ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Επεξεργασία Δεδομένων σε Δίκτυα Αισθητήρων

ΠΛΗ 511

Εργασία Εξαμήνου – Μέρος 2ο

«Προσομοίωση Δικτύου Αισθητήρων με αρχή λειτουργίας σύμφωνα με το TiNA»

Part 2: "Επέκταση της λειτουργικότητας του δικτύου για επιλογή νέας συνάρτησης συνάθροισης σε κάθε εποχή"

Πέτρου Δημήτριος – 2018030070

Καθηγήτης: Α. Δεληγιαννάκης

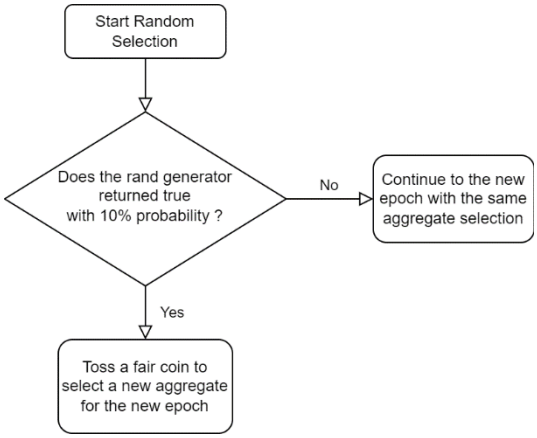
Χανιά, Δεκέμβριος 2022

1. Εισαγωγή

Στο 2^{ο} μέρος της εργασίας τροποποιήθηκε η λειτουργικότητα του δικτύου ώστε σε κάθε εποχή η ρίζα να επιλέγει εκ νέου το σύνολο των συναθροιστικών συναρτήσεων που θα εκτελεστούν από τους κόμβους. Η αλλαγή της συναθροιστικής συνάρτησης συμβαίνει σε κάθε γύρο με πιθανότητα 10 % και η επιλογή της νέας σε περίπτωση που χρειαστεί γίνεται με ρίψη ενός τίμιου νομίσματος (Heads & Tails).

2. Επιλογή Συναθροιστικής συνάρτησης

Στο διπλανό διάγραμμα φαίνεται η ροή για την διαδικασια επιλογής νέου συνόλου συναθροιστικής συνάρτησης για τη νέα εποχή. Η συγκεκριμένη σειρά ενεργειών εκτελείται μόνο σε κάθε επόμενη εποχή και όχι κατά την 1^{η}.



<u>Μέθοδος επιλογής:</u>

Αρχικά κρίθηκε απαραίτητο να υλοποιηθεί μία συνάρτηση η οποία θα ήταν αληθής με 10% πιθανότητα. Η λειτουργικότητα αυτή αντικατοπτρίζεται στην C συνάρτηση:

```c
/*
    Returns true with 10% probability
*/
bool get10PercentTrue(){
    double probability;
    srand(time(NULL));

    probability = (double) rand() / (double) RAND_MAX;
    dbg("Randomness","Probability is: %f\n",probability);

    return probability <= 0.1 ? TRUE : FALSE;
}
```

Στη συνέχεια εφόσον με 10% πιθανότητα έπρεπε να επιλεγεί νέα συναθροιστική συνάρτηση αυτό γινόταν με τον εξής τρόπο:

```c
/*
    Selects new aggregate by tossing a coin
*/
uint8_t randReaggregate(uint8_t currAggregate){

    uint8_t coin;
    uint8_t newAggr;

    coin = rand()%2;
    dbg("Randomness","Coin flip produced %s \n", coin==0 ? "heads":"tails");

    //Select new aggregate among the other two left

    newAggr = coin;

    if(currAggregate==0){
        newAggr = coin+1;
    }

    if(currAggregate==1){
        if(coin==1){
            newAggr += 1;
        }
    }

    dbg("Randomness","Prev Aggr: %d New Aggregate flag is %d (MAX=0, COUNT=1, MAXCOUNT=2)\n",currAggregate, newAggr);
    return newAggr;
}
```

Πιο αναλυτικά, η μεταβλητή coin λαμβάνει μια τυχαία τιμή μεταξύ 0 και 1 (heads-tails). Προκειμένου να αναχθεί η συγκεκριμένη τιμή σε flag επιλογής συναθροιστικής συνάρτησης λαμβάνεται υπόψη η τρέχουσα συναθροιστική συνάρτηση. Η αντιστοιχία μεταξύ flag και συναθροιστικών συναρτήσεων είναι η εξής:

Flag	Aggregate Function
0	MAX
1	COUNT
2	MAX & COUNT

Στην επιλογή νέας συνάρτησης διακρίνονται οι εξής περιπτώσεις όσον αφορά το αποτέλεσμα της ρίψης (έστω coin):

- **Περίπτωση που πριν εκτελούνταν MAX& COUNT:**
  currentAggregate = MAX & COUNT (2)
  newAggregate = coin (δηλ. 0 ή 1 / MAX ή COUNT)

- **Περίπτωση που πριν εκτελούνταν MAX**
  currentAggregate = MAX (0)
  newAggregate = coin+1 (δηλ. 1 ή 2 / MAX ή MAX & COUNT)

- **Περίπτωση που πριν εκτελούνταν COUNT**
  currentAggregate = COUNT (1)
  newAggregate = coin + 1 αν coin = 1 OR newAggregate = coin αν coin=0

Σημειώνεται πως η παραγωγή τυχαίων αριθμών στις συγκεκριμένες συναρτήσεις είναι ψευδοτυχαία (pseudorandom) εξαιτίας της χρήσης της συνάρτησης rand() (stdlib.h) σε αντίθεση με τη διαδικασία παραγωγής τυχαίων μετρήσεων στους κόμβους όπου χρησιμοποιείται το αρχείο /dev/urandom. Το γεγόνος αυτό μπορεί να προκαλέσει παραγωγή ίδιων καταστάσεων μεταξύ διαδοχικών εποχών, δηλαδή αν με πιθανότητα 10% αλλάξει η συνάρτηση στην τωρινή εποχή τότε ενδεχομένως θα αλλάξει ξανά στην επόμενη.

3. Προσθήκες στον κώδικα

Σε περίπτωση επιλογής νέας συνάρτησης σε κάθε επόμενη εποχή είναι απαραίτητο να ενημερωθούν όλοι οι κόμβοι με το νέο flag. Αυτό είναι εφικτό εφόσον μεταδοθεί στο δίκτυο μήνυμα το οποίο περιέχει την πληροφορία αυτή. Η μετάδοση υλοποιείται με ένα σύστημα μετάδοσης λήψης παρόμοιο με αυτό του Routing. Η μετάδοση ξεκινάει από την ρίζα και συνεχίζει μέχρι τα φύλλα.

Κάθε κόμβος εφόσον επιτυχώς λάβει ένα νέο μήνυμα συνάθροισης αναλαμβάνει να το μεταδώσει στους διαδέχοντες κόμβους.

Ωστόσο επειδή η μετάδοση του μηνύματος της συναθροιστικής συνάρτησης γίνεται με παραλήπτη την διεύθυνση broadcast του δικτύου, κάθε κόμβος ενδέχεται να λάβει περισσότερα του ενός μηνύματος κάθε εποχή. Αυτό θα είχε ως αποτέλεσμα τα μηνύματα να παραμένουν στις ουρές χωρίς να εξυπηρετούνται και εν συνεχεία να εμποδίζουν την κυκλοφορία πληροφορίας επόμενων εποχών. Προκειμένου να αποφευχθεί αυτό το πρόβλημα κρίθηκε απαραίτητο να αδειάζονται οι ουρές στην αρχή κάθε εποχής:

```
event void RoundTimer.fired(){

    roundCounter++;

    if(TOS_NODE_ID==0){
        dbg("Rounds", "##########################################\n");
        dbg("Rounds", "                  ROUND   %u                \n", roundCounter);
        dbg("Rounds", "##########################################\n");

        shouldReaggregate = get10PercentTrue();

        if(shouldReaggregate){

            aggregateSelection = randReaggregate(aggregateSelection);
            post sendReaggregateTask();

        }

    }

    while(!call ReaggregateReceiveQueue.empty()){
        call ReaggregateReceiveQueue.dequeue();
    }
    while(!call ReaggregateSendQueue.empty()){
        call ReaggregateSendQueue.dequeue();
    }
}
```

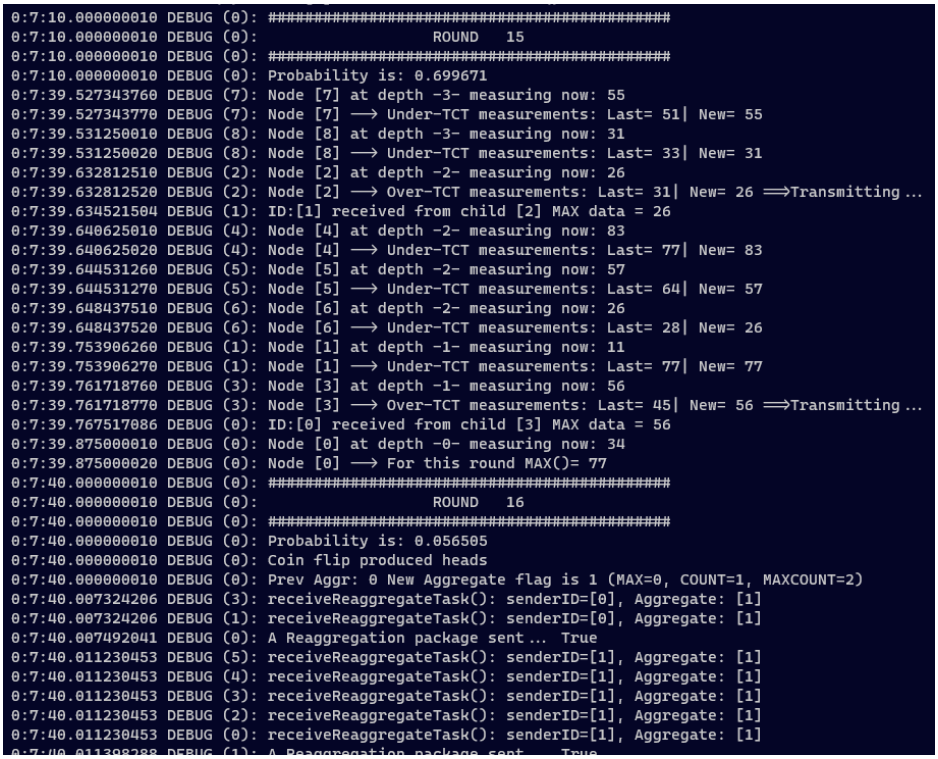
Η αποστολή και η λήψη των μηνυμάτων γίνονται από τα tasks **sendReaggregateTask()** και **receiveReaggregateTask()** αντίστοιχα. Ο τρόπος λειτουργίας και των δύο είναι παρόμοιος με αυτόν των routing tasks με μόνη διαφορά ότι κατά την αποστολή του μηνύματος η προετοιμασία και η ενσώματωση της πληροφορίας γίνονται μέσα στο ίδιο task ενώ κατά τη λήψη γίνεται κλήση της αποστολής με σκοπό την μετάδοση σε επόμενους κόμβους:

```
/*
    Get a reaggregate task & dequeue message
*/
task void receiveReaggregateTask()
{
    uint8_t len;
    message_t radioReaggregateRecPkt;

    radioReaggregateRecPkt= call ReaggregateReceiveQueue.dequeue();

    len= call ReaggregatePacket.payloadLength(&radioReaggregateRecPkt);

    dbg("ReaggregateRuntime","ReceiveReaggregateTask(): len=%u \n",len);

    if(len == sizeof(ReaggregateMsg))
    {
        ReaggregateMsg * mpkt = (ReaggregateMsg*) (call ReaggregatePacket.getPayload(&radioReaggregateRecPkt,len));
        uint8_t msource = call ReaggregateAMPacket.source(&radioReaggregateRecPkt);

        dbg("Reaggregate" , "receiveReaggregateTask(): senderID=[%d], Aggregate: [%d]\n", msource , mpkt->newAggregate);

        aggregateSelection = mpkt->newAggregate;

        post sendReaggregateTask();

    }
    else
    {
        dbg("ReaggregateRuntime","receiveRoutingTask():Empty message!!! \n");
        return;
    }
```

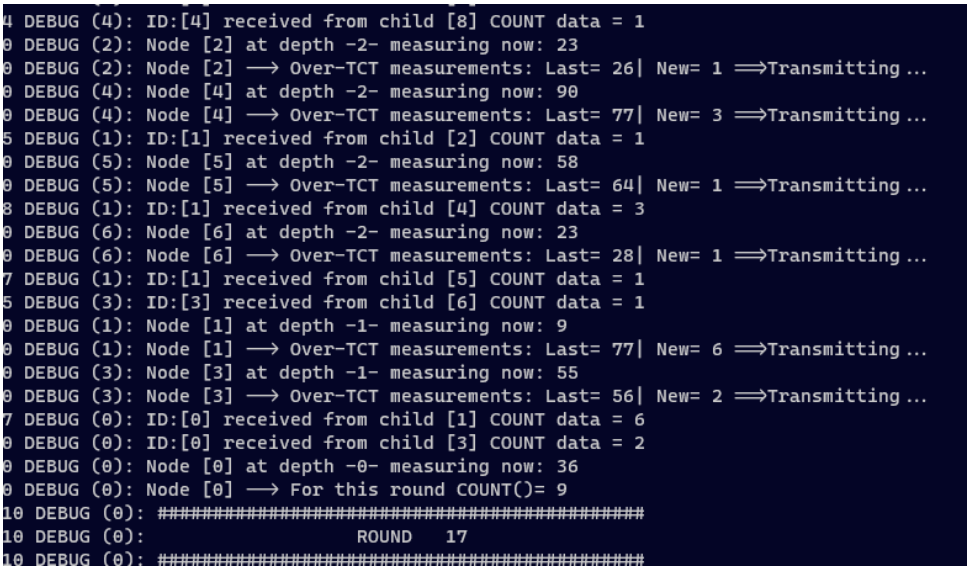
Συνοπτικά, υλοποιήθηκαν όλα τα απαραίτητα components για την δημιουργία ενός συστήματος αποστολής & λήψης μηνύματων, ουρές, αποστολείς, παραλήπτες, tasks και events. Η διαδικασία μετάδοσης μηνυμάτων συνάθροισης ξεκινάει από την ρίζα και περιέρχεται στην ευθύνη του εκάστοτε κόμβου.

4. Παράδειγμα εκτέλεσης σε πλέγμα 3x3

Στο παράδειγμα που ακολουθεί στον έως και τον γύρο 15 εκτελούνταν η συναθροιστική συνάρτηση MAX ενώ στον γύρο 16 τυχαίνει να πρέπει να αλλάξει σε COUNT:



Στο στιγμιότυπο παραπάνω αποτυπώνεται και μέρος της διαδικασίας μετάδοσης μηνυμάτων συνάθροισης στους κόμβους. Με την ολοκλήρωση του 16ου γύρου οι κόμβοι έχουν εκτελέσει την συνάρτηση COUNT:



5. Σύνοψη

Η ζητούμενη επέκταση στο δίκτυο δεν επέφερε αλλαγές στο δέντρο των κόμβων καθώς κάτι τέτοιο θα απαιτούσε να γίνει εκ νέου η διαδικασία του routing, γεγονός που θα απαιτούσε μεγαλύτερη κατανάλωση ενέργειας. Οποιαδήποτε αλλαγή στο routing δεν είναι απαραίτητη για την υλοποίηση της εν λόγω λειτουργίας. Αντ'αυτού εκτελείται μια διαδικασία παρόμοια με το routing χωρίς όμως αυτή να αλλάζει η δενδρική δομή με σκοπό την ελαχιστοποίηση του μεγέθους των μηνυμάτων. Σε κάθε περίπτωση δεν είναι δυνατή η ενημέρωση των κόμβων σε κάθε νέα εποχή με μηδενική κατανάλωση ενέργειας.

Σημειώνεται πως δεν κατέστη εφικτό να επιλυθεί το πρόβλημα του προγράμματος 2 σύμφωνα με το οποίο όταν εκτελούνται ταυτόχρονα οι συναρτήσεις MAX & COUNT δεν γίνεται σωστή μέτρηση του COUNT.