Sincronização de Relógios (ESP8266)

João Mesquita
FEUP - MIEEC
Porto, Portugal
up201305568@fe.up.pt

João Gomes
FEUP - MIEEC
Porto, Portugal
ee11306@fe.up.pt

Roberto Lima
FEUP - MIEEC
Porto, Portugal
up201502837@fe.up.pt

***Resumo*—A tecnologia atual nos mais diversos sistemas depende cada vez mais de sistemas distribuídos de tempo real. Tal implica a implementação de mecanismos que introduzam uma noção global de tempo entre todos nodos da rede distribuída, permitindo identificar a ordem temporal em que determinado evento ocorreu ou mesmo levar a cabo ações num determinado instante de tempo em todos os nodos. Neste projeto, foram estudados e implementados numa rede de módulos *WiFi* de baixo consumo (*ESP8266*) dois dos algoritmos que permitem a referida sincronização de relógios: algoritmo de Berkeley (cliente-servidor) e sincronização distribuída (com *Fault Tolerance Average* (FTA)). Foram posteriormente realizadas medições comparativas das duas arquiteturas implementadas, tendo-se consigo obter uma maior precisão utilizando o algoritmo de Berkeley.**

***Palavras-chave*—Clock synchronization, ESP8266, Berkeley Algorithm, Distributed clock synchronization, FTA, TDMA**

I. Definição do problema

Num sistema distribuído cada nodo tem o seu próprio relógio interno, sendo que sem mecanismos de sincronização de relógios não existe uma noção coerente e global de tempo entre cada um dos intervenientes na rede distribuída [2]. São exemplos da importância de existência desses mecanismos, a necessidade de estabelecer a ordem pela qual determinado evento ocorreu numa rede distribuída, a sincronização dos vários nodos no acesso a recursos partilhados, a realização *timestamps* de eventos (exemplo: informação recolhida de sensores) ou a implementação de mecanismo de acesso ao meio (exemplo: *Time-Division Multiple Access* (TDMA)). Isto é particularmente critico quando os relógios das maquinas que constituem a rede distribuída possuem uma reduzida precisão e apresentam *drift rates* elevados que variam entre cada um dos intervenientes, obrigando a uma sincronização periódica de forma a obter um *offset* máximo global reduzido. De entre os vários algoritmos que permitem atingir o referido sincronismo destacam-se os métodos de sincronismo externo e os de sincronismo interno [3]. Os primeiros são referentes a uma fonte externa de relógio (exemplo: *Coordinated Universal Time* (UTC)) que os nodos da rede tomam como correta e atualizam os seus relógios internos de acordo com a mesma. O segundo tipo de algoritmos estabelece a sincronização tendo em conta um tempo comum entre as maquinas participantes em determinada rede, fazendo para isso uma média entre os tempos de cada uma ou tendo como referencia um tempo fornecido por um *time master* presente na rede.

Neste projeto, foram estudados e implementados dois algoritmos de sincronização interna numa rede de módulos *WiFi* de baixo consumo (*ESP8266*): algoritmo de Berkeley (cliente-servidor) e sincronização distribuída (com FTA).

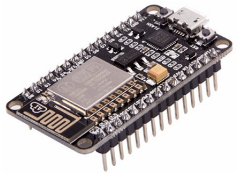

Fig. 1. Módulos *WiFi* de baixo consumo utilizados (ESP8266)

Nas secções seguintes vão ser apresentadas as implementações que realizamos dos dois algoritmos, os resultados obtidos com cada uma e as conclusões que pudemos retirar com o trabalho desenvolvido.

II. Descrição da solução

Antes de iniciar a implementação dos algoritmos, foi necessário perceber qual a melhor forma de manter uma contagem de tempo e poder atualizar a mesma a cada iteração do algoritmo de sincronismo. Optou-se pela utilização do tempo fornecido pelo modulo *Real-time clock* (RTC) e da zona de memória subjacente a este e presente nos nodos utilizados. Apesar do nome atribuído a este modulo (RTC), o mesmo não se comporta como um Real-Time Clock em todos os sentidos, ou seja, é capaz de manter uma contagem de tempo mesmo que o nodo se encontre num estado de *deep sleep*, mas é altamente dependente da temperatura não permitindo por isso obter uma elevada exatidão nas leitura efetuadas. De forma a tentar minorar essa dependência, a *framework* utilizada no desenvolvimento deste projeto (*esp-open-rtos*) dispõe de uma função que permite obter o período atual do relógio RTC tendo em conta fatores como a temperatura do *chip*. Esta *framework* dispõe ainda de funções para leitura dos valor do contador de *ticks* do modulo RTC, mas não permite que esse contador seja alterado diretamente. De forma a contornar tal situação, optou-se por utilizar 3 dos registos presentes na memória do RTC (e que por isso também mantêm o seu valor mesmo em estados de *deep sleep*) da seguinte forma: um registo (*timer_ref*) que guarda o número de *ticks* presente no contador do modulo RTC aquando da chamada da função de atualização do valor do relógio, um registo (*cal*) que guarda o valor retornado pela função que determina o período atual do *clock* RTC e um registo (*time_base*) que guarda a conversão

do novo valor que se pretende para o relógio interno (us) em *ticks*. Com estes três registos criados e respetivas funções de atualização é possível manter uma contagem de tempo e modificar a mesma sempre que necessário. Terminado o desenvolvimento do código necessário para atualização dos registos no modulo RTC, iniciou-se a implementação dos algoritmos de sincronização:

A. Algoritmo de Berkeley (Cliente - Servidor)

O algoritmo de Berkeley, tal como desenvolvido por Gusella e Zatti [1], pressupõe que nenhuma das maquinas presentes na rede é uma fonte fidedigna de tempo com quem as restantes podem estabelecer comunicação e sincronizar os seus relógios internos. Em vez disso, opta por uma arquitetura em que um nodo servidor faz um pedido periódico a todos os restantes nodos da rede para enviarem o seu relógio interno. De seguida, é calculada uma média dos relógios recebidos pelo servidor, eliminado nodos que apresentem um valor muito desfasado dos restantes e tendo em conta os atrasos de rede desde o envio do pedido aos clientes até à receção da resposta. Posteriormente, é enviado, para cada um dos nós que respondeu ao pedido, o valor do ajuste, positivo ou negativo, que devem aplicar ao seu relógio interno e não o valor atualizado do relógio global determinado pelo servidor, evitando desta forma a introdução de novas incertezas resultantes do atraso de rede no envio das mensagens por parte do servidor aos restantes nodos. Na figura 2 apresenta-se um esquema representativo do algoritmo descrito.

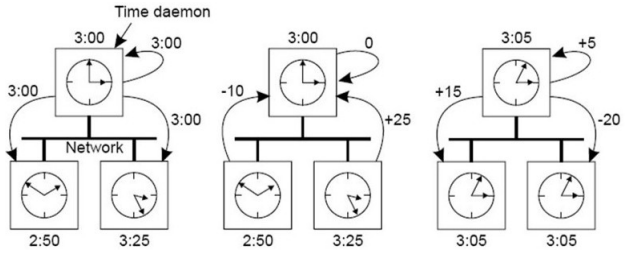

Fig. 2. Algoritmo de Berkeley

No caso particular da implementação realizada e de forma a tornar a mesma independente do facto de o nó servidor ser também utilizado como *Access Point* (AP), optou-se por utilizar UDP *broadcast* na mensagem que o servidor envia com o pedido do relógio interno aos restantes nodos. Desta forma, não é necessário saber à partida quais os nodos que se encontram ligados ao servidor em cada momento. Com o envio, por parte do servidor, da referida mensagem é guardado em memória uma *timestamp* e assim que é obtida a resposta dos restantes nodos é adicionada a informação recebida a um vetor de registos que em cada posição contem informação como o *id* do nodo à qual a resposta é referente, o endereço IP do mesmo, o *round-trip delay* (calculado com base na *timestamp* de envio da mensagem inicial) e o valor de *clock* recebido. Ao fim de um *timeout* pré-definido, o servidor calcula um valor global de *clock* com a média dos valores recebidos e o seu próprio relógio, ignorando para esse calculo os relógios que tenham um *offset* maior que 100 ms. De seguida, é calculado e enviado para cada nodo o desvio que o seu relógio tem em relação ao valor global calculado anteriormente e cada um dos nodos atualiza o seu relógio interno aplicando o desvio recebido. Apresenta-se na figura 3 um diagrama de sequencias representativo do que foi implementado nos nodos.

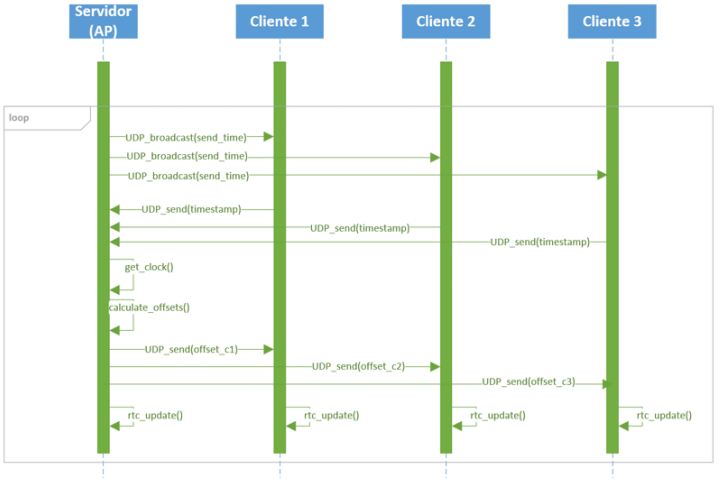

Fig. 3. Diagrama de sequência do algoritmo de Berkeley

B. Sincronização Distribuída (c/ FTA)

O algoritmo de sincronização distribuída implementado pressupõe que não existe um relógio *master* com quem os restantes nodos devem trocar informação dos seus relógios. Pelo contrário, todos os nodos devem partilhar o valor dos seus relógios entre si e determinar uma média FTA com os valores obtidos. De forma a evitar colisões nesta troca de mensagens, o algoritmo foi implementado baseado numa arquitetura de comunicação *Time Division Multiple Access* (TDMA), que consiste em dividir em *slots* o tempo de acesso ao meio para cada nodo. Na camada de transporte foi utilizado UDP *broadcast* para difundir as mensagens entre os nodos.

A figura 4 apresenta a arquitetura de alto nível do algoritmo implementado. Os *nodes* comunicam com os vizinhos trocando mensagens com os seus relógios, ao receberem uma mensagem de sincronismo cada um atualiza o seu relógio interno realizando uma FTA dos valores recebidos.

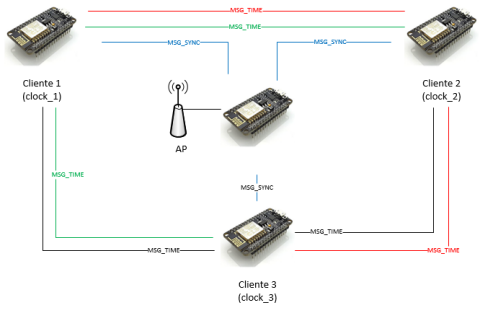

Fig. 4. Sincronização Distribuída

De uma forma mais detalhada, e tal como representado no diagrama de sequência da figura 5, o funcionamento desta solução consiste em um atribuir a um *node* a função de estabelecer a sincronização da comunicação entre os restantes, para isso o mesmo envia uma mensagem de SYNC para os restantes *nodes*. Após um tempo t dessa mensagem de sincronismo, os restantes *nodes* enviam os seus relógios para os vizinhos, esse tempo está relacionado com o tempo de slot (Td) e o identificador do *node* (Id).

$$t = Td * Id \tag{1}$$

Cada *node* vai recebendo os relógios dos *nodes* vizinhos e esses valores vão sendo guardados em memória, até ser recebida a mensagem de sincronismo seguinte. Ao receber essa mensagem, cada um atualiza o seu relógio, calculando a média dos tempos que recebeu. Caso o número total de relógios recebidos seja superior a três, o calculo da média é efetuado utilizando o método FTA, ou seja, ignorando os tempos extremos. Caso contrário, é utilizado para o calculo todos os tempos dos relógios que recebeu.

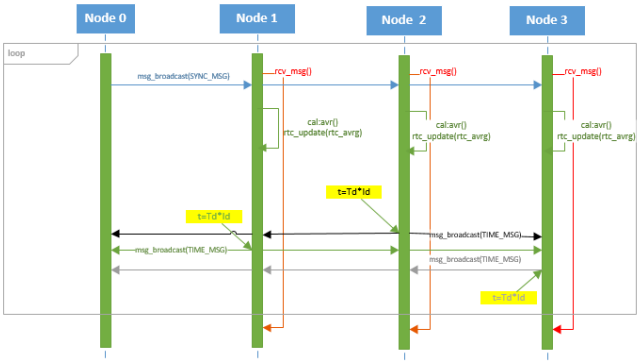

Fig. 5. Diagrama de sequencias sincronização distribuída

III. Resultados

Foram realizados diversos testes recorrendo aos diferentes algoritmos. Em seguida serão apresentados os testes realizados que tiveram valores mais fiáveis e otimistas.

A. Algoritmo Berkeley

O seguinte teste foi realizado com um período de sincronismo de 2 segundos e num ambiente com poucas interferências, determinado pelo grupo como um local ideal. Durante a execução do teste foi adicionado um *node* à rede de modo a ilustrar o comportamento da rede nesta situação. Ao total foram utilizados 4 nós neste teste. A seguinte figura contém no seu gráfico uma representação da máxima diferença entre um relógio remoto, para o relógio do servidor, tomado como referência.

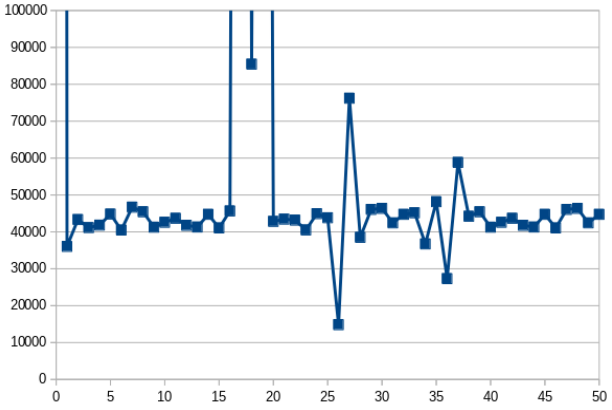

Fig. 6. Offset Máximo Servidor-Cliente

B. Observações Adicionais - Algoritmo Berkeley

O grupo teve acesso a 2 tipos de placas de desenvolvimento com o chip ESP8266, sendo que um tipo era proveniente de um fabricante mais barato, com componentes de menor qualidade. Pode-se ver este facto evidenciado na figura abaixo, reparando no elevado *offset* constantemente observado pelo *node* com o endereço de IP com terminação em 2, em contraste com o observado pelo *node* com o endereço de IP com terminação em 3. O *node* 2 tem uma performance constantemente pior, ilustrando a falta de qualidade do seu cristal oscilatório. À esquerda podemos ver a primeira iteração do algoritmo, como os relógios dos *nodes* estão completamente não relacionados, o servidor toma como referência o seu próprio relógio, mostrando efetivamente um dos casos de uso do algoritmo FTA.

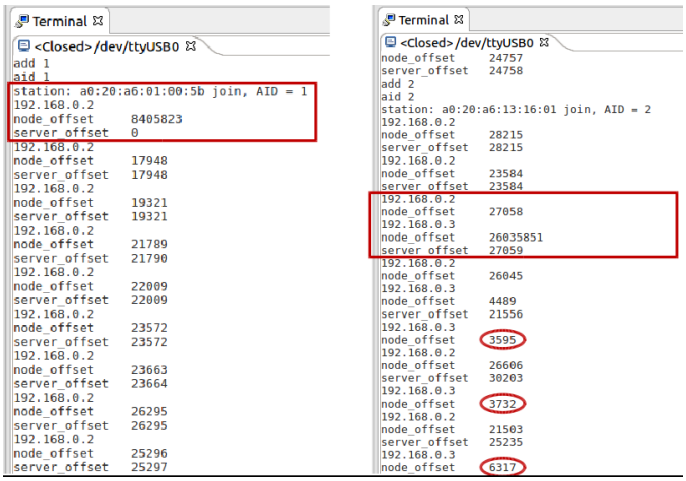

Fig. 7. Offset Máximo Servidor-Cliente

C. Sincronização distribuída

Para este teste foi utilizado TDMA com um tempo de intervalo de 100ms. O tempo do macro-ciclo é de 1000ms e foram utilizados 5 nós na realização deste teste. Durante a execução do teste foi adicionado um *node* à rede de modo a ilustrar o comportamento da rede nesta situação. A seguinte

figura contém no seu gráfico uma representação da máxima diferença entre um relógio remoto, para o relógio do servidor, tomado como referência.

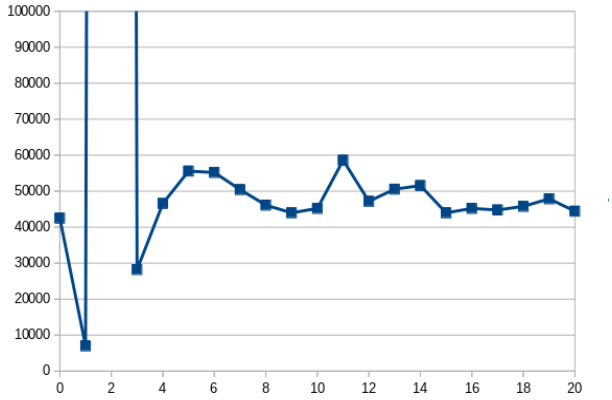

Fig. 8. Offset Máximo de Nó para Nó

IV. Conclusões

Após a finalização deste projeto podemos retirar algumas conclusões referentes aos microcontroladores usados, bem como às arquiteturas de relógio distribuído utilizadas. Um facto inesperado, foi a evidente diferença dos componentes, nomeadamente o cristal, presentes nos ESP8266 utilizados. Tendo acesso a diferentes placas de diferentes fabricantes, foi possível constatar que a mais barata, tinha um maior *drift* associado ao seu relógio, que era constante. Em termos de resultados, foi atingida uma precisão de sincronização de relógio máxima na ordem dos 50ms. Consideramos estes satisfatórios, dada a a topologia da rede utilizada, o meio em que os testes foram efetuados e o hardware utilizado. Contudo, a sincronização de relógio foi mais eficaz utilizando o algoritmo de Berkeley, diferindo aproximadamente 10ms. Ao longo do projeto foram encontradas diversas dificuldades relacionadas com a natureza dos *nodes* utilizados no trabalho. Um dos maiores problemas foi o facto de não haver um processo fiel de *debug*, tendo o grupo recorrido a chamadas de sistema para imprimir informação através da consola de série. Este mecanismo interfere a certo ponto os tempos que foram medidos, sendo que esta interferência não foi contabilizada. O trabalho realizado foi dividido pelos vários estudantes tendo o João Mesquita realizado a parte referente as funções para atualização dos registos do RTC, a implementação do algoritmo de Berkeley e os respetivos testes. O João Gomes realizado parte do código produzido para teste do algoritmo FTA distribuído com TDMA e um *script* em *python* para recolha de dados. O Roberto Lima implementou a comunicação UDP nos dois algoritmos e realizou a outra parte do código do algoritmo FTA distribuído com TDMA.

Em suma, tentou-se distribuir o trabalho de forma igual entre os elementos, mas essa distribuição não foi exatamente igual. Por isso, foi dado uma percentagem a cada elemento correspondente ao trabalho realizado no projeto.

- João Mesquita-35%
- João Gomes-35%
- Roberto Lima-30%

Referencias

[1] Riccardo Gusella and Stefano Zatti. The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD. Technical Report UCB/CSD-87-337, EECS Department, University of California, Berkeley, 1 1987.

[2] Paul Krzyzanowski. Lectures on distributed systems Clock Synchronization. *Rutgers University – CS*, 417, 2009.

[3] Amritha Sampath and C Tripti. Synchronization in Distributed Systems. In Natarajan Meghanathan, Dhinaharan Nagamalai, and Nabendu Chaki, editors, *Advances in Computing and Information Technology: Proceedings of the Second International Conference on Advances in Computing and Information Technology (ACITY) July 13-15, 2012, Chennai, India - Volume 1*, pages 417–424. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.