APRIMORAMENTO DO MÉTODO DE ATUALIZAÇÃO DE FIRMWARE DE NÓ SENSOR SEM FIO EM PONTOS REMOTOS ATRAVÉS DA INTERFACE RÁDIO

Paulo Baraldi Mausbach
Faculdade de Engenharia da Computação
CEATEC - Centro de Ciências Exatas, Ambientais e de Tecnologias
pbmausbach@hotmail.com

Omar C Branquinho
Sistema de Telecomunicações – Gestão de Redes e Serviços
Planejamento Integrado e Gestão de Sistemas de Infraestrutura Urbana
branquinho@puc-campinas.edu.br

Resumo: *A atualização remota de firmware é uma das funcionalidades mais importantes em uma rede sensor sem fio (RSSF) dado que muitos nós sensores encontram-se em locais remotos e de difícil acesso. Atualmente a atualização de firmware de muitos nós sensores ainda é realizada via conexão local. Esse artigo apresenta uma proposta de implantação da funcionalidade de atualização remota, denominada firmware over the air (FOTA), sob uma RSSF. Como benefícios, essa pesquisa desenvolve uma arquitetura de implantação que permita a programação remota de um nó sensor, um protocolo para permitir a comunicação entre os dispositivos dessa arquitetura e todas as aplicações necessárias para coloca-la em prática.*

Palavras-chave: *IOT, RSSF, FOTA.*

Área do Conhecimento: *Engenharias – Engenharia de computação e telecomunicação – redes de sensoriamento.*

1. INTRODUÇÃO

RSSFs tornaram-se importantes para o desenvolvimento de tecnologias inclusas na ideologia de Internet das Coisas (IOT). Esses sensores contemplam algoritmos gravados em suas memórias chamados *firmwares*, os quais permitem a realização de diversas tarefas pelo nó sensor.

Atualmente para realizar a gravação de um novo *firmware* em sensores de diversas RSSFs é necessário ir ao local onde o mesmo está instalado e, através de uma conexão física por meio de uma porta serial do sensor com um computador, efetuar o chamado *upload* do novo *firmware*. Entretanto, esse método de atualização dos nós sensores dificulta a atualização do *firmware* de dispositivos em áreas remotas ou de difícil acesso, tornando esse procedimento custoso e árduo.

Como benefício, essa pesquisa desenvolve uma arquitetura de implantação que disponibilize a atualização remota dos *firmwares* de uma RSSF; a criação de um protocolo que permita a execução dos comandos necessários para a estratégia proposta e os testes para consolidar o funcionamento do que foi desenvolvido.

2. REVISÃO DA LITERATURA E ESTADO DA ARTE

O *upload* de *firmware* sempre foi um tópico muito importante em relação à RSSFs [1]. Atualmente, quando o assunto se trata de RSSF e atualização dos *firmwares* dos nós sensores, encontra-se uma grande limitação presente na maioria das RSSFs, a necessidade de realizar localmente esse *upload* de um novo programa. Essa limitação prejudica a escalabilidade do uso de RSSFs, pois dificulta todo o processo de manutenção e inserção de novas funcionalidades nos nós sensores, o que reflete negativamente na qualidade da RSSF.

A estratégia FOTA é uma potencial solução para esse tipo de limitação, pois é amplamente utilizada em diversos setores, como por exemplo, o setor automobilístico [2] e o de dispositivos *mobile* [3]. Essa estratégia de *upload* de *firmware* permite a realização de diversas operações remotamente. No caso das RSSFs, não haveria mais a necessidade do deslocamento até o local de cada nó sensor para realizar a atualização de seu programa, tornando todo o processo mais ágil e econômico.

3. PROPOSTA

3.1. Arquitetura de implantação inicialmente proposta

A arquitetura proposta inicialmente para o desenvolvimento dessa pesquisa era composta por um computador conectado, via cabo USB, a um microcontrolador que funcionaria como a base da RSSF. Essa base possuiria um módulo de rádio para se comunicar com os nós sensores de toda a rede por meio da plataforma Radiuno[4]. Cada nó sensor seria com-

posto apenas por um microcontrolador com módulo de rádio. Nessa arquitetura, o próprio nó sensor, após receber todo o programa e o comando para a gravação de um novo código, seria realizado um procedimento de *reset* automático, para que durante a execução do código do *bootloader* fosse possível gravar o código recebido no próprio nó sensor, resultando em uma autoprogramação de um novo *firmware*.

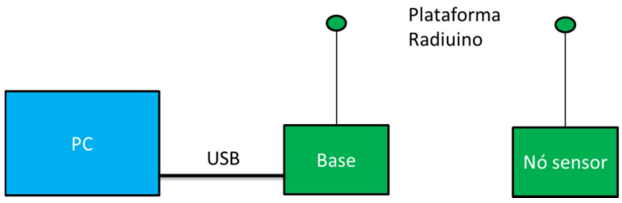

Figura 1. Arquitetura inicialmente proposta

Entretanto foram encontradas algumas limitações de *hardware* que impossibilitariam ou tornariam pouco viável essa arquitetura, exigindo então algumas mudanças. A primeira está relacionada a uma limitação da memória dos processadores presentes nos nós sensores. O microcontrolador ATmega 328 [5], que é o microcontrolador presente nos nós sensores, possui uma memória total de 32KB, na qual está gravado o código do *bootloader* e o *firmware* que é executado. Desta forma não há como armazenar um *firmware* que ocupe mais do que a memória restante no nó sensor. Como normalmente o *firmware* da plataforma Radiuino, executado nos nó sensores, já ocupa mais da metade da memória disponível, torna-se impossível o recebimento de um novo *firmware* sem sobrescrever o antigo ou o *bootloader*. Dessa forma é necessário o uso de uma memória externa, para possibilitar o armazenamento de todo o novo *firmware* a ser recebido, sem que haja a sobrescrita dos outros códigos já presentes na memória que causaria um mau funcionamento do nó sensor.

Para ser realizada a autoprogramação do nó sensor, seria necessário reinicia-lo, para que durante a execução do *bootloader,* o microcontrolador entrasse em modo de programação e realizasse sua autoprogramação. Contudo, visto a necessidade do uso de uma memória externa, seria necessária a customização do *bootloader* para que o mesmo pudesse acessa-la possibilitando a leitura do *firmware* a ser gravado em sua memória. Entretando, devido à limitação do tamanho reservado para o código do *bootloader*, as alterações nesse código poderiam estourar o limite de espaço e também reduzir o potencial futuro de novas funcionalidades a serem implantadas. Dessa forma optou-se por utilizar um segundo microcontrolador para realizar a programação do novo *firmware* no nó sensor

3.2. Nova arquitetura de implantação proposta

Devido às alterações especificadas anteriormente a nova arquitetura proposta para o desenvolvimento da solução e a realização de testes é composta por um computador conectado, por meio de um cabo USB, a um microcontrolador que funciona como a base da RSSF. Essa base possui um módulo de rádio para realizar a comunicação com os nós sensores por meio de radiofrequência com o uso da plataforma Radiuino. Cada nó sensor é composto por dois microcontroladores, um sendo efetivamente o nó sensor, conectado a diversos medidores de grandezas e contendo um módulo de rádio para permitir a comunicação com a base. O outro com um shield SD para o armazenamento de *firmwares* recebidos e funcionando como programador do nó sensor.

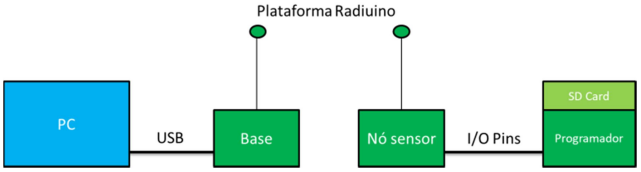

Figura 2. Arquitetura proposta

O funcionamento dessa arquitetura se daria pela troca de mensagens entre programador e computador, tendo o nó sensor e a base funcionando apenas como retransmissores. O computador funciona como *mestre* de todas as operações e o programador como *escravo*, apenas executando os comandos e respondendo ao computador sobre a ocorrência ou não de problemas durante a execução. Em geral o computador envia mensagens via USB à base, a qual é retransmitida ao nó sensor endereçado nela. O nó sensor, ao receber essas mensagens, as repassa para o programador o qual efetua o comando referente a cada mensagem e em seguida envia para o computador uma resposta pelo caminho inverso. As mensagens trocadas são referentes ao protocolo desenvolvido nessa pesquisa.

3.3. Estratégia para o desenvolvimento da arquitetura proposta

Considerando a arquitetura proposta, o projeto de desenvolvimento foi divido em partes - preparadas separadamente e depois unidas para a realização de testes em conjunto - com o objetivo de verificar o correto funcionamento simultâneo dos componentes. A seguir estão representadas de maneira esquemática cada parte da arquitetura em suas respectivas

camadas de aplicação e enlace, quando relevantes para o projeto.

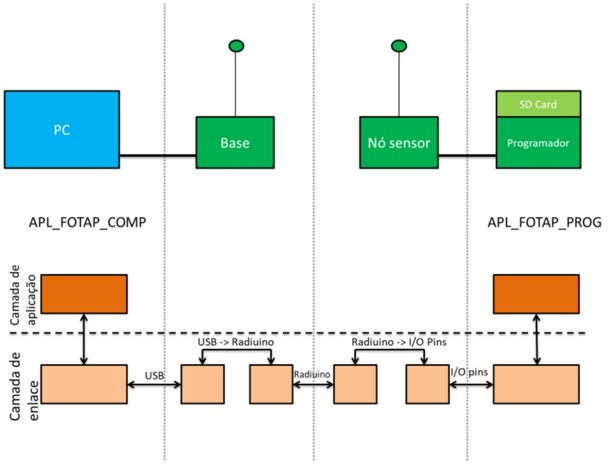

Figura 3. Particionamento da arquitetura proposta

Nessa divisão há duas partes principais nas quais o desenvolvimento está focado. A primeira é chamada APL_FOTAP_COMP, responsável pelo funcionamento do protocolo desenvolvido e todo o controle sob o fluxo de execução do envio de mensagens. Em sua camada de aplicação encontra-se toda a leitura do arquivo a ser enviado para o programador, preparo das mensagens, tratamento de resposta e controle de falhas durante a troca de mensagens. A camada de enlace é realizada pelo próprio Sistema Operacional (SO) do computador. A segunda parte, chamada de APL_FOTAP_PROG, é responsável pela aplicação da estratégia do lado do nó sensor. Em sua camada de aplicação encontram-se todos os procedimentos e funções para efetuar o tratamento das mensagens, execução dos comandos, gerenciamento do uso da memória externa e montagem das respostas. Sua camada de enlace contempla todas as funcionalidades necessárias para recebimento de mensagens e envio de respostas. As camadas de aplicação da base da RSSF e do nó sensor não estão presentes na Figura 3, pois elas são totalmente transparentes para o processo de comunicação entre computador e programador, mantendo seus códigos inalterados. Suas camadas de enlace foram representadas apenas para demonstrar o caminho realizado pelas mensagens trocadas e quais as plataformas utilizadas para suas transmissões.

3.4. Detalhamento do protocolo FOTAP (Firmware Over The Air Protocol)

O protocolo FOTAP foi desenvolvido para realizar a estratégia de *upload* FOTA e teve como base o protocolo rdt 3.0 apresentado no Capitulo 3 do livro Redes de Computadores e a Internet[6]. Esse tipo de protocolo é chamado de *stop and wait*, pois para cada mensagem enviada, espera-se receber sua resposta para realizar o envio da próxima. Sendo assim, não é realizado nenhum tipo de paralelismo no envio e recebimento das mensagens.

O protocolo rdt 3.0 foi escolhido como exemplo base para a criação do protocolo FOTAP para essa pesquisa, pois é de fácil desenvolvimento e contempla controles de falhas recorrentes ao envio de mensagens entre dispositivos, como por exemplo, mecanismo de bit alternado para detecção de mensagens duplicadas e mecanismo de *timeout* para controle de perda de mensagens, tornando-o robusto o suficiente para a realização de testes. Esse tipo de protocolo, sem dúvidas, não é aquele que possui o melhor desempenho, mas é o suficiente para obter resultados que comprovem o funcionamento da arquitetura proposta.

O protocolo FOTAP teve de ser moldado sob as limitações de tamanho de mensagem existente na plataforma Radiuno, já que essa foi a plataforma escolhida para ser usada para a transmissão de mensagens. Dado esse fato ele foi modelado sob o limite de 36 bytes, referente ao tamanho disponível para envio de dados pela plataforma Radiuino. Contudo, com pequenas alterações nas aplicações desenvolvidas, é possível que elas se adequem a plataformas com outros tamanhos de mensagens.

3.4.1 *Mensagens e comandos*

Atualmente o protocolo FOTAP é constituído por seis diferentes comandos e suas respectivas respostas, sendo cada mensagem referente a um comando que pode ser efetuado pelo programador.

Tabela 1. Mensagens e respostas com seus respectivos parâmetros

Mensagem	Resposta
ApagarArq (nNó, nBytes, nomeArquivo)	OK / NOK(ERR)
CriarArq (nNó, nBytes, nomeArquivo)	OK / NOK(ERR)
EnviarPac (nNó, nBytes, pacote)	OK / NOK(ERR)
VerificaArq (nNó, cs, nBytes, nomeArquivo)	OK / NOK(ERR)
Programar (nNó, nBytes, nomeArquivo)	OK / NOK(ERR)
Sincronizar(nNó)	OK / NOK(ERR)

Os comandos executados pelo programador referente a cada mensagem são os seguites:

- **ApagarArq:** Apaga o arquivo com o nome enviado
- **CriarArq:** Cria um novo arquivo com o nome enviado
- **EnviarPac:** Envia um pacote de dados. Utilizado para o envio dos segmentos do novo código que será salvo no cartão SD para futuramente ser programado no nó sensor.
- **VerificarArq:** Realiza um *checksum* do arquivo presente no cartão SD e compara com o *checksum* enviado na mensagem, o qual é referente ao mesmo calculo de *checksum* realizado pelo programador sob o arquivo presente no computador.
- **Programar:** Realiza a programação de um novo código no nó sensor, sendo esse código referente ao arquivo com o nome enviado na mensagem.
- **Sincronizar:** Realiza o sincronismo do valor do bit alternado entre os dois dispositivos que estão se comunicando.

3.4.2 *Formatos de mensagens*

O protocolo FOTAP contempla, até o momento, quatro formatos diferentes de mensagens. É comum entre esses diferentes formatos o primeiro byte ser referente ao número do nó sensor a qual a mensagem é destinada e o segundo byte ser o chamado cabeçalho ou *header* da mensagem, o qual contém os dados referentes à operação que será realizada, e em casos específicos também é utilizado no envio de dados. Os bytes restantes são nomeados corpo ou *payload* da mensagem, são nesses bytes em que está contido o seu conteúdo. Como a plataforma Radiuino permite o uso de apenas 36 bytes para o envio de dados, o tamanho máximo para o corpo da mensagem é de 34 bytes.

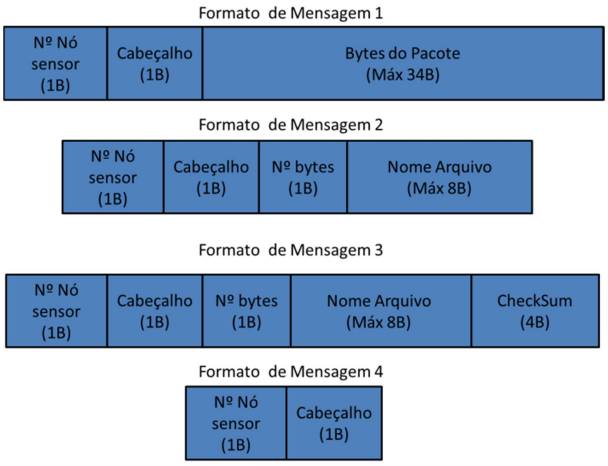

Figura 4. Formato das Mensagens

O formato de mensagem 1 é um formato especialmente moldado para as mensagens do comando "EnviarPac", pois visa tirar maior proveito de todos os bytes do corpo da mensagem, já que esse comando é executado diversas vezes durante a programação remota do nó sensor. O formato 2 é o mais comumente utilizado pelos outros comandos com exceção do "VerificaArq", o qual possui 4 bytes a mais em seu corpo para envio do *checksum* do arquivo dando origem ao formato 3, e o comando "Sincronizar" que utiliza o último formato. O formato 4 também é utilizado por todas as respostas a todos os comandos. Possui apenas dois bytes, pois não é necessário o uso dos bytes disponíveis para o corpo da mensagem. É um formato bem compacto, pois é utilizado com bastante frequência durante todo o procedimento, já que todo comando requer uma resposta.

Os diferentes comandos do protocolo possuem algumas divergências em relação à formatação do cabeçalho de suas respectivas mensagens e respostas. Essas formatações estão explicitadas nas tabelas 2 e 3 a seguir, sendo a primeira respectiva às mensagens e a segunda às respostas.

Tabela 2. Formatação das mensagens

Comando	Cabeçalho	Corpo	OP Code(CCC)
EnviarPac	VNNNNNN1	Formato 1	XXX
CriarArq	VXXXCCC0	Formato 2	000
ApagarArq	VXXXCCC0	Formato 2	001
VerificaArq	VXXXCCC0	Formato 3	010
Programar	VXXXCCC0	Formato 2	011
Sincronizar	VXXXCCC0	Formato 4	100

Na tabela 2 as posições dos cabeçalhos em que se encontram a letra **V** são referentes à posição em que o bit alternado é posicionado no cabeçalho das mensagens; as letras **X**s representam bits não usados do cabeçalho; as letras **C**s foram utilizadas para demonstrar três bits ocupados pelo OP Code do comando, os quais concatenados com o bit menos significativo do cabeçalho, resultam no OP Code completo. Os seis bits representados pelas letras **N**s no cabeçalho do comando "EnviarPac" são utilizados para envio de dados relacionados ao número de bytes armazenados no corpo desse tipo de mensagem. Essa informação foi inserida no cabeçalho apenas das mensagens desse comando especificamente, pelo fato de que ele seria o comando mais frequente durante a programação remota do nó sensor. Desta forma, busca-se minimizar o chamado *overhead* que seria gerado por esse tipo de mensagem.

Tabela 3. Formatação das respostas

Comando	Cabeçalho	Corpo	OP Code(CCC)
EnviarPac	VXXXXXX1	Formato 4	XXX
CriarArq	VEEECCC0	Formato 4	000
ApagarArq	VEEECCC0	Formato 4	001
VerificaArq	VEEECCC0	Formato 4	010
Programar	VEEECCC0	Formato 4	011
Sincronizar	VEEECCC0	Formato 4	100
Iniciar cartão SD	VEEECCC0	Formato 4	111

Na tabela 3 as posições dos cabeçalhos em que se encontram a letra **V** são referentes à posição em que o bit alternado é posicionado no cabeçalho; as letras **X**s representam bits não usados do cabeçalho; as letras **E**s foram usadas para demonstrar os três bits ocupados pelo código de erro contido na resposta; e as letras **C**s concatenadas ao bit menos significativo do cabeçalho formam o OP Code completo referente a qual comando gerou a respectiva resposta.

É importante observar que não existe mensagem para o comando "Iniciar cartão SD" mostrado na tabela 3, pois esse é um comando que ocorre durante a inicialização (*boot*) do programador do nó sensor, sem a necessidade de ele ter recebido previamente uma mensagem indicando que deva ser realizada essa operação. Contudo é de extrema importância que o computador que controla todo o processo tenha conhecimento se o cartão SD foi inicializado ou não com sucesso, tornando crucial o envio de uma resposta avisando se ocorreu ou não algum tipo de falha durante esse procedimento. Desta forma para que seja possível diferenciar uma resposta referente a essa operação interna do programador das outras relacionadas aos comandos que podem ser requisitados por meio do protocolo, o OP Code 1110 foi reservado como código de operação do comando de inicialização do cartão SD, não podendo ser utilizado futuramente caso sejam inseridos novos comandos ao protocolo FOTAP.

Na resposta de todos os comandos é enviado um código de erro identificando se não ocorreu nenhum erro ou, caso tenha ocorrido, qual foi ele. Para todas as operações o código de valor 000 é indicativo de que a operação foi bem sucedida - qualquer valor diferente indica a ocorrência de algum problema durante a execução do comando.

Com exceção do comando "Sincronizar" o qual não retorna nenhum tipo de erro, pois identifica se esse procedimento foi efetuado corretamente, através da comparação do bit alternado enviado nas mensagens. Todas as outras operações possuem diferentes tipos de erros que podem vir a acontecer, portanto estão representados na tabela 4 todos os códigos de erro e seus respectivos significados para cada comando.

Tabela 4. Códigos de erro

Comando	Código de erro	Significado
EnviarPac	001	Operação Invalida
EnviarPac	010	Não foi possível abrir o arquivo
EnviarPac	011	Não foram escritos todos os dados no arquivo
CriarArq	001	Arquivo já existente
CriarArq	010	Erro durante a abertura do arquivo
ApagarArq	001	Arquivo não existe
ApagarArq	010	Erro durante a remoção do arquivo
VerificaArq	001	Arquivo não existe
VerificaArq	010	Não foi possível abrir o arquivo
VerificaArq	011	Checksum divergente
Inicializar cartão SD	111	Erro na inicialização do cartão SD

3.4.3 *Fluxo de mensagens durante a programação remota de firmware*

Durante a programação remota do *firmware* do nó sensor, são trocadas diversas mensagens entre computador e programador. Sempre após o envio de uma mensagem, a próxima só é enviada após receber a resposta referente a ela, o que é característica de um protocolo *stop and wait*, conforme já mencionado. Caso a reposta recebida seja que tudo ocorreu como esperado, o fluxo principal continua normalmente.

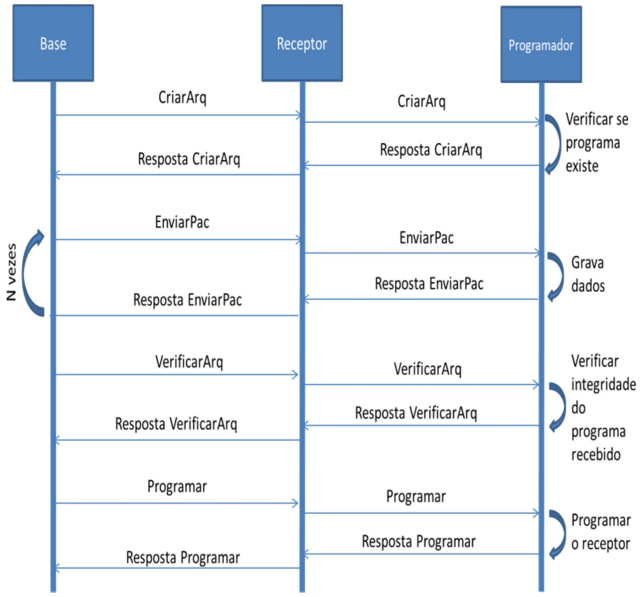

Figura 5. Fluxo de mensagens sem ocorrência de falhas

No caso de a resposta ser um aviso de ocorrência de um erro durante a execução do comando, o fluxo principal é abortado, já que é de extrema importância que tudo ocorra corretamente para que seja segura a gravação de um novo código no nó sensor. Isso porque qualquer falha mínima durante esse procedimento pode resultar na gravação de um firmware defeituoso, prejudicando o funcionamento do nó sensor.

Para esses casos há a funcionalidade “ApagarArq” que pode ser solicitada pelo computador ao programador para que ele efetue a remoção de um arquivo da memória externa, permitindo que sejam excluídos arquivos que potencialmente estejam com problemas devido a falhas durante o processo de gravação remota. A seguir está um diagrama mostrando o fluxo de mensagens trocados para a realização dessa operação.

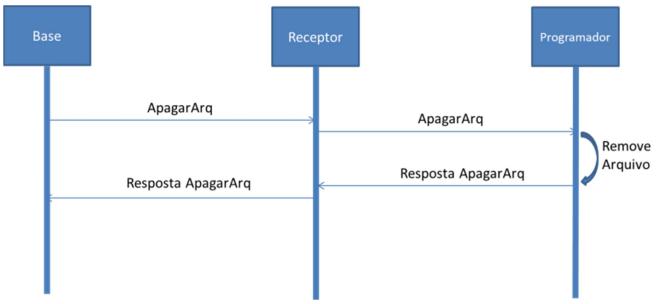

Figura 6. Fluxo de mensagens - comando “ApagarArq”

Ainda há casos em que as mensagens ou respostas podem ser perdidas durante o envio de um dispositivo para outro. É importante que esse tipo de problema não cause a finalização do procedimento, pois como é algo que acontece com certa frequência, pode prejudicar bastante o desempenho e o funcionamento de toda a arquitetura proposta. Para isso foram inseridos os mecanismos de *timeout* e bit alternado no protocolo FOTAP para realizar o controle de perda de mensagens e detecção de mensagens duplicadas.

3.4.4 *Mecanismo de timeout*

O mecanismo de *timeout* é responsável por detectar se uma mensagem enviada ou sua respectiva resposta foram perdidas. Esse mecanismo é de extrema importância, pois sem ele seria possível em algum momento, uma mensagem ou resposta ser perdida e o computador então ficar em um espera infinita para receber uma resposta a essa mensagem. Isso ocorreria, pois como o programador nunca teria recebido nenhuma mensagem, devido a sua perda, ele se manteria em espera pelo recebimento de uma mensagem contendo o comando que ele deve executar, e consequentemente nunca enviaria uma resposta ao computador. Como consequência disso, haveria um bloqueio na comunicação entre os dois dispositivos e o funcionamento correto da arquitetura proposta ficaria prejudicado.

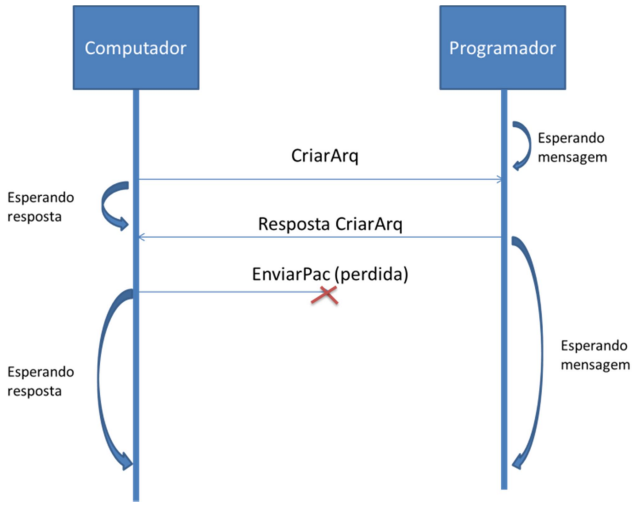

Figura 7. Simulação de problema causado por perda de mensagens

O conceito por trás desse mecanismo é basicamente verificar, sempre após o envio de uma mensagem, se durante um intervalo de tempo não foi recebida a resposta referente à mensagem enviada. Caso a resposta não chegue dentro desse intervalo, ocorre o evento chamado de *timeout*, ou seja, acabou o tempo de espera por uma resposta, pressupondo-se então que a mensagem ou sua resposta foi perdida. A solução para isso seria tentar enviar novamente a mensagem, tentando então realizar a operação que supostamente ainda não foi executada, e caso, dessa vez, fosse recebida uma resposta referente à mensagem enviada, continuar-se-ia o fluxo de execução para efetuar a programação remota do nó sensor.

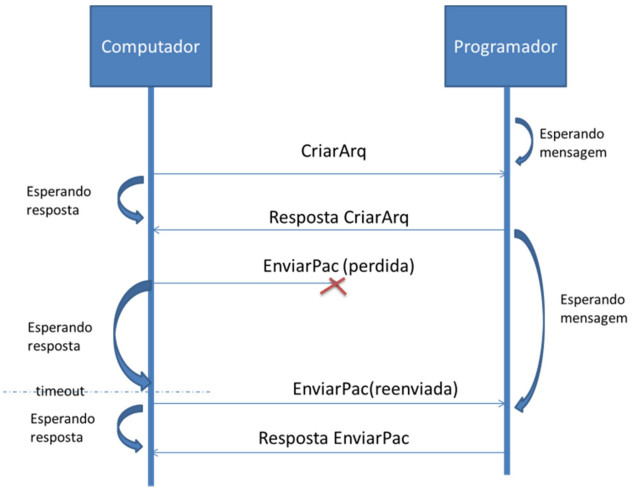

Figura 8. Solução gerada pelo timeout

Entretanto o reenvio de mensagens supre um problema, mas acaba criando outro, a execução duplicada de um mesmo comando.

3.4.5 *Mecanismo de bit alternado*

Caso não aconteça a perda da mensagem, mas sim de sua resposta, se a mensagem for reenviada, o programador estará realizando a mesma operação duas vezes. Para alguns comandos isso não gera nenhum tipo de problema grave, mas se considerada a ocorrência desse fato sob a execução de um comando “EnviarPac”, ocorreria a escrita duplicada de uma mesma linha de código no arquivo, comprometendo sua integridade e causando, futuramente, um erro que abortaria todo o procedimento quando fosse executada a verificação do *checksum* do arquivo. Como já citado, perdas de mensagens é um problema recorrente na comunicação remota entre dispositivos, considerando que o comando “EnviarPac” é o comando mais realizado durante a estratégia proposta, isso poderia prejudicar de maneira significante o desempenho e funcionamento da arquitetura proposta.

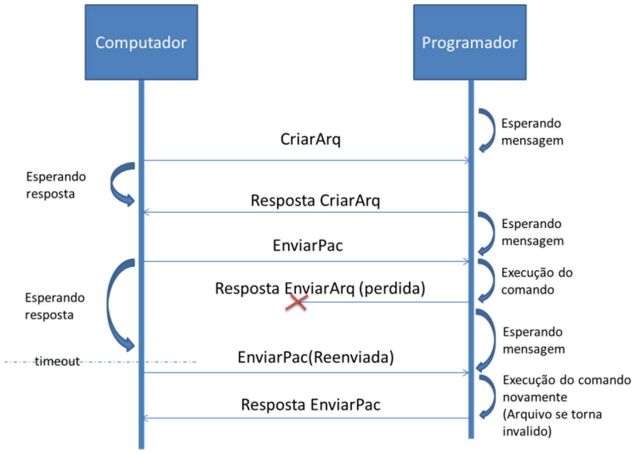

Figura 9. Problema gerado pelo reenvio de mensagens

O mecanismo de bit alternado foi utilizado exatamente para solucionar o problema das mensagens duplicadas. A ideia principal por trás desse mecanismo é que sempre é enviado junto a todas as mensagens e respostas um bit em seu cabeçalho com valor 0 ou 1, o qual funciona como um marcador para que os dispositivos saibam se a mensagem recebida é a esperada ou se é alguma retransmissão. Tanto o computador quanto o programador possuem variáveis internas em seus códigos que armazenam qual é o valor esperado do bit alternado na próxima mensagem. Caso o valor do bit da mensagem que chega ao programador não seja o esperado, entende-se que houve a perda da ultima resposta enviada, fazendo com que ela seja reenviada ao computador.

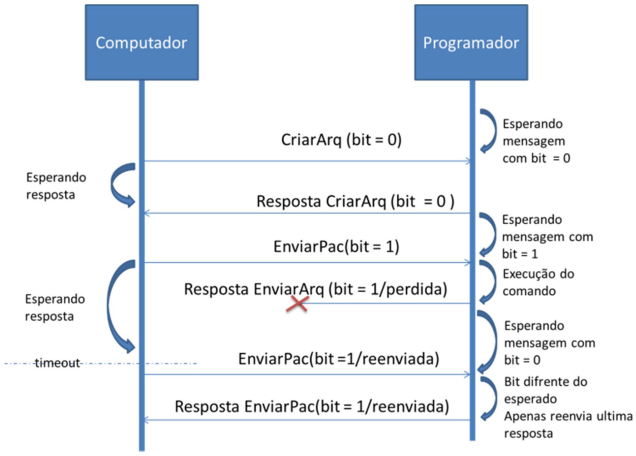

Figura 10. Solução gerada pelo bit alternado

No caso de o bit chegar ao computador com um valor diferente do esperado, entende-se que está acontecendo algum tipo de erro de sincronismo, pois o programador está esperando mensagens com um valor do bit diferente do valor esperado pelo computador, isso gera um erro e aborta todo procedimento, pois a troca de mensagens foi comprometida.

Esse problema de sincronismo pode ocorrer principalmente quando os dispositivos são iniciados, sendo assim é necessário algum tipo de operação que permita que antes de eles começarem todo o processo de atualização remota de *firmware* eles sincronizem o valor de suas variáveis do bit alternado para o mesmo valor - sendo exatamente essa a funcionalidade do comando “Sincronizar”.

Esse comando é efetuado sempre que o programa presente no computador é iniciado, para que ambos os dispositivos troquem uma mensagem, sendo ela aceita pelo programador, independente do valor do bit alternado, possibilitando o sincronismo de suas variáveis para um mesmo valor. Quando ocorre algum tipo de *reset* no nó sensor, um sincronismo também deve ser realizado, mas ao invés de efetuado um comando “Sincronizar”, sempre que isso ocorre o programador aceita a próxima mensagem não importando o valor do bit alternado e atribui esse valor a sua variável de controle, fazendo com que novamente ambos os dispositivos estejam sincronizados.

4. MATERIAIS

Para o desenvolvimento e teste do projeto foi necessária a utilização de diversos materiais. A seguir estão citados e justificados o uso de todos os materiais que foram utilizados durante a pesquisa.

- **BE900:** Módulo transmissor e receptor (transceptor) de rádio. Atualmente é o módulo utilizado pela plataforma Radiuino.

Figura 11. Módulo transceptor BE900

- **DK105:** Placa de desenvolvimento com encaixe para o módulo BE900. Essa placa permite a interface com diversos sensores e é a atualmente usada nos nós sensores da plataforma Radiuino.

Figura 12. Placa DK105

- **Programador UartSBee v4.0:** Conversor USB/serial que permite a comunicação entre computador e o módulo BE900. Através dessa comunicação, é possível realizar a programação do microcontrolador presente nos módulos BE900 pelo uso do protocolo utilizado pelo ambiente de desenvolvimento integrado (IDE) Arduino[7] durante o processo de *upload* de *firmwares*.

Figura 13. Programador UarTSBee v4

- **Computador**: Necessário para toda a codificação das aplicações desenvolvidas no projeto e do envio dos comandos para a base da RSSF.

- **Seeeduino:** Placa de desenvolvimento equipada com um microcontrolador ATMEL 328. Utilizado como o programador do nó sensor na arquitetura proposta. A escolha dessa placa em especifico foi simplesmente pelo fácil acesso a esse tipo de placa pelo autor. Qualquer outro tipo de placa compatível a essa poderia ser utilizada.

Figura 14. Placa Seeeduino

- **SD Card Shield v3.0 :** Placa que permite o uso de cartões de memória por microcontroladores. Ele é acoplado ao programador do nó sensor para permitir o acesso aos dados presentes no cartão de memória.

Figura 15. SD Card Shield v3.0

- **Micro SD Card 4GB:** Cartão de memória para armazenamento dos *firmwares* que serão gravados no nó sensor.

Figura 16. Micro SD Card

5. MÉTODO

O método de desenvolvimento utilizado consistiu das etapas de:

- Desenvolvimento das aplicações necessárias
- Teste de funcionalidades via conexão USB
- Teste de mecanismos de resiliência a falhas via conexão USB
- Testes de funcionalidades via conexão rádio

5.1. Desenvolvimento das aplicações necessárias

Inicialmente, para possibilitar o teste do protocolo proposto, foi necessário o desenvolvimento das aplicações, APL_FOTAP_COMP e APL_FOTAP_PROG, por eles serem os programas responsáveis, na arquitetura proposta, de realizar toda a troca de mensagem e execução de seus respetivos comandos. Com ambas as aplicações desenvolvidas, iniciou-se a fase de testes.

5.2. Teste de funcionalidades via conexão USB

O primeiro conjunto de testes teve seu foco no funcionamento dos mecanismos básicos do protocolo e execução dos comandos pelo programador. Para realizar esses testes, sem ter de se preocupar com problemas gerados pela comunicação via rádio, como por exemplo, perda de mensagens, optou-se inicialmente por realizar os testes com conexão via USB entre as duas aplicações.

O ambiente em que os testes foram realizados é composto pelo computador, o programador e o nó sensor. Nesse ambiente de teste não foi utilizada a placa DK105 conectada ao microcrocontrolador com módulo de rádio, devido a alguns problemas ainda desconhecidos do funcionamento da programação com o microcontrolador acoplado a essa placa. O computador foi conectado ao programador por meio de um cabo USB e o programador foi conectado ao nó sensor através de suas portas de entrada e saída com a utilização de fios *jumpers*.

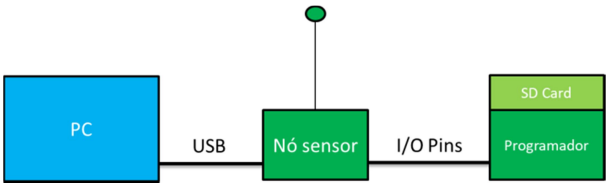

Figura 17. Ambiente de realização dos testes

O esquema de conexão das portas entre o nó sensor e o programador foi o seguinte:

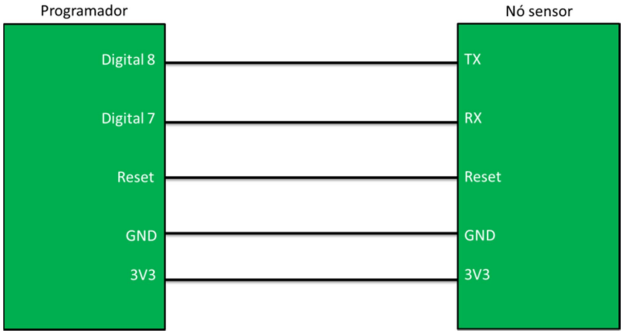

Figura 18. Esquema de conexão entre nó sensor e programador

Considera-se que os pinos 7 e 8 nesse caso são respectivos aos pinos TX e RX do mecanismo de *software serial* utilizado para simular uma porta serial durante os testes.

5.3. Teste de mecanismos de resiliência a falhas via conexão USB

Após as funcionalidades terem sido testadas, foram realizados os testes dos mecanismos de *timeout* e bit alternado, simulando por meio de *software* e *hardware* os eventos que colocariam esses mecanismos a prova.

5.4. Teste de funcionalidades via conexão rádio

Os testes realizados via conexão USB foram replicados, porém dessa vez utilizando a plataforma Radiuino para realizar a comunicação por radio entre os dispositivos.

6. RESULTADOS E DISCUSSÃO

Os objetivos dos testes foram inicialmente averiguar se todas as funcionalidades utilizadas durante a programação remota de *firmware* funcionavam corretamente e se era possível realizar a gravação de um novo *firmware* no nó sensor para em seguida verificar se os mecanismos de resiliência a falhas também funcionariam corretamente.

6.1. Análise dos resultados dos testes de funcionalidades via conexão USB

Após a realização dos testes, foi possível verificar o correto desempenho de todas as funcionalidades desenvolvidas. Tanto o computador enviou as mensagens e tratou as respectivas respostas corretamente quanto o programador executou todos os comandos e enviou suas respectivas respostas com sucesso. Durante esses testes, tentou-se programar um código do tipo *Blink* no nó sensor. Esse teste funcionou corretamente e foi possível realizar a gravação desse código no nó sensor. Entretanto quando se tentou realizar o mesmo procedimento com o código utilizado nos nós sensores da plataforma Radiuino, ocorreu que o nó sensor depois de programado, parou de se comunicar com a base da RSSF por motivos desconhecidos, ainda em análise.

6.2. Análise dos resultados dos testes dos mecanismos de resiliência a falhas via conexão USB

Com a certeza de que todas as funcionalidades operavam corretamente, foi possível realizar os testes dos mecanismos de resiliência a falhas. Foi possível verificar pela simulação de eventos de perdas de mensagens, que o mecanismo de *timeout* funcionou corretamente para realizar o reenvio das mensagens. E que mesmo sendo realizado novamente o envio de uma mesma operação, sua execução não foi efetuada mais de uma vez devido ao mecanismo de bit alternado, pois no fim do procedimento de *upload* de *firmware* a programação ocorreu corretamente. Levando em conta esses resultados foi possível concluir o correto funcionamento desses mecanismos.

6.3. Análise dos resultados dos testes de funcionalidades via conexão radio

Durante a realização dos testes ocorreram diversas dificuldades na utilização da comunicação via rádio para o envio de mensagens entre os dispositivos. Ainda não são de conhecimento do autor os exatos motivos desses problemas encontrados e os mesmos seguem sendo investigados para tentar identificar uma solução que não prejudique a proposta da pesquisa.

7. CONCLUSÃO

Com os resultados obtidos nessa pesquisa foi possível verificar que o protocolo desenvolvido é capaz de ser utilizado para a realização da gravação de um novo *firmware* em um nó sensor e que ele é robusto o suficiente para resistir a falhas recorrentes da comunicação via rádio. Sendo assim é possível concluir que a pesquisa teve êxito em sua proposta de mostrar esse protocolo é confiável para esse tipo de operação. Entretanto ainda existem muitos estudos e testes a serem feitos, pois ainda é necessária a validação da plataforma Radiuino como uma forma funcional de transmissão de mensagens sob a arquitetura proposta. É, portanto, crucial conseguir que os nós sensores, depois de programados com o código utilizado pelos nós sensores da plataforma Radiuino, mantenham-se em comunicação com a base da RSSF. Concluídas essas etapas é possível concluir que a arquitetura de aplicação proposta nessa pesquisa é funcional e adequado para o procedimento de *upload* remoto de *firmware*.

Futuramente é interessante, após bem consolidado o protocolo e seu funcionamento por meio da comunicação via rádio, a melhoria do FOTAP para um tipo de protocolo que realize o envio e recebimento das mensagens de maneira paralela, evitando que só possa ser recebida uma mensagem de cada vez. Esse tipo de alteração poderia trazer um aumento no desempenho de todo o procedimento de atualização remota de *firmware* proposto. Outro fato a ser averiguado é o agendamento das atualizações dos *firmwares* na RSSF, pois é preciso verificar quais serão os melhores momentos para realizar essa operação, já que os nós sensores ficariam inoperantes por um intervalo de tempo e consequentemente Finalmente, realizar estudos de mecanismos para garantir a segurança dos nós sensores, já que o processo de atualização FOTA, pode trazer certas aberturas que podem ser exploradas por atacantes que desejem prejudicar a infraestrutura da RSSF [8].

8. AGRADECIMENTOS

- Primeiramente gostaria de agradecer a reitoria da PUC – Campinas pelo financiamento desta bolsa de iniciação científica.
- Agradeço ao orientador profº drº Omar Carvalho Branquinho pela orientação e apoio durante a pesquisa.
- Agradeço aos meus familiares, principalmente meu pai Amaury Mausbach Filho pelo apoio e ajuda com a pesquisa.
- Agradeço a minha namorada Bianca da Cunha Bandeira Rodrigues pela ajuda com a revisão gramatical e estrutural desse documento.

9. REFERÊNCIAS

[1] HABERMANN, A. N. **Dynamically modifiable distributed systems**. CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1978.

[2] SHAVIT, Moshe; GRYC, Andy; MIUCIC, Radovan. **Firmware update over the air (FOTA) for automotive industry**. SAE Technical Paper, 2007.

[3] SKAN, Peter L. **Method for over-the-air firmware update of NAND flash memory based mobile devices**. U.S. Patent n. 7,698,698, 13 abr. 2010.

[4] RADIUINO. 24/07/2018. http://www.radiuino.cc/.

[5] ATMEGA328, Atmel. P datasheet complete, 2016-06 [Internet] URL: http://www. atmel. com/Images. **Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_datasheet. pdf [Hämtad: 2016-09-29]**.

[6] KUROSE, James F.; ROSS, Keith W. Redes de Computadores e a Internet. **Uma nova**, 2006.

[7] ARDUINO. 30/07/2018. https://www.arduino.cc/

[8] DEVANBU, Premkumar; GERTZ, Michael; STUBBLEBINE, Stuart. Security for automated, distributed configuration management. In: **ICSE Workshop on Software Engineering over the Internet**. 1999.

10. APENDICÊS

Todos os códigos desenvolvidos para essa pesquisa podem ser obtidos no repositório pessoal do autor https://github.com/Gigarrum/FOTA