CSI5308: Programming Project

Topic: Distributed algorithms in dynamic networks

Sophie Le Page, 5992312

https://github.com/sofa13/DynamicNetworks

Introduction

This is a report about the *DynamicNetworks* program which was implemented to simulate algorithms in dynamic networks. This report explains the functionality of the program and how to run it, explains the algorithms implemented, and explains the results of running the simulations. The program and this report can be found at https://github.com/sofa13/DynamicNetworks.

Motivation

The motivation for this program is to simulate simple algorithms in order to better understand dynamic networks. The problem of dynamic networks has gained importance and popularity over the years with the growth of the Internet and mobile communication. Such networks change over times as nodes move around, and communication links appear and disappear. In some networks, e.g., peer-to-peer, the topology can change at a high rate, while in wireless ad-hoc networks, nodes are mobile and move around unpredictably.

Objective

Implement the flooding and routing algorithms in dynamic networks as discussed in [1], in which the network topology can change arbitrarily from round to round. Simulate the algorithms in different topologies and compare the message and time complexity results.

The Model

The graph G is allowed to be mobile in the sense that at all times G is connected. Message transmissions are asynchronous. Note the algorithms were simulated as synchronous for easy of understanding the simulation, however synchronicity was not used to solve the algorithm.

Along with the receipt of a message, nodes need to know about a local topology change as an event. For node v, if node w enters the neighborhood, node v will receive this as a neighborhood change and will immediately broadcast a message, then w is guaranteed to

receive it, since it is assumed the graph locally cannot change faster than it takes for a message to transmit. However, if node v receives a message at which point node w is in the neighborhood, when node v broadcasts the message, during transmission, node w might have moved, so node w might not receive the message.

DynamicNetworks program

Libraries

This program makes use of JBotSim, a tool for simulating distributed algorithms.
https://sites.google.com/site/gadiluna/home/distributedalgorithmsim

Getting started

Download the program and extract the files. Once the files have been extracted, go to the directory of DynamicNetworks program and compile each of the algorithms from the console.

```
C:\path> javac <Algorithm>.java
```

Example

```
C:\path> javac CounterFlooding.java

C:\path> javac IDFlooding.java

C:\path> javac MobileRouting.java
```

Input

To run the algorithms from the console, run the following line:

```
C:\path> java <Algorithm> <# of nodes> <topology type> <speed>
```

<Algorithm>: CounterFlooding, IDFlooding, MobileRouting
<# nodes>: must be greater than 1 and less than 200
<topology type>: thin, dense, or adversary
<speed>: fast, or slow

Example

```
C:\path> java CounterFlooding 25 thin slow

C:\path> java IDFlooding 50 dense fast
```

```
C:\path> java MobileRouting 30 adversary fast
```

Output

When the program is run there will be a GUI of the graph simulation, as well as output to the console. In addition, during the simulation, when all nodes have received the message, an output file is generated with a breakdown of the # messages and # time it took to achieve correctness.

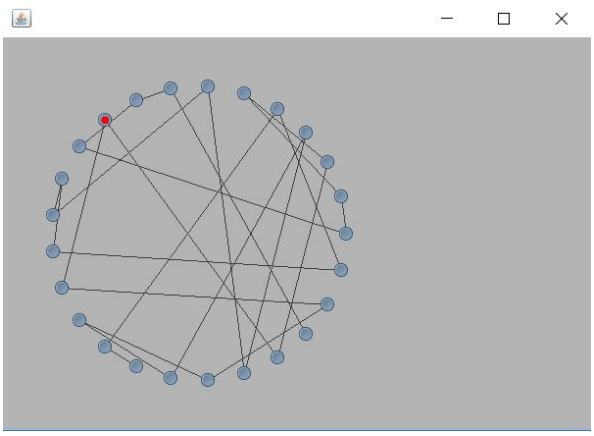
GUI

The nodes are either:

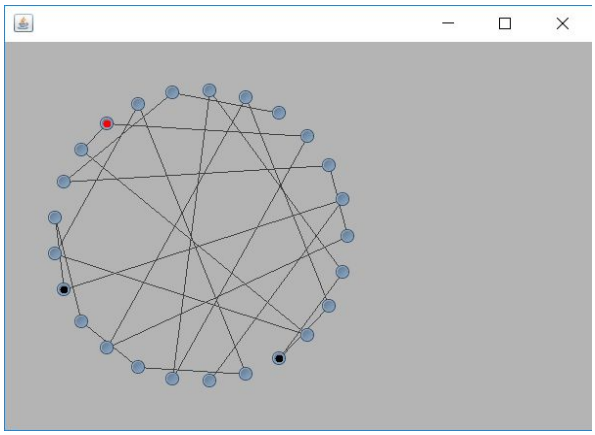
- PASSIVE (**grey**): Waiting to receive a message.
- INITIATOR (**red**): There is only one initiator at the beginning of the algorithm and by default it is always the node with ID 0. This node sends the message to all neighbors. The default message is the initiator's ID. When sending a message to a destination ID, the default destination is the maximum ID. Note when the initiator node receives a message, the node becomes received.
- RECEIVED (**black**): Receives a message and broadcasts the message to all neighbors minus the sender. A node stays in state received until it's counter k (number of neighborhood changes) is less than 2*n, where n is an upper bound on the number of nodes.
- DONE (**white**): Node will no longer send any more messages (number of neighborhood changes is greater than 2*n, where n is an upper bound on the number of nodes).

Example

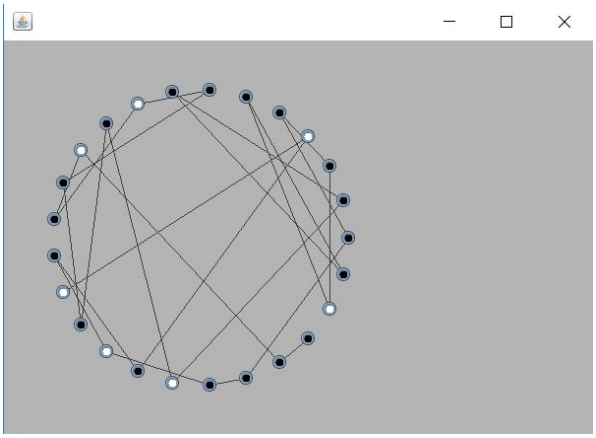
1



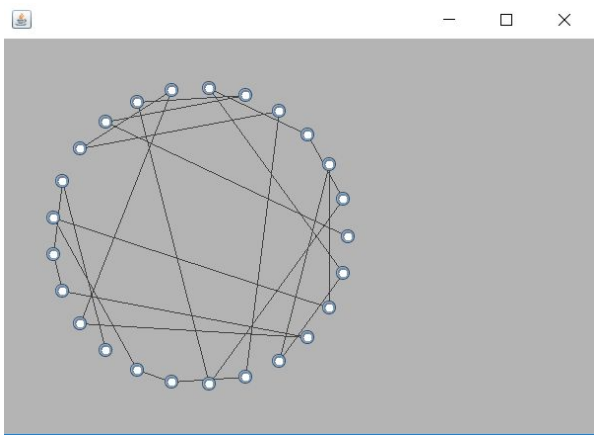
2



3



4



Console

Everytime a message is received by a node, the node prints the message received, as well as the node's current upper bound on n and it's count k. Notice print statements stop printing to the console when nodes are no longer receiving messages (when all nodes are in state DONE).

My ID: <# MY ID> **RCVD:** <#ID RCVD FROM> **->** <# MY ID>**:** <MESSAGE> **n:** <#n> **k:** <#k>

Example

Node 16 receives a message from Node 1

CounterFlooding

message is "0":
My ID: 16 RCVD: 1 -> 16: 0 n: 22 k: 4

IDFlooding

message is [msg="0", upper bound on n]:
My ID: 16 RCVD: 1 -> 16: [0, 23] n: 22 k: 4

MobileRouting

message is [msg="0", destination ID=max ID, upper bound on n]:
My ID: 16 RCVD: 1 -> 16: [0, 24, 23] n: 22 k: 4

Output file

When all nodes have received the message, a file titled "<Algorithm>Correctness.txt" is outputted in the same directory as the program, with the following information reported by the last node ID to have received the message:

Example

*** ALL RECEIVED, <ID> ***
*** NUMBER MESSAGES RECEIVED BY EACH ID ***
....
*** TOTAL NUMBER OF MESSAGES ***
...
*** TOTAL TIME ***
..

Algorithms Implemented

Algorithms

CounterFlooding

This algorithm has the knowledge of n, and keeps a counter k of the number of times it has broadcasted a message. Initially when a node receives a message, the node broadcasts the message and begins keeping count. The node then only continues broadcasting the message upon detecting a neighborhood change. When k >= 2*n, a node stops sending messages. Thus a node stops sending messages after 2*n neighborhood changes. A proof shows that this condition guarantees correctness and termination.

IDFlooding

This algorithm does not have knowledge of n, but requires nodes to have unique identifiers. In this way the algorithm tries to estimate an upper bound of the nodes in the graph based on the maximum ID a node receives a message from. Thus a node must remember a constant number of other node's IDs. This algorithm makes use of CounterFlooding to guarantee correctness and termination. Every time the number of node estimate increases (when a new maximum ID is discovered), CounterFlooding is called with a new estimate of n. Once a node has received the maximum node ID in the graph, the node will stop sending messages.

Mobile Routing

This algorithm does not have knowledge of n. This algorithm requires nodes to have unique identifiers, since the destination ID is used to check locally whether it is the destination or not. Again, the algorithm estimates an upper bound on n and uses CounterFlooding to guarantee correctness and termination. In this case, we want to estimate the number of nodes without the use of IDs, where instead the destination node acknowledges the receipt of the message and initiates a termination phase, making use of the upper bound of the message counter. There are three states, INIT, FLOOD, and TERM. In the INIT state n is incremented, in FLOOD state n is

used as an upper bound. Once new upper bound > n, then node moves to TERM state and executes CounterFlooding algorithm which will guarantee termination and correctness.

Complexity

For flooding algorithms, correctness means that all nodes must be reached. For routing algorithms, correctness means that the destination must be reached. Termination condition is defined as the eventual point when no node will transmit any more messages. The following table compares the correctness complexity for each of the algorithms.

Algorithm	**Condition**	**Correctness Complexity**	
		Time	**Memory**
CounterFlooding	Knowledge of n.	O(n)	O(1)
IDFlooding	Each node has a unique identifier.	O(n^2)	O(logn)
MobileRouting	Initiator knows the destination ID.	O(n)	O(logn)

For message complexity, while correctness is achieved in time O(n) or O(n^2), it might take a long time before the algorithm actually terminates at all nodes. Termination complexity was not analyzed in the paper [1].

Experiment

Simulation

In this section the three implement algorithms, Counter Flooding, ID Flooding, and Mobile Routing, were simulated to derive some experimental results. The first part of the experiment compares running the simulation in a thin graph (few edges), and the second part compares running the simulation in a dense graph (many edges). In the third part of the experiment we consider a worst-case model in which the communication links for each round are chosen by an adversary. In this case the adversary topology is a line, in which the changes in the dynamic network do not help propagate the message and only increases the message complexity. Note when the graph changes dynamically, the graph is still connected at all times.

The simulation is run with number of nodes = 25, averaging three runs.

Table 1: Algorithm simulation results, average of three runs, number of nodes = 25

	CounterFlooding		IDFlooding		Routing	
Runs	**Message**	**Time**	**Message**	**Time**	**Message**	**Time**
Thin						
1	98	4	103	4	53	3
2	92	4	85	5	12	3
3	86	4	103	4	28	3
Avg	92	4	97	4.33	31	3
Dense						
1	279	2	461	3	63	2
2	286	2	348	1	39	1
3	308	1	691	2	108	2
Avg	291	1.67	500	2	70	1.67
Adversary						
1	254	24	254	24	254	24
2	254	24	254	24	254	24
3	254	24	254	24	254	24
Avg	254	24	254	24	254	24

As expected with broadcasting algorithms, in the thin topology there are less messages being sent, whereas in dense topologies there are many more messages being sent, although correctness is achieved in about the same amount of time for both topologies. The adversary topology which was designed as a worst case scenario ended up performing the worst especially for the amount of time it took to achieve correctness. The adversary topology time complexity is comparable to broadcast algorithm in static graph where the topology is a line.

Conclusions

In this project, a comparison between three dynamic network algorithms was conducted. In this comparison, we chose an algorithm for when the number of nodes is known (CounterFlooding), an algorithm when the number of nodes is not known (IDFlooding), and a routing algorithm (MobileRouting). The conducted experiments showed that CounterFlooding exchanges less messages and takes less time to achieve correctness, since an upper bound on n is already known. However sometimes an upper bound on n is not known, in which case IDFlooding does not perform much worse than CounterFlooding to achieve correctness. Mobile Routing algorithm is implemented in a way to makes use of the destination ID, so as to achieve correctness with less messages and less time than CounterFlooding and IDFlooding. In

conclusion, the simulations show that for all the algorithms, correctness and termination are achieved in finite amount of time.

References

1. O'Dell, Regina, and Rogert Wattenhofer. "Information dissemination in highly dynamic graphs." Proceedings of the 2005 joint workshop on Foundations of mobile computing. ACM, 2005.