TOKEN RING NETWORK USING LGI

Niraj Dholakia (nd387)

Saurabh Deochake (srd117)

Romina Nayak (rn279)

04/27/2016

CS 547 Security & Dependability of Distributed Systems

Prof. Naftaly H. Minsky

RUTGERS UNIVERSITY, New Brunswick

TABLE OF CONTENTS

ABSTRACT

In a Local Area Network (LAN), a number of computers are interconnected. These systems might need to constantly communicate with each other and with servers. But, if multiple systems start communicating at the same time then, it would lead to congestion in the network. To overcome this problem, we present and implement a solution – Token Ring Protocol (TRP) – which is essentially a communications protocol to eliminate collisions of contention-based access methods [2]. Our TRP uses Law Governed Interaction (LGI) to maintain the structure of the network of systems in the form of a ring and ensuring the existence of a single token in the network [1].

Keywords: Token Ring Protocol, TRP, Law Governed Interaction, LGI, Moses

TOKEN RING NETWORK USING LGI

In a Local Area Network (LAN), a number of computers are interconnected. These systems might need to constantly communicate with each other and with servers. So, the systems need to be arranged in an orderly structure otherwise, simultaneous communications can cause congestion in the network. To overcome this problem, we present and implement Token Ring Protocol (TRP) to our network. A Token Ring Protocol is a communications protocol used for eliminating collisions in communications between interconnected systems in a network like LAN [2]. For implementing this protocol, the systems in the network are arranged in a logical ring such that each system knows only about the next system in the ring. Then, a token, which might be a 3-byte frame [2] or a simple message with a title denoting it's a token, is passed along all the systems in the logical ring in a particular direction (clockwise or anti-clockwise). Whenever a system gets the token, it can send messages across the network; otherwise it needs to wait until it receives the token.

Law Governed Interaction introduces a Law, which is a set of rules, between a group of distributed heterogeneous agents to interact with each other with confidence that each of the agent will comply with the Law [3]. In a Token-Ring protocol, LGI helps to ensure that the token is passed between all the members of the ring in an orderly way and within a reasonable time. The Law also provides fault tolerance to our network in case of a token loss or agent loss.

INITIAL IMPLEMENTATION

a) SETUP

User Roles:

An agent in our Token ring network can have one of the following roles:

i) Manager – An agent system which monitors the ring members and has special conditions in the Law.

ii) Server – An agent system which passively records messages from ring members possessing the token.

iii) Ring member – An agent system which wants to communicate with the Server and so adopts the Law to become a part of the ring. It also gives updates to the Manager.

Policies:

Our Law implements the following policies:

i) Only one token exists in the system at all times.

ii) Only the holder of the token is allowed to communicate and send messages.

iii) The token travels in a pre-defined direction through the ring.

iv) An agent can possess the token only upto a maximum period of time.

Assumptions:

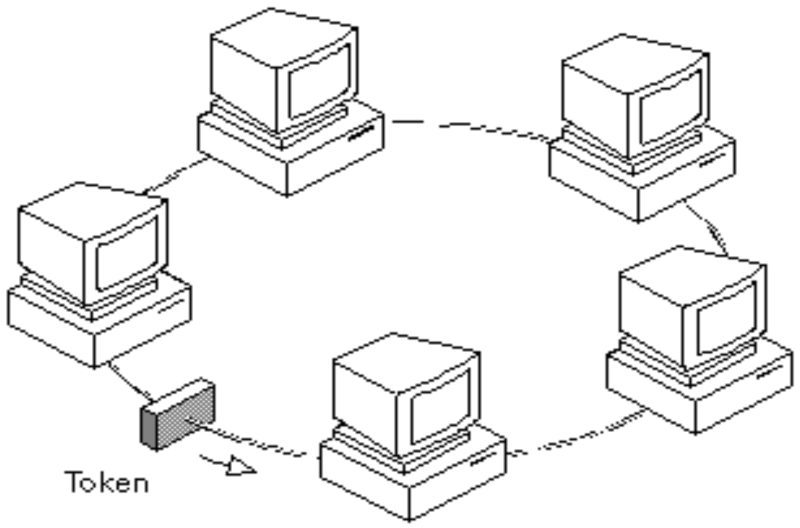
In addition to the policies, we have assumed:

i) The Manager never dies/fails.

ii) Any agent who adopts our Law automatically becomes a member of the ring.

iii) The Manager and Server are pre-defined in the Law and they do not change.

b) IMPLEMENTATION



In our initial prototype, we created a static ring of three agents – a, b and c. As soon as each of the agents get adopted, their "*next*" field gets updated with another agent. For example, if agent 'a' gets adopted, then our law checks if the agent is 'a' and if yes, it assigns agent 'b' in a's *next* field. If the agent adopted is 'b', then it's *next* field stores agent c and if agent 'c' gets adopted, then it's *next* field points to agent a. Each agent knows only it's successor agent and not it's predecessor agent. Hence, a basic network with a circular one-directional structure is programmed.

Another system acts as a Manager, which has some additional responsibilities (obligations & actions) over an agent. A token, that decides which system gets a chance to send messages in the network, is introduced by the Manager.

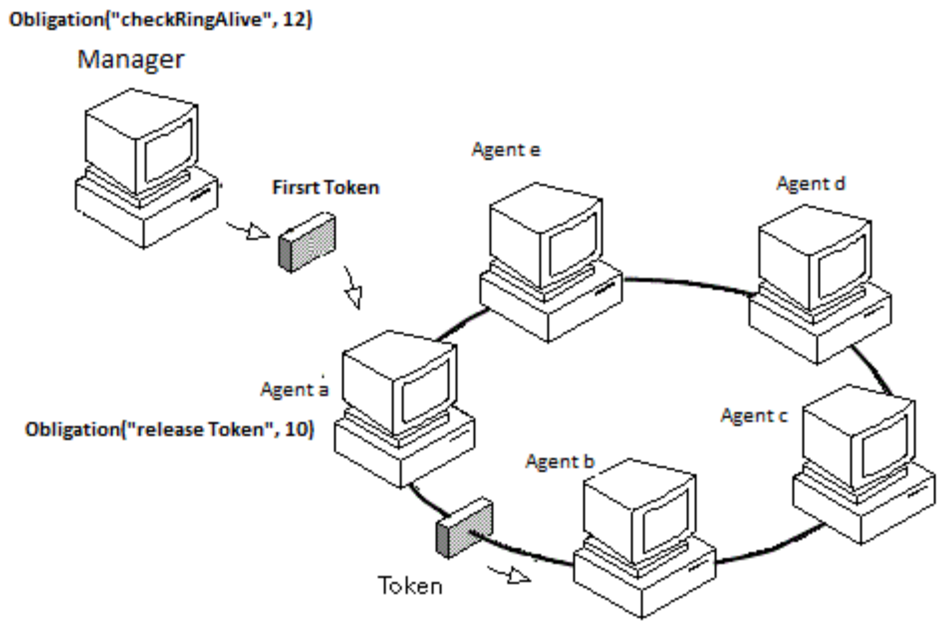
FAIRNESS AND LIVENESS

Fairness of the system means that every agent should get a chance to send messages in the network i.e. every agent should get the token and *liveness* means that the next agent should receive the token within a reasonable time. We ensure that our system is live by imposing *obligations* (written in the common Law) on the agents. These obligations are certain restrictions which enforce a modification in the behavior of the agents. To maintain liveness in our system, the agent with token is obliged to release and pass the token to the next agent after 10 seconds. So, we say that *obligation is due* after every 10 seconds which maintains liveness in the system.

FAULT TOLERANCE

There are two major situations that our system can encounter – token loss and agent loss. We have incorporated mechanisms to handle both of these problems:

a) TOKEN LOSS


Obligation("checkRingAlive", 12)
Manager
Firsrt Token
Agent e
Agent d
Agent a
Obligation("release Token", 10)
Agent c
Agent b
Token

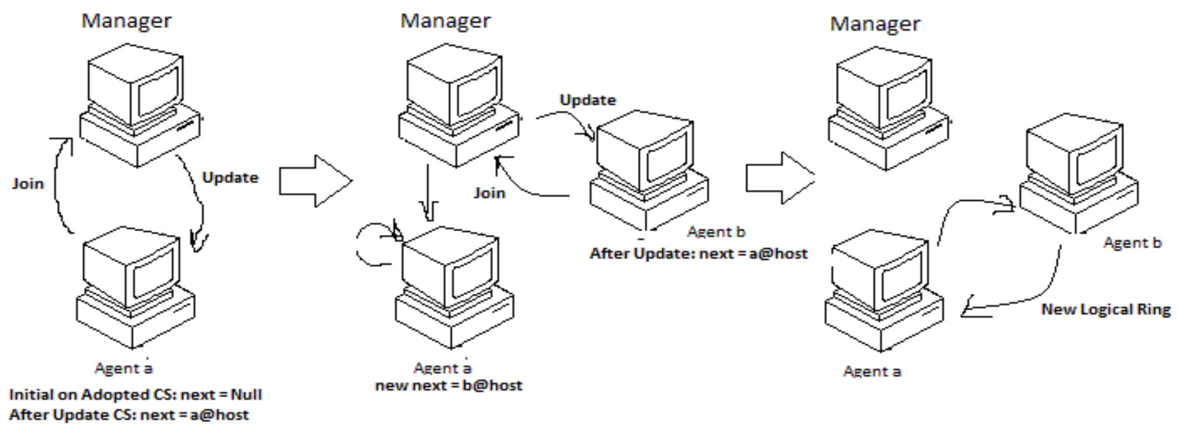
Logical Ring formed between Agents

In our Token-ring network, there is an obligation *releaseToken* on the agents to release token in 10 seconds and inform the Manager about it by sending appropriate messages. Another obligation *checkRingAlive* is written in the Law which enforces the Manager to keep a check on the Token passing. If the Manager doesn't receive a Token release message within 12 seconds, then it assumes the token has been lost. When this happens, the Manager introduces a new token in the ring network. The new token is assigned to the first agent in the network and passing of the token can continue again.

b) AGENT LOSS

When an agent dies, a *disconnected* event is fired in the form of a message to the Manager. As soon as the Manager receives this message, it updates the list of the members it maintains. The Manager locates the position of the disconnected agent ad removes this agent from its list. Along with the removal, it updates the *next* pointer of the previous member in the list with the *next* value of the disconnected agent. Hence, the ring is connected and resumes again.

INCREMENTAL MODEL



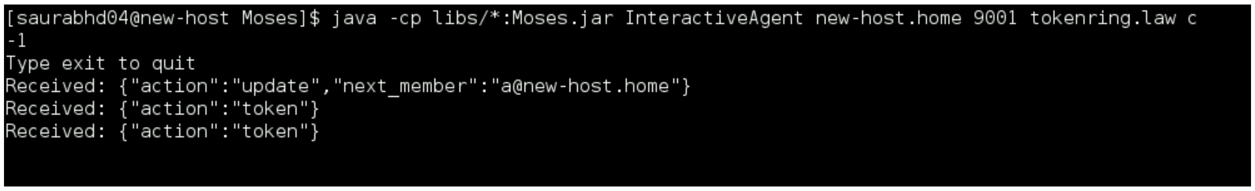
Incremental Ring Model

Once our token passing was up and running for our static ring of agents, we started incorporating incremental approach for adding of new agents to the network. For this, we created a common list of members. Whenever a new agent wants to join the network, it sends a *join* message to the Manager. To approve the addition, the Manager updates its list of members by appending this new member to its common list. Also, the Manager sends an *Update* message to the new agent with the name of the *next* member in the ring. Once, the new agent updates its *next* pointer, it has been added to the ring network.

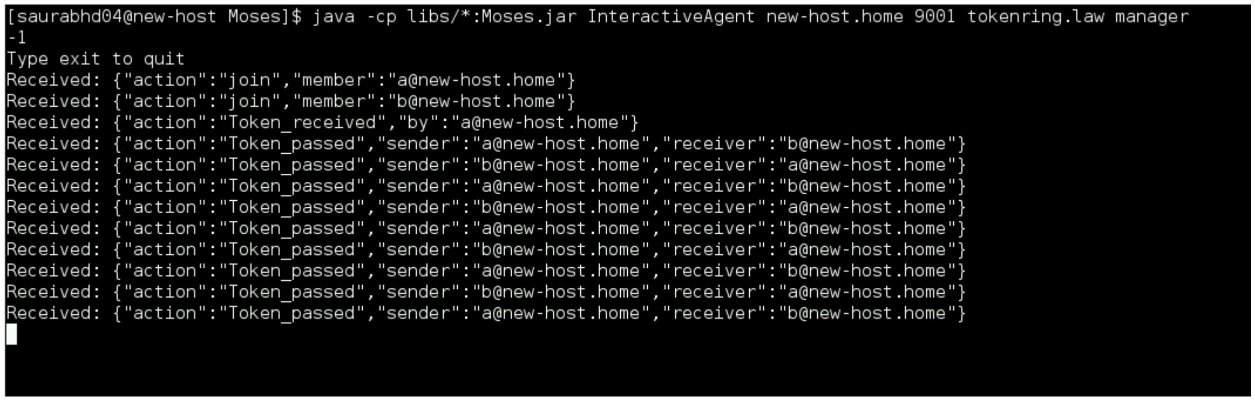
FINAL IMPLEMENTATION

To implement the system, we have used Moses [4] and written a Law *tokenring.law* in JavaScript. The following screenshots show the final walkthrough of our implementation.

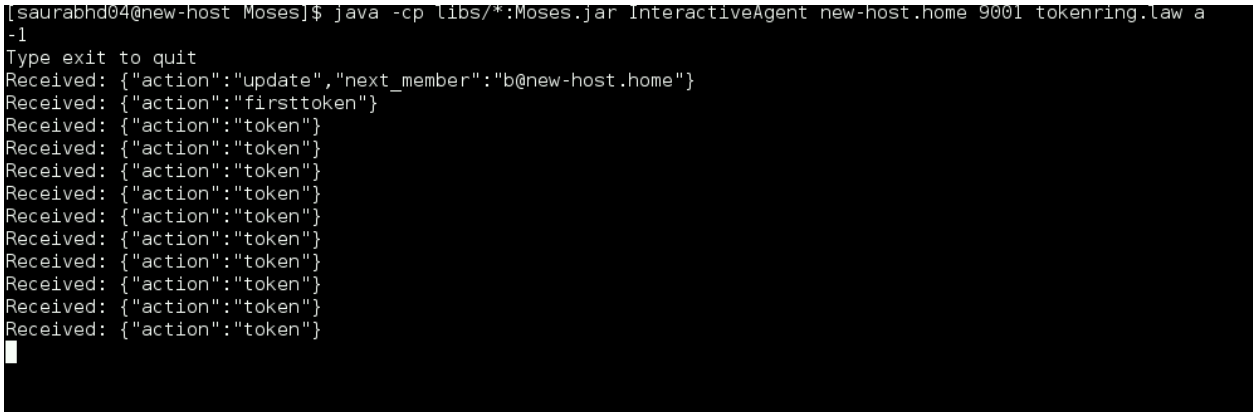
Moses Console [4]:


```
[saurabhd04@new-host Moses]$ java -cp libs/*:Moses.jar InteractiveAgent new-host.home 9001 tokenring.law c
-1
Type exit to quit
Received: {"action":"update","next_member":"a@new-host.home"}
Received: {"action":"token"}
Received: {"action":"token"}
```

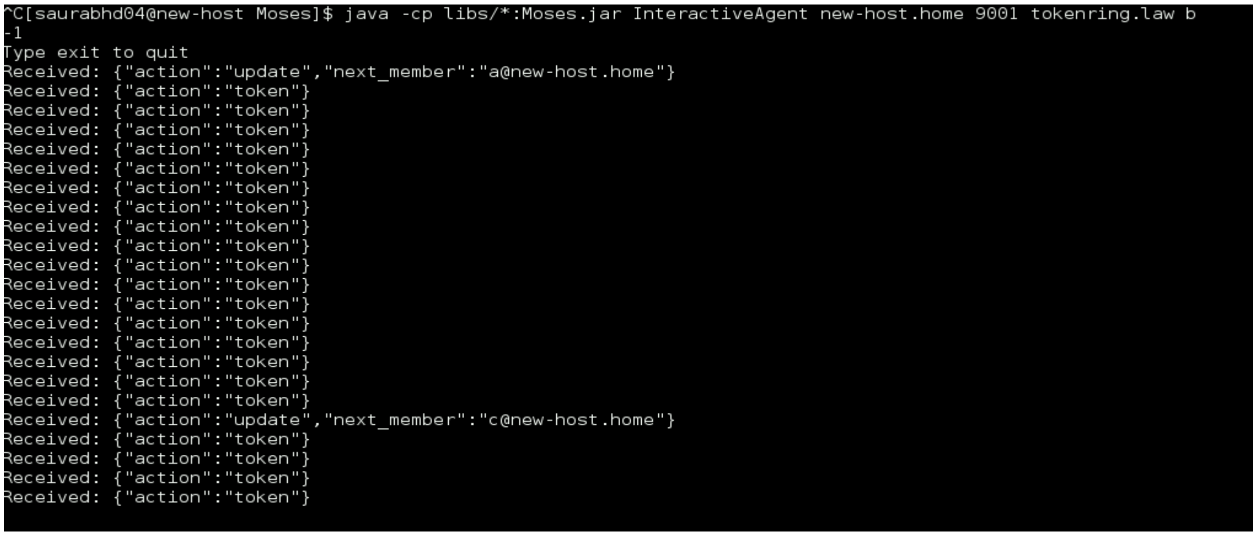
Manager Console:


```
[saurabhd04@new-host Moses]$ java -cp libs/*:Moses.jar InteractiveAgent new-host.home 9001 tokenring.law manager
-1
Type exit to quit
Received: {"action":"join","member":"a@new-host.home"}
Received: {"action":"join","member":"b@new-host.home"}
Received: {"action":"Token_received","by":"a@new-host.home"}
Received: {"action":"Token_passed","sender":"a@new-host.home","receiver":"b@new-host.home"}
Received: {"action":"Token_passed","sender":"b@new-host.home","receiver":"a@new-host.home"}
Received: {"action":"Token_passed","sender":"a@new-host.home","receiver":"b@new-host.home"}
Received: {"action":"Token_passed","sender":"b@new-host.home","receiver":"a@new-host.home"}
Received: {"action":"Token_passed","sender":"a@new-host.home","receiver":"b@new-host.home"}
Received: {"action":"Token_passed","sender":"b@new-host.home","receiver":"a@new-host.home"}
Received: {"action":"Token_passed","sender":"a@new-host.home","receiver":"b@new-host.home"}
Received: {"action":"Token_passed","sender":"b@new-host.home","receiver":"a@new-host.home"}
Received: {"action":"Token_passed","sender":"a@new-host.home","receiver":"b@new-host.home"}
```

Agent a Console:


```
[saurabhd04@new-host Moses]$ java -cp libs/*:Moses.jar InteractiveAgent new-host.home 9001 tokenring.law a
-1
Type exit to quit
Received: {"action":"update","next_member":"b@new-host.home"}
Received: {"action":"firsttoken"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
```

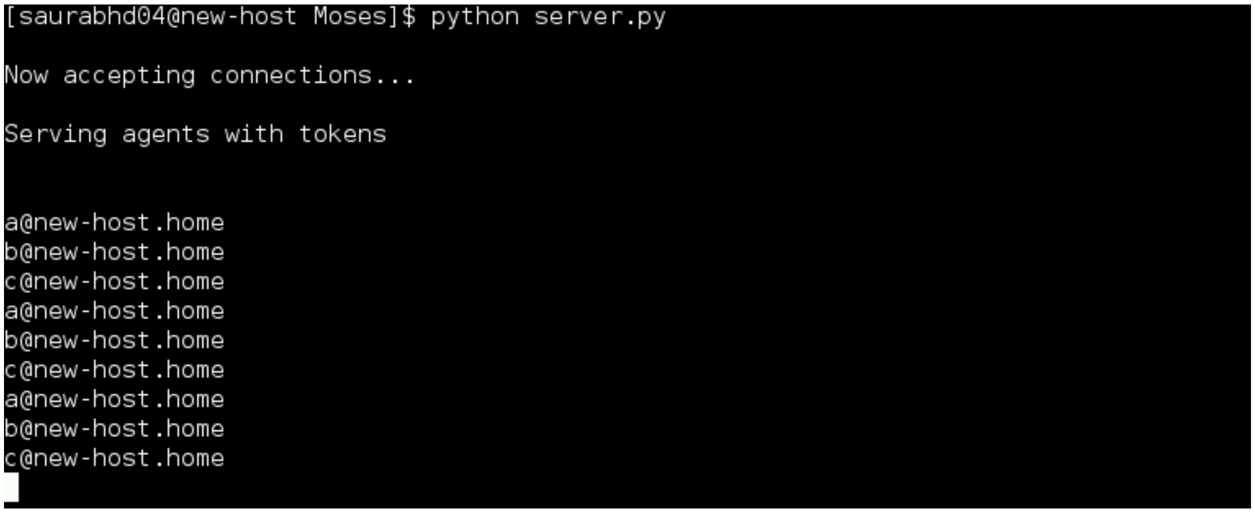
Agent b Console:



```
^C[saurabhd04@new-host Moses]$ java -cp libs/*:Moses.jar InteractiveAgent new-host.home 9001 tokenring.law b
-1
Type exit to quit
Received: {"action":"update","next_member":"a@new-host.home"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"update","next_member":"c@new-host.home"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
Received: {"action":"token"}
```

VISUALISATION

We have created a python script which displays the token possession through the Server console. So every time the token is passed, the name of the agent possessing it appears at the Server console (as depicted below).



```
[saurabhd04@new-host Moses]$ python server.py

Now accepting connections...

Serving agents with tokens


a@new-host.home
b@new-host.home
c@new-host.home
a@new-host.home
b@new-host.home
c@new-host.home
a@new-host.home
b@new-host.home
c@new-host.home
```

GITHUB LINK

All the project related documents have been uploaded to Github and it can be accessed at: https://github.com/rominanayak/lgi_token_ring

FUTURE WORK

Future work can include making the system more secure by introducing Certificates for new member entry. So, whenever a new agent wants to join the token-ring network, it needs to provide a Certificate to the Manager validating its authenticity. If the Certificate is approved, the agent is allowed to join the network.

Currently, our system has a certain part centralized in the form of a Manager. So, we would like to plan and re-structure the network in a way that the Manager is not required, and hence the system will be truly decentralized.

We have explained visualization of token passing in our token-ring network above. But, we would like to take it further and create a dynamic graphical representation of the token-ring network and token passing in it.

ACKNOWLEDGMENTS

The authors would sincerely like to thank Prof. Naftaly H. Minsky and Chen Cong for their valuable input throughout the semester and bringing this project to fruition.

REFERENCES

[1] Minsky N., Law-Governed Internet Communities
http://www.cs.rutgers.edu/~minsky/papers/community.pdf

[2] Wikipedia, https://en.wikipedia.org/wiki/Token_ring

[3] LGI at Rutgers, http://www.cs.rutgers.edu/lgi/

[4] Moses at Rutgers, http://www.moses.rutgers.edu/