Trabalho IoT

Guilherme B. Freire[1], **Raphael C. Almeida**[1], **Vítor Augusto Vasconcellos**[1], **Thales Freitas**[1]

[1]Departamento de Ciência da Computação – Universidade Federal do Rio de Janeiro (UFRJ)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{guilhermebfreire,raphaelpda140, vasvas10@gmail.com, thls.mglhs}@gmail.com

Abstract. *Our project consists of creating a network of IoT devices taking advantage of Data Fusion techniques. A wireless network of five nodes was built. Four of which collect data through sensors and a fifth which is responsible for receiving and combing the data. This fifth node also handles the external interactions with entities outside this local network.*

Resumo. *Nosso trabalho consiste em montar um projeto de IoT com a implementação de uma técnica de fusão de dados. Para isso cinco nós foram criados em uma rede de sensores sem fio. Quatro desses coletam dados através de sensores e o quinto fica responsável por recebê-los, combiná-los e enviá-los para fora da rede.*

1. Introdução

Para contextualizar o projeto, pensamos que esse cenário seria empregado em um sistema que necessite de uma constante e frequente atualização dos dados coletados. Esse sistema interage com a rede de sensores através do quinto nó. É interessante que mesmo com essa constante requisição ao quinto nó, os outros sensores não tenham suas baterias prejudicadas por terem que enviar dados constantemente.

Para fazer essa proposta dar certo temos que minimizar o fluxo de rede. Afinal essa é a parte responsável pela maior parte do consumo de energia no mote e mesmo assim ser capaz de responder ao usuário as informações sempre que necessário. Para suprir essas necessidades, precisamos de um algoritmo que responda ao usuário informações relevantes, mesmo que ainda não existam dados atualizados no sistema. Ou seja, o quinto mote deve ser capaz de prever os resultados enquanto não recebe novas informações dos nós sensoriais.

Essa dualidade entre estar sempre disponível e economizar bateria somado ao hardware precário do mote deu espaço a um método leve, simples e eficiente de fazer previsões a respeito dos dados futuros. Acabamos optando pela técnica de Mínimos Quadrados. A ideia é aplicá-la sobre os dados já capturados anteriormente de forma a gerar uma reta que pode indicar o valor do dado para um tempo futuro. Entretanto, essa abordagem a princípio apresenta dois problemas:

1. Para gerar a reta que melhor se adequa aos dados, precisávamos guardar todos os pontos anteriores. Na prática isso se torna inviável devido à memória limitada do SKY mote (10KB).
2. Os dados antigos tinham a mesma influência que os novos na reta que estamos gerando. Isso implica em um erro cada vez maior ao longo do tempo, visto que os dados em geral não seguem uma reta, mas uma função mais complicada.

Para mitigar o problema do erro, acabamos encontrando uma técnica conhecida como Kalman Filter. Esse algoritmo consiste em uma ponderação extremamente simples, mas poderosa. Aplicando um peso ao dado previsto e o dado medido, é possível aproximar a previsão do valor real sem ser afetado por ruídos. Dessa forma, tornamos a previsão do mínimos quadrados mais próxima da realidade e suavizamos os erros de medida encontrados nos sensores dos motes.

Aplicar esse filtro melhorou consideravelmente a proximidade dos valores previstos com os reais. Entretanto, ainda estávamos com o problema de uma divergência crescente devido ao erro acumulado ao longo do tempo. Para resolver esse problema, optamos por armazenar apenas os últimos 20 valores recebidos. Dessa forma a reta gerada pelo mínimos quadrados será mais fiel às últimas tendências recebidas e a as previsões serão melhores.

Um ganho adicional que ganhamos ao optar por esse corte foi em armazenamento. Agora só é necessário guardar os últimos 20 dados, e não todos os dados recebidos desde o início. Isso proporcionou uma enorme economia de memória - o que é crucial para o bom funcionamento do sistema como um todo.

Apesar de tornar a operação viável utilizando apenas os 20 dados mais recentes, isso ainda requeria em média 22KB de memória. Para um aparelho tão simples como o SKY mote, isso se mostrou demais, afinal ele possui apenas 10KB de memória. Então procuramos uma solução que utilizasse menos memória ainda.

Com isso em mente bolamos um algoritmo de mínimos quadrados iterativo. Isso significa que condensamos todos os dados anteriores em um único estado. Utilizando esse estado junto com a próxima leva de dados, calculamos a mesma reta do mínimos quadrados convencional. Dessa forma temos um algoritmo que guarda apenas dois estados - passado e atual; e apenas com esses dados, é capaz de gerar a reta utilizada na predição dos valores medidos. Isso gerou uma economia tremenda de memória. Passamos dos 22KB originais para apenas 672B - uma economia de 97%.

2. Implementação

Com o problema de memória resolvido resta apenas melhorar a questão da divergência ao longo do tempo conforme mais dados são recebidos. Para resolver essa situação, voltamos à mesma ideia de considerarmos apenas os valores mais recentes. Entretanto, dessa vez não temos os valores discretamente disponíveis para considerarmos apenas os mais recentes, em vez disso temos um único valor que representa o passado do que já foi recebido. Então para simular o mesmo efeito, pensamos em uma aplicação de decaimento exponencial.

Antes de unirmos os dados do passado com o atual para fazer uma passada do mínimos quadrados iterativo, multiplicamos esse passado por um valor de decaimento. Dessa forma, os dados mais velhos vão se tornando exponencialmente mais irrelevantes e os novos têm uma proeminência maior nas tendências das predições futuras.

Essa evolução nos métodos de predição e os resultados dos experimentos foram acompanhadas de testes feitos em Python. Esses testes tiveram o intuito de validar as propostas que bolamos e verificar se realmente houve um ganho significativo. O gráfico a seguir ilustra a comparação entre os diversos métodos que testamos com dados artificiais.

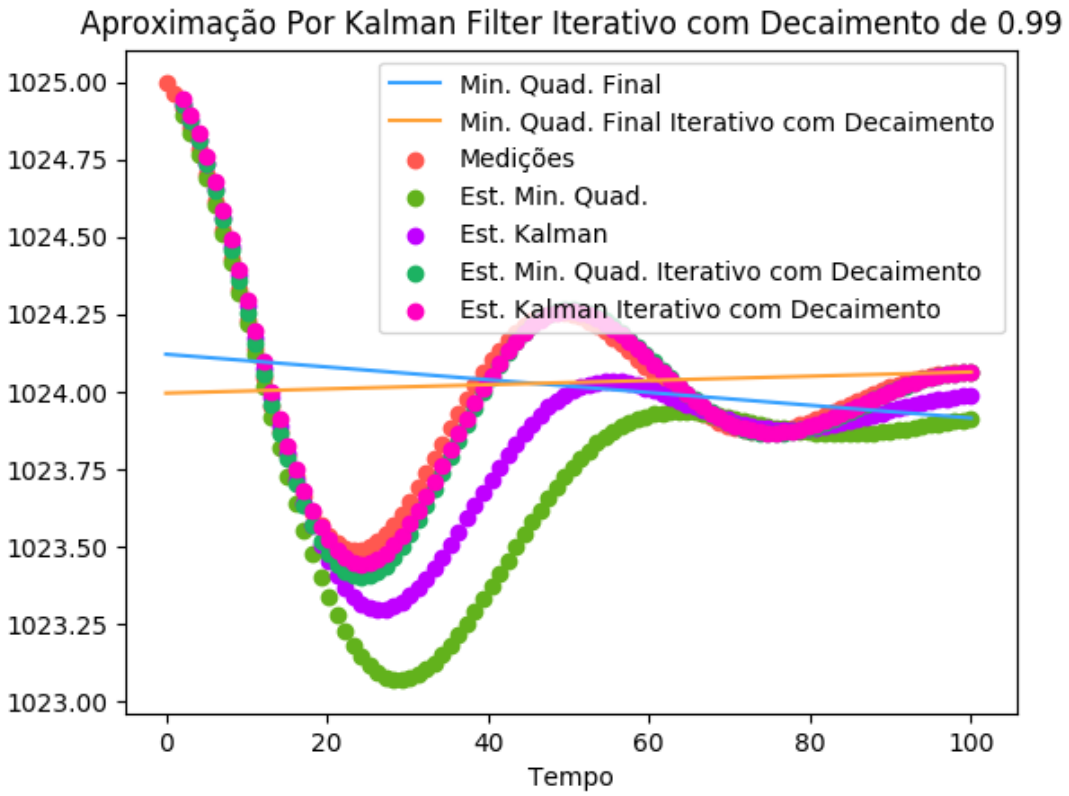

Figura 1. Comparação entre métodos de previsão

A primeira coisa a notar é a progressiva melhoria que obtivemos a cada iteração do algoritmo de predição. Vemos que os pontos verdes, gerados por um mínimos quadrados puro, foram os mais longes dos valores reais (representados em laranja). Ao aplicar o Kalman Filter em cima dos pontos verdes, obtivemos uma melhoria significativa nas predições (representados em roxo). Com a ideia do decaimento implementada, a proximidade das predições com os dados reais se torna quase indistinguível. Finalmente, aplicando o Kalman Filter e o decaimento em cima do mínimos quadrados, percebemos que os dados se também se aproximam dos valores reais, mas bem mais rápido do que a implementação sem Kalman Filter. Sendo esses últimos valores extremamente próximos aos valores reais.

Outra curiosidade interessante a se notar é a diferença de inclinação das duas retas finais dos mínimos quadrados. Em um primeiro momento percebemos que as modificações em cima do mínimos quadrados puro realmente surtiram efeito. Afinal, a reta final do método puro possui um coeficiente angular negativo, ao passo que a reta com decaimento e Kalman Filter aplicados possui um coeficiente angular positivo. Em um segundo momento, vemos que essas mudanças realmente refletem a melhoria das mudanças implementadas. Isso fica claro ao supormos que haja uma interrupção no fluxo de dados medidos. A reta com as técnicas aplicadas em cima do mínimos quadrados levaria a predições muito mais condizentes com o que imaginamos serem os valores reais do que a reta do método puro.

Com todas essas mudanças e iterações no método inicial conseguimos atingir o ponto que queríamos: capacidade de previsão com pequena taxa de erro e baixo consumo de memória que eram os objetivos iniciais. Como realizamos poucas operações por iteração do mínimos quadrados, também ganhamos um ganho de bateria em relação ao método não iterativo.

Um lado negativo do método que implementamos está na quantidade mínima de dados necessários até que as predições comecem a ser condizentes. Outra limitação da implementação se encontra no modo como unimos os dados dos quatro motes em um só. Para realizar essa fusão, utilizamos uma média simples. Isso significa que qualquer discrepância entre os valores dos motes sensoriais será atenuada pelos valores dos outros. Uma possível solução para isso seria guardar, por exemplo, o desvio padrão também. Dessa forma, mesmo tendo apenas a média como dado concreto, sabemos a discrepância dos valores individuais em relação a ela. Isso facilita a detecção de algum evento que possivelmente seria mascarado observando apenas a média.

Outro problema encontrado foi na implementação. O método proposto é complexo de ser implementado em C. Como consequência a quantidade de código estourou a memória ROM do SKY mote, que possuia apenas 48KB. Optamos, então, por migrar o projeto para o Z1. O Z1 possui 96KB de ROM, o que se mostrou suficiente para implementar os métodos em questão.

Para conexão e transmissão de dados usamos protocolo UDP sobre IPv6. Nela, os motes que enviam os dados iniciam um RPL DAG (routing protocol low-power directional acyclic graph) e o mote receptor entra como raiz nesse DAG. Isso forma, então, uma RPL DODAG (routing protocol low-power destination oriented directional acyclic graph). Sabemos que em protocolo UDP mensagens perdidas não são reenviadas, entretanto a nossa expectativa que os algoritmos de predição sejam fiéis o suficiente para calcular um valor aproximado. Tomando essa decisão, temos um ganho na economia de energia, afinal não temos retransmissões. Efetuamos diversos testes com taxas de erro distintas - inclusive em cenários onde a taxa de erro não é extremamente alta - e o nó que faz a fusão consegue fazer previsões de forma extremamente satisfatória.

Nosso grupo teve diversos problemas para utilizar e desenvolver nosso projeto. Alguns desses problemas já foram citados acima - como pouca memória ROM e RAM. Além dessas questões, também encontramos complicações com a versão desatualizada da VM do Contiki OS no site oficial. Precisamos descobrir que o Cooja se tratava de um repositório git no qual deveríamos executar git pull para receber as últimas atualizações.

Além disso, a arquitetura dos motes é extremamente simples e não é capaz de executar print em float. Esse fator atrapalhou consideravelmente o processo de debug e avaliação de corretude do código. Tivemos que implementar uma função printDouble, responsável por executar o print dos dados em formato double. Entretanto o texto efetivamente mostrado era convertido para fixed point, o que gerou um problema em se tratando de números especiais em ponto flutuante. O mais proeminente deles decorria do fato de que em determinado momento no código um NaN era gerado (através de uma divisão por zero). Como NaN é mapeado para 0 no tipo inteiro, nosso print incorretamente mostrava o número 0. Uma consequência disso é que o resultado de qualquer operação com esse valor resultava em zero. Um exemplo disso era uma subtração, 4-0, que retornou 0 como

resultado. Só fomos capazes de detectar e solucionar o problema depois de descobrir e usar ferramentas de debug presentes no próprio Cooja.

Outro problema foi com a arquitetura - que é de 16 bits. Acostumados com outro paradigma de programação, assumimos que estávamos programando como se fosse de 32 bits. Fator este que gerou muitos valores estranhos derivados de overflows que a princípio não percebemos. Para todos esses casos a documentação ajudou pouco, ou simplesmente não ajudou, caracterizando um problema grave no Contiki OS. A documentação é rasa demais para o tipo de problema que tivemos. Com relação às partes mais importantes e relevantes do código, podemos citar para o nó que recebe as informações, as funções abaixo:

1. Receive: Efetua o recebimento dos dados e chama as funções responsáveis por armazenar e atualizar os modelos de predição disponíveis.
2. Combine data: Faz média dos dados que serão enviados como input para função de update que atualiza o modelo de previsão baseado nos novos dados.
3. Update: Efetua operações de atualização de modelo, é responsável por gerar a função de predição que será usada até que novos dados cheguem ao sistema.
4. Predict: Utiliza o modelo atualizado pelo update e faz predição do valor de determinado dado em um tempo futuro t.

Para os nós que fazem o envio dos dados, podemos citar:

1. Sender_process: responsável por coletar e enviar periodicamente os dados ao nó que efetua a fusão dos dados.

3. Experimentos

Para os testes fizemos um código de automatização em Javascript. Este código roda através do Cooja Simulation Script Editor e possibilitou testes em grupos de 30 em 30, todos executando simultaneamente. Cada bloco de 30 são referentes a 30 testes de 30 minutos de funcionamento da nossa rede para uma dada taxa de sucesso em transferência. No nosso caso fizemos testes em intervalos de 10%, com as taxas de sucesso de transmissão variando de 10% a 100%. Ou seja, no total, para executar os testes foram necessárias 300 testes de 30 minutos cada para obtenção dos logs que foram usados para gerar os resultados finais. Nesses testes foram avaliadas duas grandezas. A primeira é o consumo energético, como podemos observar no gráfico da Figura 2.

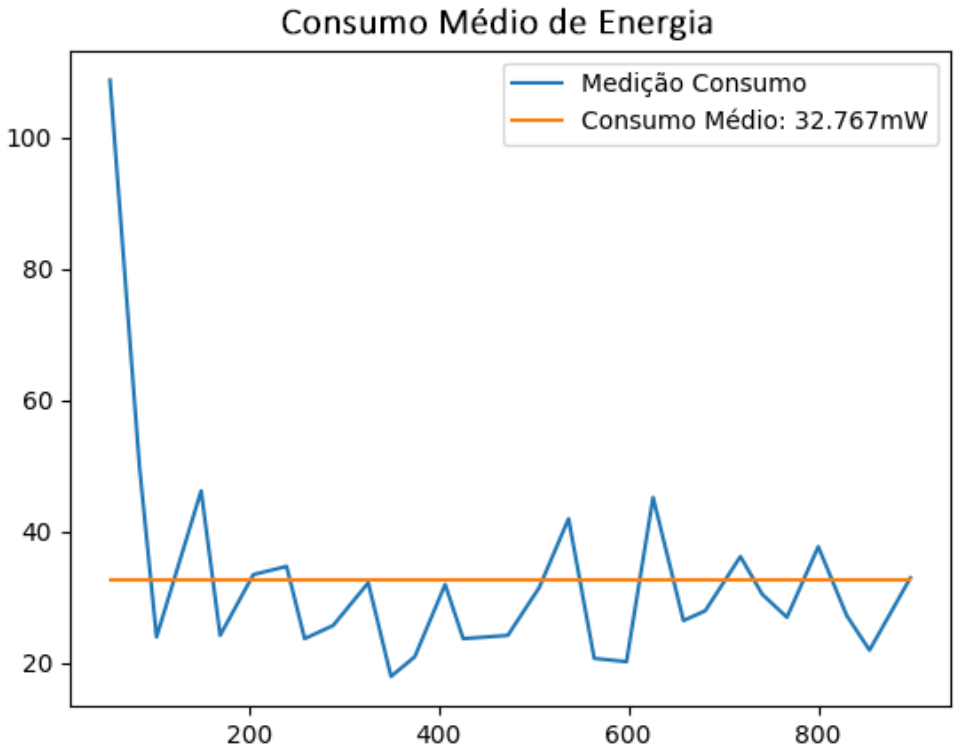

Figura 2. Consumo de Energia

Após as 30 execuções fizemos a média do consumo energético e com base nisso calculamos a média de consumo em mW ao longo de todo o experimento. Temos, agora, a taxa de consumo de energia por segundo. Analisando o gráfico, vemos que no início temos um gasto muito maior de energia, isso é dado pela inicialização do mote e sua inserção na rede. Mais para frente o consumo se estabiliza em um nível mais baixo e com picos periódicos. Isso reflete exatamente a atividade realizada pelo mote em coleta e transmissão de dados.

Outra análise que julgamos interessante fazer, foi o tempo de vida de um mote com uma carga de trabalho dessas. Sabendo da capacidade de uma pilha alcalina de alta duração e que o mote Z1 que estamos usando tem 2 pilhas AA como alimentação, fomos capazes de estimar exatamente esse tempo. Fazendo cálculos simples, conseguimos chegar ao valor de 238 horas ou aproximadamente 10 dias. Para obter esses resultados consideramos que o consumo advém inteiramente do processamento e da transmissão. Isso é uma suposição razoável, uma vez que todos os outros componentes do Z1 tem um consumo na ordem de μW.

Para os testes do fluxo de informações com taxa de acerto variável, usamos como dado a ser transferido os coletados por dois sensores de luminosidade - um de luminosidade solar e outro de luminosidade fotossintética. Os dados foram gerados com os testes descritos anteriormente e a média dos resultados está no gráficos abaixo das Figuras 3 e 4.

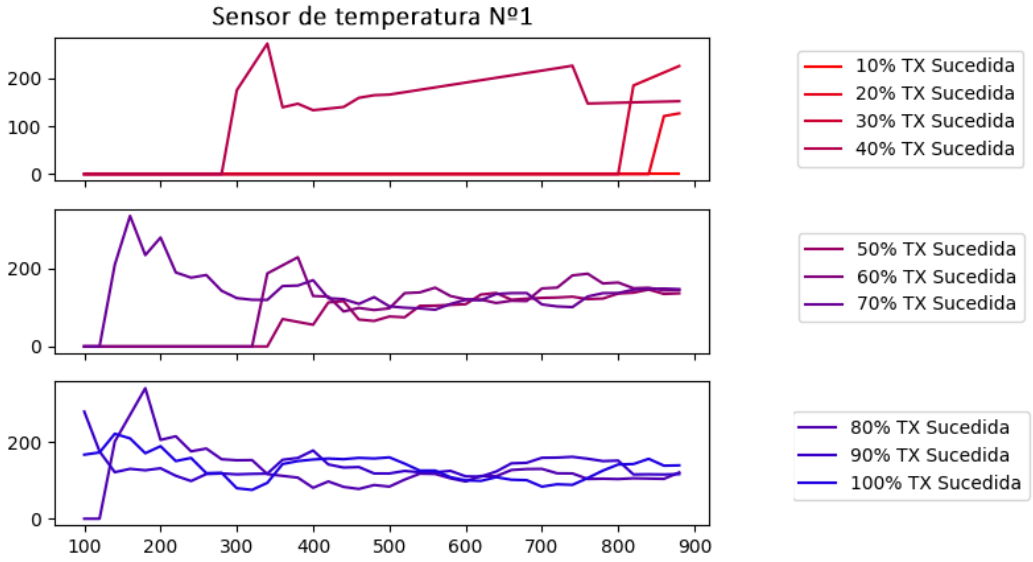

Figura 3. Sensor de Luminosidade Fotossintética

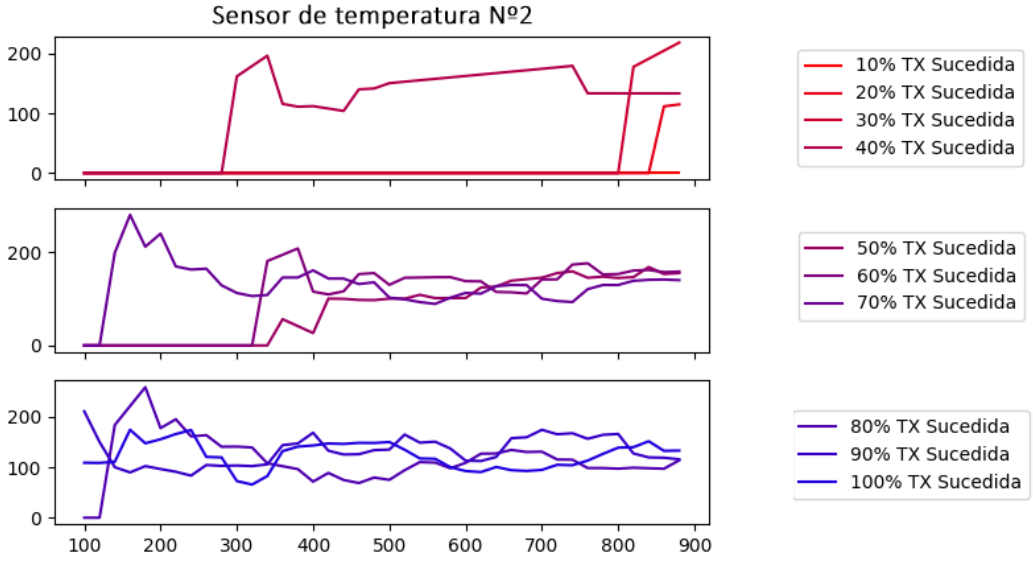

Figura 4. Sensor de Luminosidade Solar

Devemos lembrar e levar em consideração o fato de que as previsões estão acontecendo nesses gráficos. Dessa forma, mesmo com perda de informação na transmissão, os dados ainda podem ser previstos e consequentemente podemos extrair informações ainda que elas não estivessem disponíveis no momento. Além disso, podemos perceber que até com 50% de perda o comportamento da curva após receber uma quantidade suficiente de dados é incrivelmente comportada o que mostra um excelente resultado dos métodos empregados. Mais do que isso, esse resultado se repete tanto para o sensor de luminosidade

1 quanto para o sensor de luminosidade 2.

Na Figura 5 temos o gráfico de uma simulação feita em Python usando o nosso algoritmo. Esse gráfico foi utilizado para validar nossos algoritmos. Nele, vemos um fluxo de dados com 50% de taxa de erros, um valor extremamente acima de situações convencionais. Ainda assim, os resultados são muito satisfatórios. Podemos perceber que para muitos pontos no gráfico, nos quais o receptor não obteve informações, o estimador foi capaz de estimar com sucesso grande parte dos dados. Isso é principalmente verdade para as partes do gráfico onde a variação ocorre de forma menos abrupta.

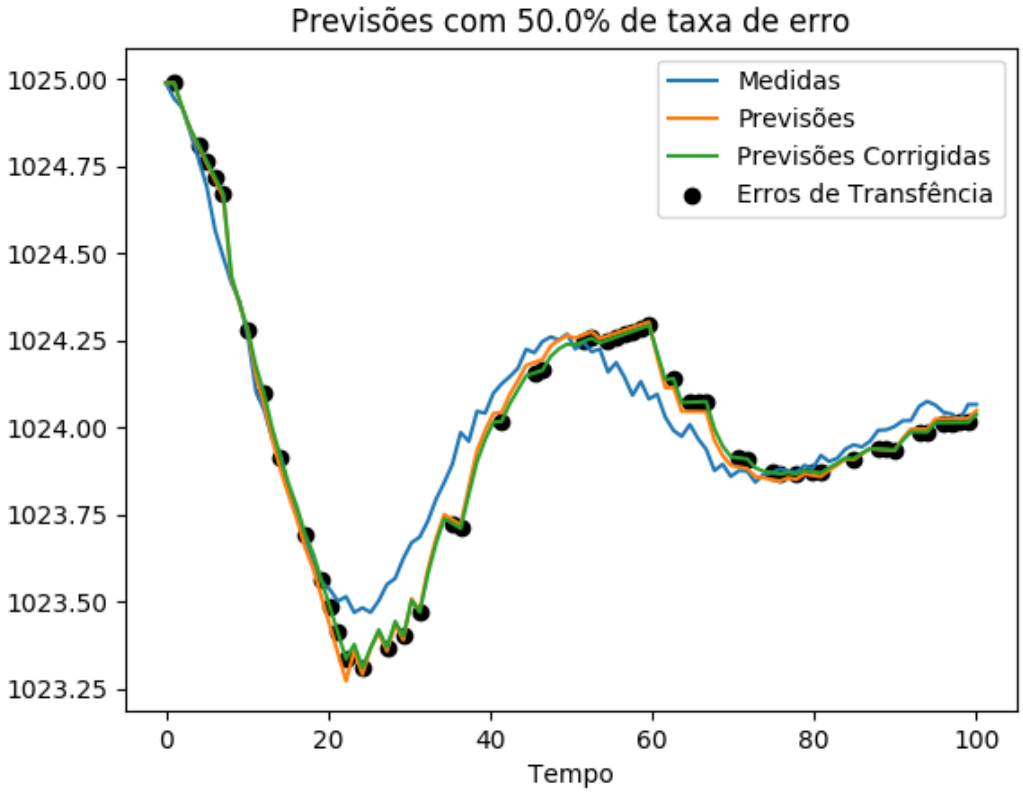

Figura 5. Simulação de Previsão com 50% de Erro de Transmissão

4. Conclusão

Mediante a todos esses gráficos e informações, mesmo após longas e sucessivas rodadas de testes, o comportamento da previsão se manteve coerente. Para diversas variações de taxas de erro - inclusive para valores extremamente acima do normal - o nosso modelo foi capaz de se recuperar e gerar dados relevantes. Dessa forma, podemos perceber que para aplicações onde se deseja ter uma maior duração de bateria, pode-se usar o método proposto para atingir esse fim sem prejudicar muito a acurácia dos dados. Além disso, nosso método possibilita usar predição em aplicações onde precisa-se ter alta disponibilidade em momentos de falta de dados por falha de transmissão ou qualquer outro motivo. Um aspecto interessante é que pode ser feita a combinação desse método com outros métodos de fusão de dados para outros propósitos. Além disso, com a

evolução do hardware dos motes e a disseminação do IoT, torna-se possível usar métodos ainda melhores para previsão de dados e suavização de curvas afim de tornar a precisão do nosso método ainda maior. Por fim, para quem tiver interesse, o projeto pode ser encontrado no link abaixo:
`https://github.com/Raphael-C-Almeida/Wireless-Sensor-Network`.