Netwrok Project Report

Xuan Luo, Siyi Yang, Haoran Wang 511309077

June 19, 2014

Abstract

In this report, we introduce two network protocols. The first one is the basic broadcasting protocol, which means no matter what packet a *sensor* receives it will broadcast it to neighbors. The second protocol supports routing and forwarding and will update the forwarding table in a fixed time. Besides, we explain the principle and motivation of second protocol prove the robust to temporary interruption. Finally we compare several versions of our protocol.

In other words, our basic method is by broadcasting everything. The extra functions are the second dynamic protocol.

1 Basic Protocol: Just Broadcast

As described in abstract, the first protocol we fulfilled is just broadcasting every packet it receive. We think that this protocol is not efficient enough and didn't keep the routing mechanism. As the idea is simple and not exciting, we will focus our attention on introducing the second protocol. The concrete fulfillment is listed in code.

The rest of report is all about second dynamic protocol.

2 Second Protocol: Dynamic Routing

The second protocol supports routing and forwarding between sensors. Besides that, it will adaptively change the routing table according to routing information of adjacent *sensors*.

3 Design & Main Algorithm

There're two main operations in the algorithm.

3.1 Data Structure On *Sensor*

Each *sensor* keeps two routing tuples. Each routing tuple contains three elements, including *Reachable*, *Next*, *Distance*. *Reachable* is whether this *sensor* is reachable to *sink*. *Distance* is the least path to *sink*. And *Next* is the next node in the least path to *sink*.

One tuple is *Current Tuple*, which is influenced by *Update* Operation, which will be introduced below. One tuple is *Previous Tuple*, which is exact the updated tuple before last *Reset*.

The reason we didn't record whole routing table is that, there is only one destination in this record, which means that we can just record one tuple, namely the information of the destination.

This design will save more storage space for embedded device.

4 Forwarding Algorithm

We use *Previous Tuple* in forwarding.

Once the sensor receive the data, it will determine whether the data is invalid. The data is invalid to a *sensor* if the data comes from *Next* of *sensor*. It *sensor* receives invalid data, just ignores it. Otherwise, if the data is valid, forwarding to *Next* of *Previous Tuple*.

The motivation for this judgement on invalid data is that the invalid phenomenon happens when the routing table is not correctly set. By ignoring these packets, we can save precious resources for bettering forwarding correct packet.

5 Routing Algorithm

5.1 Relevant Operation

inf = NODE_NUM, next = 0=sinkid

Three operations are designed in this algorithm.

Operation 1: *Reset* means set evaluate *Previous tuple* with the value of updated tuple. And set the *Current Tuple* initially, which means *Reachable* is false, *Distance* is infinite, *Next* is *null*.

reset … to …

Operation 2: *Update* means that once the *sensor* receives routing tuple from adjacent nodes, it will update its *Current Tuple*. The rule is to choose the least distance path.

choose to forward to …

Operation 3: *Broadcast* means that broadcast its *Previous Tuple*.

is to

specify the previous tuple of broadcast

5.2 Description of Algorithm

firstHop, len

lastFirstHop, lastLen

Rule 1:For a fixed time interval, every node, both *sink* and *sensor*, will do *Broadcast*.

Rule 2:Every *sensor* do *Reset* after every two times *Broadcast*.

Rule 3:Once a *sensor* receive a routing tuple, do *Update*.

(The motivation and principle is showed in the proof and next section.)

5.3 Proof Of Routing Algorithm

eventually

k hops away to

Once the position between nodes is relatively stable, then the routing table will be ultimately set right. *Lemma*: Once the position is stable, if every node in graph has done at least $k+1$ times *reset*, then all the *sensor* which are in k hops close the the sink will set the right routing table. And all the *sensor* that aren't reachable to *sink* within k hops has at least $k+1$ *Distance*. And these two properties still holds in the rest of algorithm. *Proof*: Clearly it's true for $k=0$. Assume it's true for $k-1$, then for *sensor* that is exactly k hops away from *sink*, after one more *reset*, it will set the right routing table, as all *sensor* with less than or equal $k-1$ distance are correct. For *sensor* that is out of k hops, the distance will be larger than k.

format

6 Problem & Solution

6.1 *Update* happens in the middle between *Reset* and *Broadcast*

It will cause the *sensor* broadcasts wrong routing information. The solution is to add *atomic* around *broadcasting* and *clearing*. In this way, we can make sure that the updated routing tuple does broadcast to other nodes.

reset?

6.2 *Sensor* doesn't receive all adjacent states

To guarantee the robustness of our protocol, we must make sure that the routing tuple the *Sensor Broadcast* is the tuple that has been *Updated* with every neighbor *Sensor*. If we set the time intervals between every two *Reset* and *Broadcast* exactly the same, we might lose some tuple in some interval and receive duplicate tuples in other interval. The solution is do *Broadcast* in two times speed compare to *Reset*. In this way, we can guarantee the assumption.

6.3 Poisson Reverse

The poisson reverse could not be handled in this protocol. Because *sensor* broadcasts routing information between a fixed interval. Although infinitive count might exist for some time during protocol, it will not influence the frequency of broadcasting and it will disappear once the position is stable.

不是stable的问题，而是是否能够到达sink的问题

summarize extra function