")
        print("Informações do sistema de arquivos: ", file.fsinfo())
        break
    else
        file.flush()
    end
end
try_file_close()
amostras_temp_bf = {}
```

Figure 3. Exemplo de código na linguagem Lua.

A comunicação com servidor remoto foi feita utilizando o protocolo XML-RPC. Este é protocolo de chamada de procedimento remoto que utiliza XML para codificar suas chamadas e HTTP como um mecanismo de transporte. Na figura 4 pode ser visto um exemplo de chamada HTTP para a função "put_log" para o servidor XML-RPC.

```
POST /RPC2 HTTP/1.0
User-Agent: XMLRPC-NodeMCU/1.0 (ESP8266)
Host: 192.168.0.18:8000
Content-Type: text/xml
Content-length: 289

<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
<methodName>put_log</methodName>
<params>
<param><value><string>ESP8266_1</string></value></param>
<param><value><string>37|101295\n42|101291\n47|101290\n
52|101291\n57|101291\n62|101290
</string></value></param>
</params>
</methodCall>
```

Figure 4. Exemplo de envio de chamada de procedimento remoto em XML-RPC.

O servidor remoto para receber e armazenar os dados coletados do sensor foi criado utilizando linguagem Python. A ferramenta para visualização dos dados armazenados também foi construída em Python, utilizando o ambiente interativo IPython. Nesse ambiente estão disponíveis poderosas ferramentas para plotagem de gráficos.

2.4. Google Cloud Plataform

O Google Cloud Plataform é um conjunto de soluções para computação em nuvem da empresa Google. As ferramentas disponibilizadas na plataforma estão divididas nos seguintes grupos:

- **Computação**: conjunto de ferramentas para processamento de informações, inclui Compute Engine que permite criar instâncias de computadores virtuais e aplicativos que acessam diretamente APIs da plataforma. Algumas imagens de sistema operacionais pré-formatados estão disponíveis, como Debin, Ubuntu e

Windows Server. Neste trabalho foi utilizada uma instância da distribuição Debian do Linux. O acesso para configuração da máquina virtual é feito através de um interface SSH integrada na plataforma. A mesma disponibiliza uma série de ferramentas para monitoramento de desempenho, na figura 5 pode ser visto o gráfico com o histórico de uso da CPU da máquina virtual utilizada no trabalho.

Figure 5. Interface com histórico de processamento da instância da máquina virtual utilizada no trabalho.

- **Armazenamento**: nessa categoria são agrupadas ferramentas para armazenamento de dados, que podem ser: bancos de dados relacionais (SQL), bancos de dados não relacionais (DASTORE e BIGTABLE), arquivos de uso geral (STOREGE). O acesso a estas bases de dados é feito através de bibliotecas disponíveis para Java, Python e PHP.
- **Desenvolvimento e monitoramento**: nesse grupo são disponibilizadas ferramentas para monitorar trafego de dados, acesso à aplicativos, processamento em instâncias de máquinas virtuais, conforme figura 5.

Para área de IoT o Google sugere o fluxo de processamento de informações mostrado na figura 6. Cabe ressaltar que, devido ao fato do Google Cloud ser uma ferramente genérica para processamento e armazenamento de informações em nuvem, não há uma solução pronta na plataforma para publicação e visualização de dados de sensores, como existe em outras plataformas especificas para IoT. Porém combinando as ferramentas para aplicativos, computação e armazenamento é possível criar soluções que atendem o fluxo de coleta, armazenamento e visualização de dados de sensores[Google].

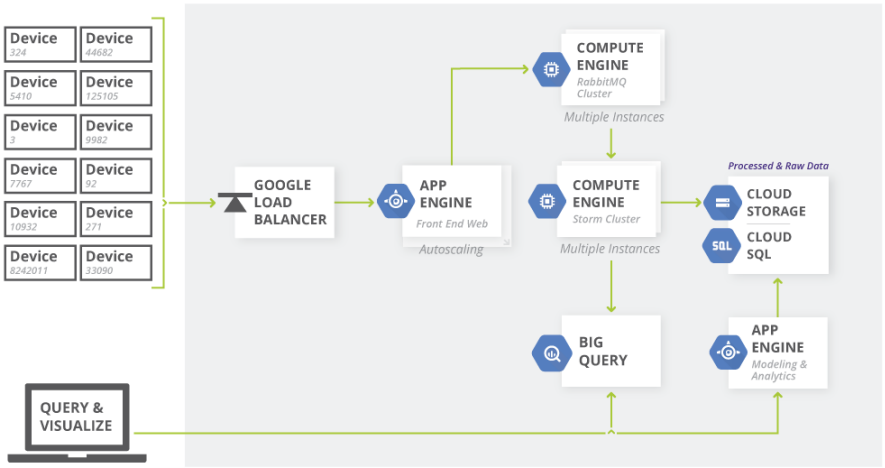

Figure 6. Modelo arquitetural sugerido pelo Google para IoT, fonte: [Google].

3. Modelagem da arquitetura

A modelagem da arquitetura foi dividida em duas partes: no servidor de borda e no servidor em nuvem. Ambos descritos a seguir.

3.1. Servidor de borda

A arquitetura do servidor de borda desenvolvido neste trabalho foi feita utilizando diagramas de máquina de estados em linguagem UML. Foram caracterizadas duas máquinas de estados principais, mostradas nas figuras 7 e 8. Como subsídio para criação das máquinas de estados e posterior teste do software embarcado foi criada a lista de requisitos e especificações mostradas a seguir.

- **Conexão por rede WiFi**: o servidor de borda deve ser capaz de iniciar uma conexão com o roteador WiFi a partir do nome e da senha da rede. Em caso de desconexão, a tentativa de reconexão deve ser feita sem a necessidade de reset e de forma automática.
- **Comunicação com servidor XML-RPC**: uma vez estabelecida a conexão com a rede WiFi o servidor de borda deve estabelecer comunicação com um servidor remoto utilizando o padrão XML-RPC sobre o protocolo HTTP. O endereço IP e a porta do servidor serão armazenados no servidor de borda em um arquivo de configuração. O servidor XML-RPC disponibilizará duas funções, uma para envio das amostras coletadas e outra para solicitar sincronização de data e hora.
- **Log de amostras em memória não volátil**: caso a conexão com o servidor remoto seja interrompida, os valores das amostras lidas do sensor devem ser armazenadas em um arquivo na memória flash do ESP8266. Quando a conexão for reestabelecida as amostras devem ser envidas para o servidor e removidas do arquivo.
- **Sincronismo de data e hora**: em um intervalo de 5 min deve ser solicitada ao servidor as informações de data e hora atuais. Esta informação deverá ser adicionada no conjunto de dados retornados para o servidor, para cada amostra deve ser registrado o tempo ocorrido após o último sincronismo.

- **Amostragem do sensor**: deve ser criado um driver para comunicação com sensor I2C capaz de realizar a leitura da pressão atmosférica proveniente do sensor MLP3115. A taxa de amostragem deverá ser de 5 s. Caso o sensor apresente falha de comunicação deverá ser executada automaticamente a rotina de inicialização do mesmo.
- **Interface para monitoramento e testes**: durante a execução dos scripts no servidor de borada devem ser enviados pela porta USB do kit de desenvolvimento mensagens em formato de texto informando eventuais erros e o estado atual da aplicação.

O diagrama de estados mostrado na figura 7 representa o ciclo de processamento que será executado pelo firmware NodeMCU. Pode notar-se que para que eventos de timers ou de comunicação sejam processados e necessário que o scrip em execução se finalizado e o controle de execução seja devolvido ao firmware, caracterizado no diagrama como o estado "NodeMCU".

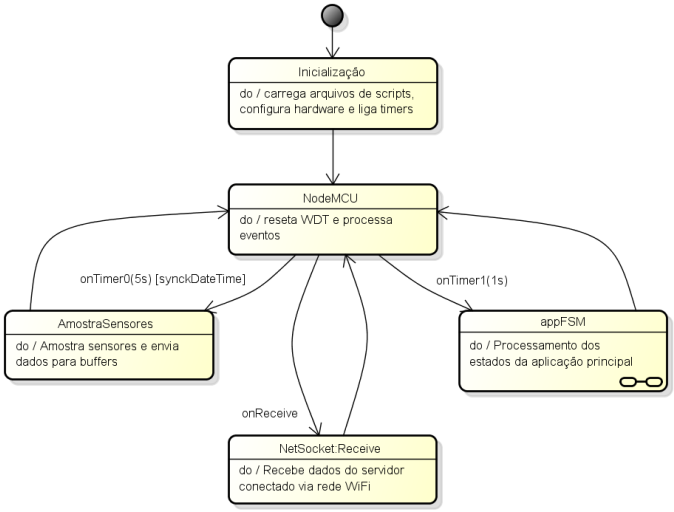

Figure 7. Diagrama de estados do contexto de execução principal da aplicação.

Os estados mostrados no diagrama da figura 8 foram elaborados para atender as especificações listadas anteriormente. A execução da máquina de estados foi vinculado a um timer com periodicidade de 1 s. Em função do modelo de processamento do firmware NodeMCU, mostrado na figura 7, os estados não podem conter instruções de espera. Desta forma, o processamento de respostas assíncronas foi feito em estados individuais, sendo eles: "CONECTA_WIFI", "CONF_ENVIO_AMOSTRAS" e "AGUARDA_DATA_SERVIDOR".

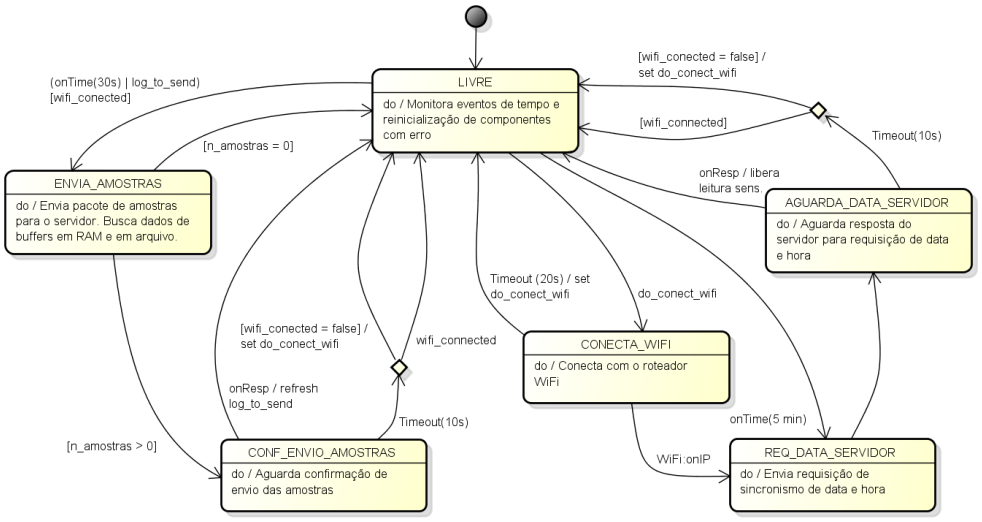

Figure 8. Diagrama de estados da sequência de ações da aplicação.

3.2. Servidor em nuvem para armazenamento e acesso

Para armazenar e permitir a visualização dos dados coletados pelo servidor de borda foi criado um servidor utilizando a ferramenta para computação em nuvem Google Cloud Plataform.

Foi criado nesse ambiente uma máquina virtual com o sistema operacional Linux da distribuição Debian. Uma vez criada a máquina virtual, a mesma foi configurada utilizando a interface SSH integrada na plataforma Google. A configuração consistiu na criação de um servidor XML-RPC utilizando a linguagem Python. O acesso a esse servidor pelas aplicações é feito através de um endereço de IP fixo na estrutura do Google.

Neste servidor foram disponibilizadas as seguintes funções:

- **“put_log”**: permite que sejam inseridos dados amostrados pelo servidor de borda em um buffer. Cada amostra contém duas informações: data de coleta e valor.
- **“get _log”**: permite que sejam lidos os dados armazenados no buffer de amostras.
- **“clear_log”**: apaga os dados do buffer de amostras.
- **“get _datetime”**: retorna os valores de data e hora atuais do servidor no formato “dd/mm/aa H:M:S”.

A interface para visualização de dados coletados foi criada no ambiente IPython, o código criado pode ser visto na figura 9.

```
import plotly
import xmlrpc.client
from datetime import datetime, date, time, timedelta

sensors_log_server = xmlrpc.client.ServerProxy(HOST_IP+':'+HOST_PORT)
client_id = "DATA_VIWER_1"

values = sensors_log_server.get_log(client_id)
print('Hora servidor: ' + sensors_log_server.get_datetime(client_id))
print('Count amostras: ' + str(len(values)))
x_values = []
y_values = []
ts_ans1 = None

for n in range(0, len(values)):
    ts_atual = datetime.strptime(values[n][0], "%d/%m/%y %H:%M:%S")
    if (ts_ans1 == None) or (ts_atual > ts_ans1):
        x_values.append(values[n][0])
        y_values.append(int(values[n][1]))
        ts_ans1 = ts_atual

plotly.offline.init_notebook_mode() # run at the start of every notebook
plotly.offline.iplot({
        "data": [{
        "x":x_values,
        "y": y_values
    }],
    "layout": {"title": "Pressão Atmosférica (Pa)"}
})
```

Figure 9. Código para visualização gráfica das amostras coletadas.

4. Testes e resultados obtidos

Os testes realizados sobre a aplicação desenvolvida foram divididos em dois tipos: caixa branca e caixa preta; a título de ilustração são listados a seguir alguns testes executados e os resultados obtidos. Cabe ressaltar que, para validar toda a aplicação, outros testes devem ser escutados.

4.1. Testes de caixa branca

Considerando a arquitetura interna da aplicação e os estados que a mesma deve assumir, foram executados os seguintes testes de caixa branca:

1. Verificação da conexão com a rede WiFi.
2. Análise dos logs gerados via porta USB para verificar se as transições de estados estão de acordo com o diagrama da figura 8.

Na figura 10 podem ser vistos os registros gerados após a inicialização da aplicação. Verifica-se que os resultados estão de acordo com a máquina de estados proposta na figura 8.

```
Estado = EST_LIVRE
Inicializando Sensor MPL3115
Sensor conectado
Estado = EST_CONECTA_WIFI
Aguardando IP...
Estado = EST_CONECTA_WIFI
Conectado: XT1068 2786
IP:192.168.43.23
Estado = EST_REQ_DATA_SERVIDOR
Estado = EST_AGUARDA_DATA_SERVIDOR
RPC(get_datetime), Status: OK  Resposta: 26/11/15 13:10:20
Estado = EST_LIVRE...
Lendo MPL3115... OK
Estado = EST_ENVIA_AMOSTRAS
Estado = EST_CONF_ENVIO_AMOSTRAS...
Aguardando resposta servidor... 1
Estado = EST_CONF_ENVIO_AMOSTRAS...
RPC(put_log), Status: OK  Resposta: OK
```

Figure 10. Registro de estados da aplicação após a inicialização.

4.2. Testes de caixa preta

Considerando apenas especificações de alto nível da aplicação, relacionadas com o envio de dados para o servidor remoto e log de amostras durante a perda de comunicação, foram executados os testes listados a seguir.

1. Verificação do envio das amostras de pressão atmosférica para o servidor, considerando a periodicidade de 5 s e faixa de valores coerentes com a pressão atmosférica do local de realização dos testes.
2. Após estabelecer comunicação inicial com o servidor e realizado o sincronismo de data e hora, verificar se o servidor de borda armazena localmente os dados lidos do sensor e os envia para o servidor remoto após a reativação da conexão.

No gráfico da figura 11 podem ser vistos os valores de pressão atmosférica medidos pelo servidor de borda e enviados para o servidor remoto. O valor medido corresponde em média a aproximadamente 1 atm, pressão esta que é coerente com o local de realização do teste. Para simular a perda de comunicação com o servidor, no período destacado na figura, o servidor remoto foi desativado. Após a reativação do servidor, as amostras armazenadas localmente no servidor de borda foram enviadas para o servidor remoto.

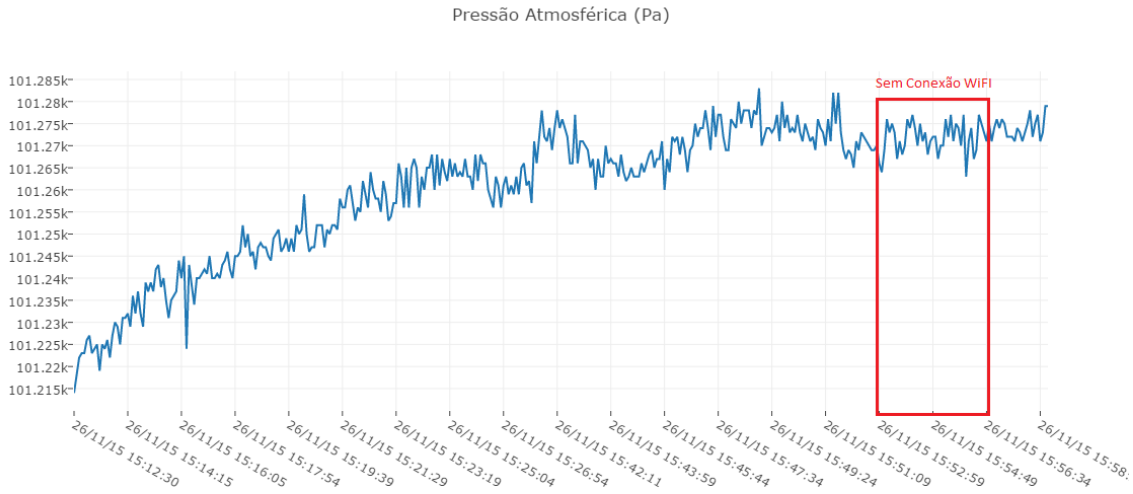

Figure 11. Amostras de pressão atmosférica enviados para o servidor remoto.

4.3. Problemas identificados

Durante o desenvolvimento da aplicação foram encontrados alguns problemas na API do NodeMCU, como erro na rotina "tmr.time()" e erros ao acessar a API de arquivos. A função "tmr.time()" deveria retornar o tempo em segundos transcorrido após o reset do sistema, porém, em alguns casos o valor decrescia para duas leituras consecutivas. Este problema está relatado na página de desenvolvimento do firmware[3].

Com relação a API de arquivos foi detectado que, de maneira aleatória, o sistema é resetado após uma tentativa de leitura ou gravação de um arquivo. Esse problema também foi relatado por outros usuários da API.

5. Considerações Finais

Através deste trabalho foi possível avaliar o fluxo de desenvolvimento de instrumento ubíquo baseado no conjunto de ferramentas ESP8266, NodeMCU, Google Cloud, fazendo uso das linguagens Lua e Python.

Relacionado ao conjunto ESP8266, NodeMCU e Lua aplicado no servidor de borda, mesmo com alguns problemas encontrados na API, a solução baseada nesse conjunto de ferramentas se mostrou bastante eficiente em termos de velocidade de desenvolvimento. A API baseada em linguagem Lua possui funções em alto nível de abstração que possibilitam uma rápida configuração e uso do hardware.

Para o servidor em nuvem baseado no Google Cloud não foram encontrados problemas durante o desenvolvimento e foram identificadas algumas possibilidades para expansão do trabalho, como o uso do banco de dado NoSQL e da API para a criação de aplicativos web possibilitando a construção um interface gráfica para controle e visualização das informações, visto que neste trabalho a visualização dos dados coletados foi feita no ambiente de programação IPython.

[3]https://github.com/nodemcu/nodemcu-firmware/issues/690

References

Ferreira, D., Koehler, C., Karapanos, E., and Kostakos, V. (2013). Ubiquitous mobile instrumentation. *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication - UbiComp '13 Adjunct*, pages 1409–1412.

Google. Architecture: Real Time Stream Processing - Internet of Things - Architectures - Google Cloud Platform.

Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2013). Context Aware Computing for The Internet of Things : A Survey. *IEEE COMMUNICATIONS SURVEYS & TUTORIALS*, X(X):1–41.

Vermesan, O., Friess, P., Guillemin, P., Gusmeroli, S., Sundmaeker, H., Bassi, A., Jubert, I. S., Mazura, M., Harrison, M., Eisenhauer, M., Doody, P., Peter, F., Patrick, G., Sergio, G., Harald, Sundmaeker Alessandro, B., Ignacio Soler, J., Margaretha, M., Mark, H., Markus, E., and Pat, D. (2009). Internet of Things Strategic Research Roadmap.