

DISTRIBUTED CONWAY'S GAME OF LIFE SIMULATION

CMPT431 Final Presentation

Background Introduction

- The Game of Life is a visualized cellular automaton devised by the British mathematician John Horton Conway in 1970.
- The universe (board) of the Game of Life is a two-dimensional orthogonal grid of square cells.
- Each cell has three states:
 - Survival
 - Death
 - New Birth



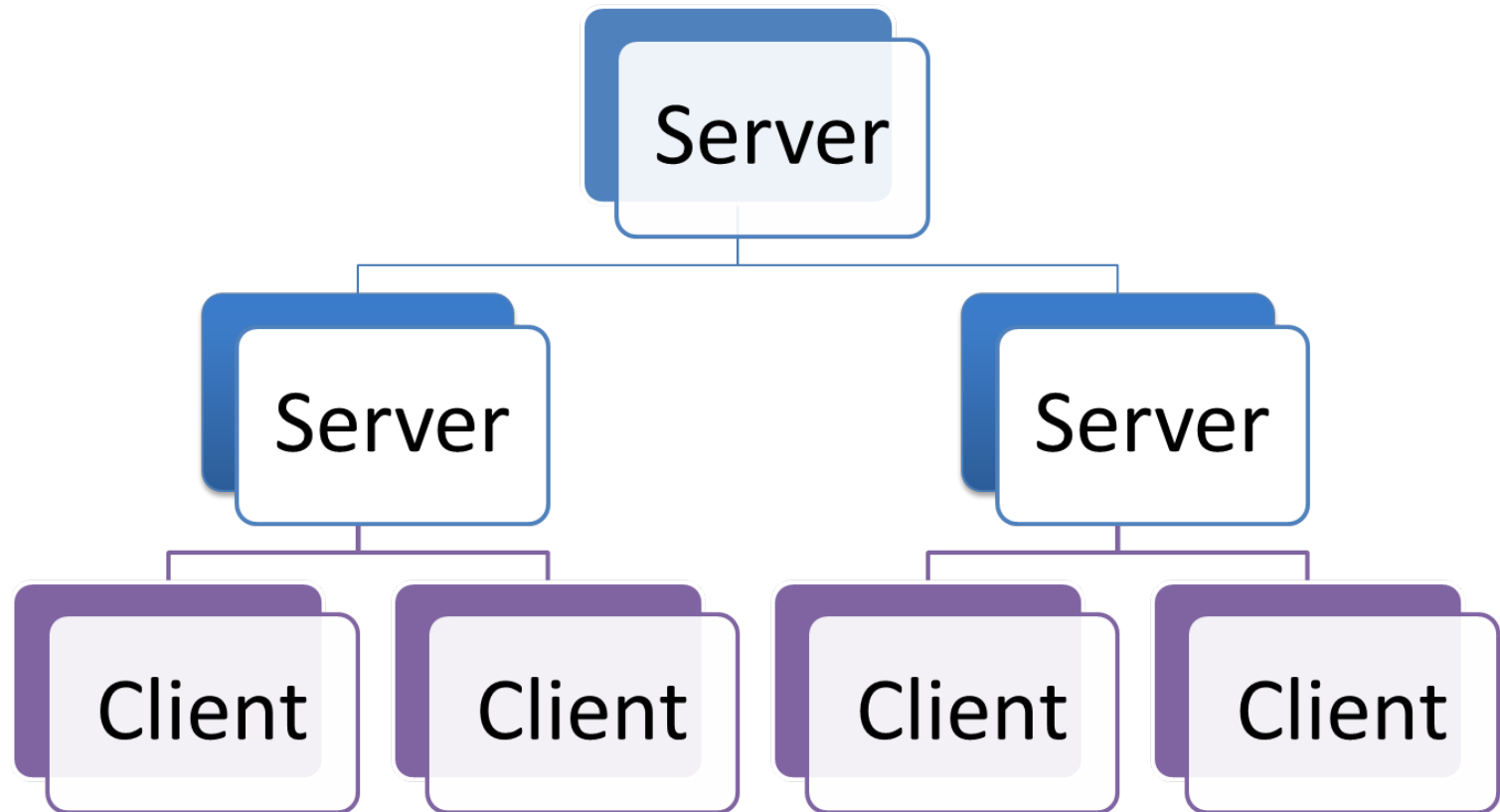
Objectives

- Implement a distributed Conway's Game of Life computational simulation
- Handle large-size packetized network messages
- Build a graphical interface to visualize the universe
- Support over 100,000,000 cells and 10,000,000 lives
- Support dynamically adding and deleting computing nodes
- Finish one cycle with 100,000,000 cells within 0.25 sec

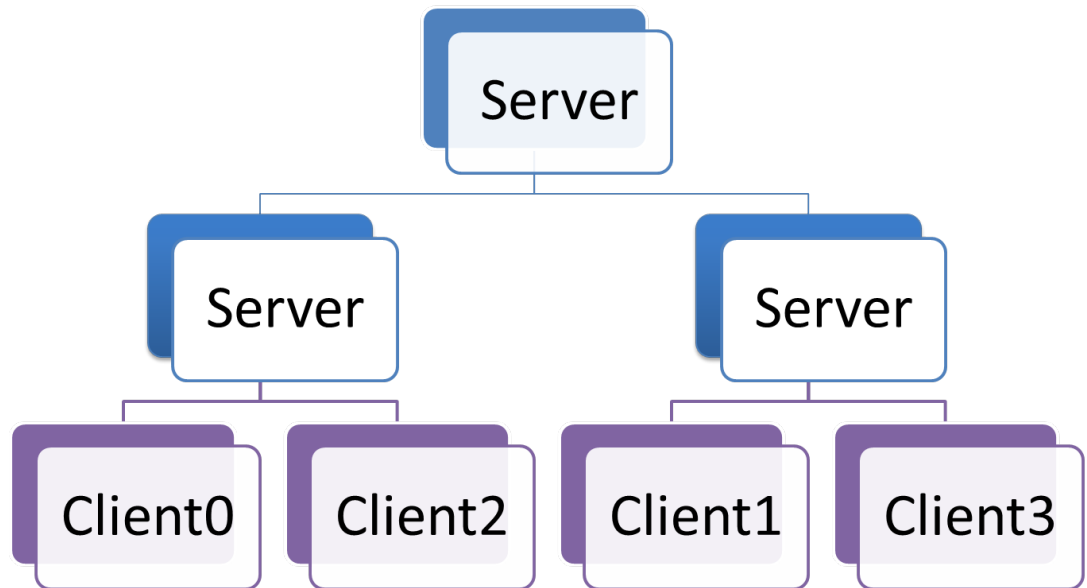
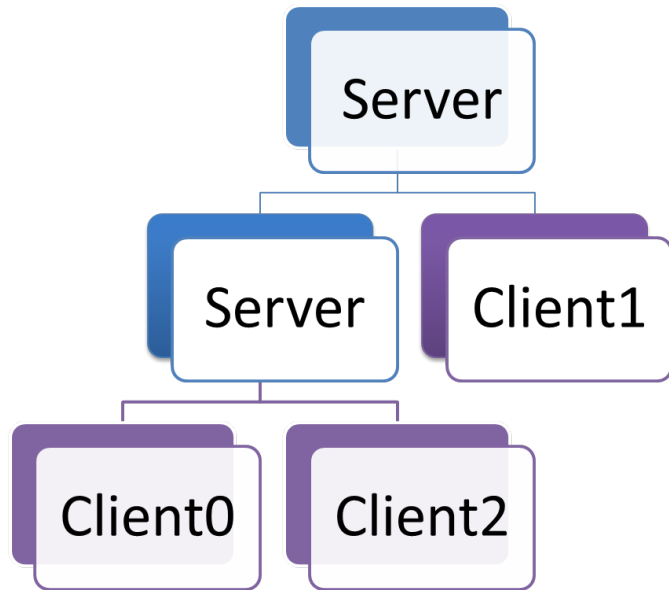
Challenges

- Computing nodes must exchange information to coordinate with each other
- Various messages must be designed to help implement dynamically adding and deleting nodes
- Server must take control to the computing nodes to guarantee transitions occur simultaneously
- Network messages should be minimized in order to avoid the system becomes network-bound with only a few computing nodes

Structures



Adding



Adding (cont.)

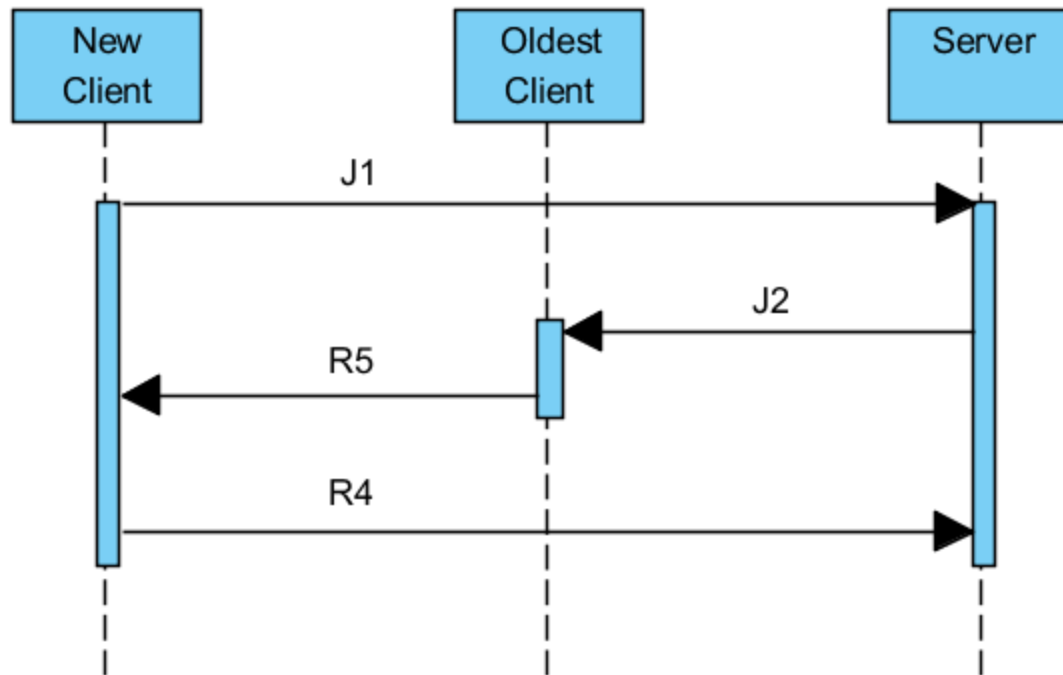
- A wants to join (first client)
 - A => Server: application
 - Server => A: accepted & data
 - A => Server: received

Adding (cont.)

- A wants to join (general)
 - A => Server: application
 - Server => B (*): ask B to work with A
 - B => A: accepted & data
 - A => B: received
 - B => Server: everything is done

*B is the first node who has no siblings; if everybody has, then choose first node.

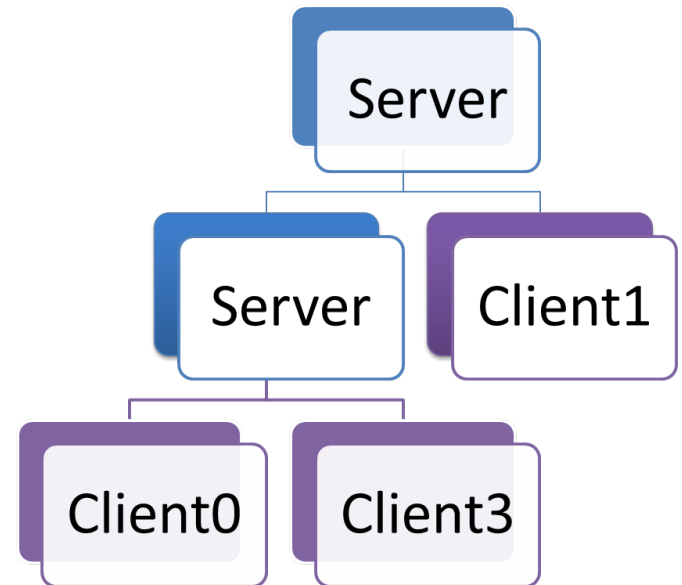
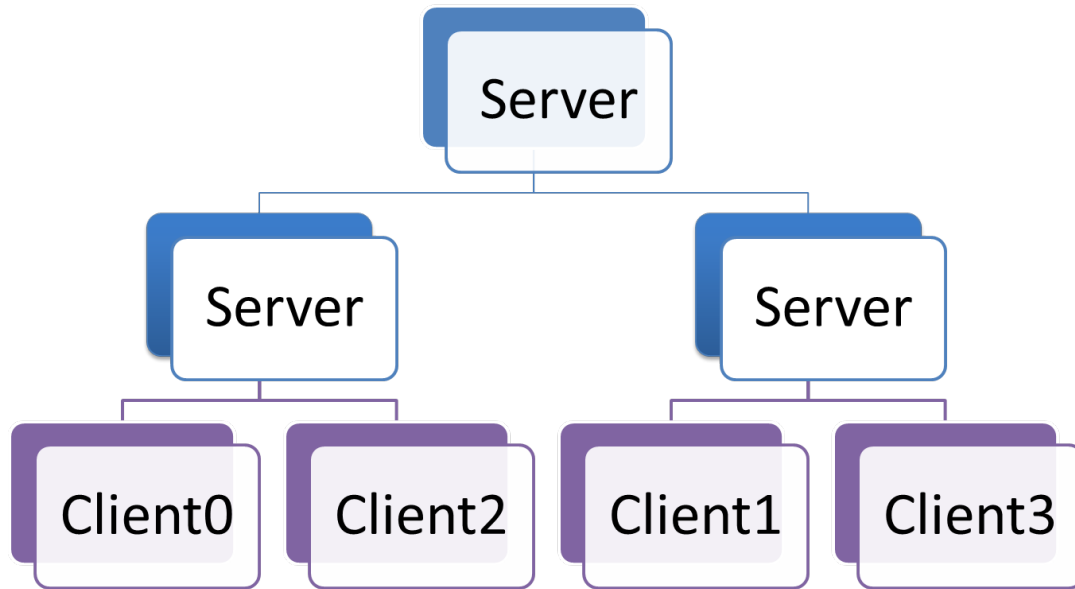
Adding (cont.)



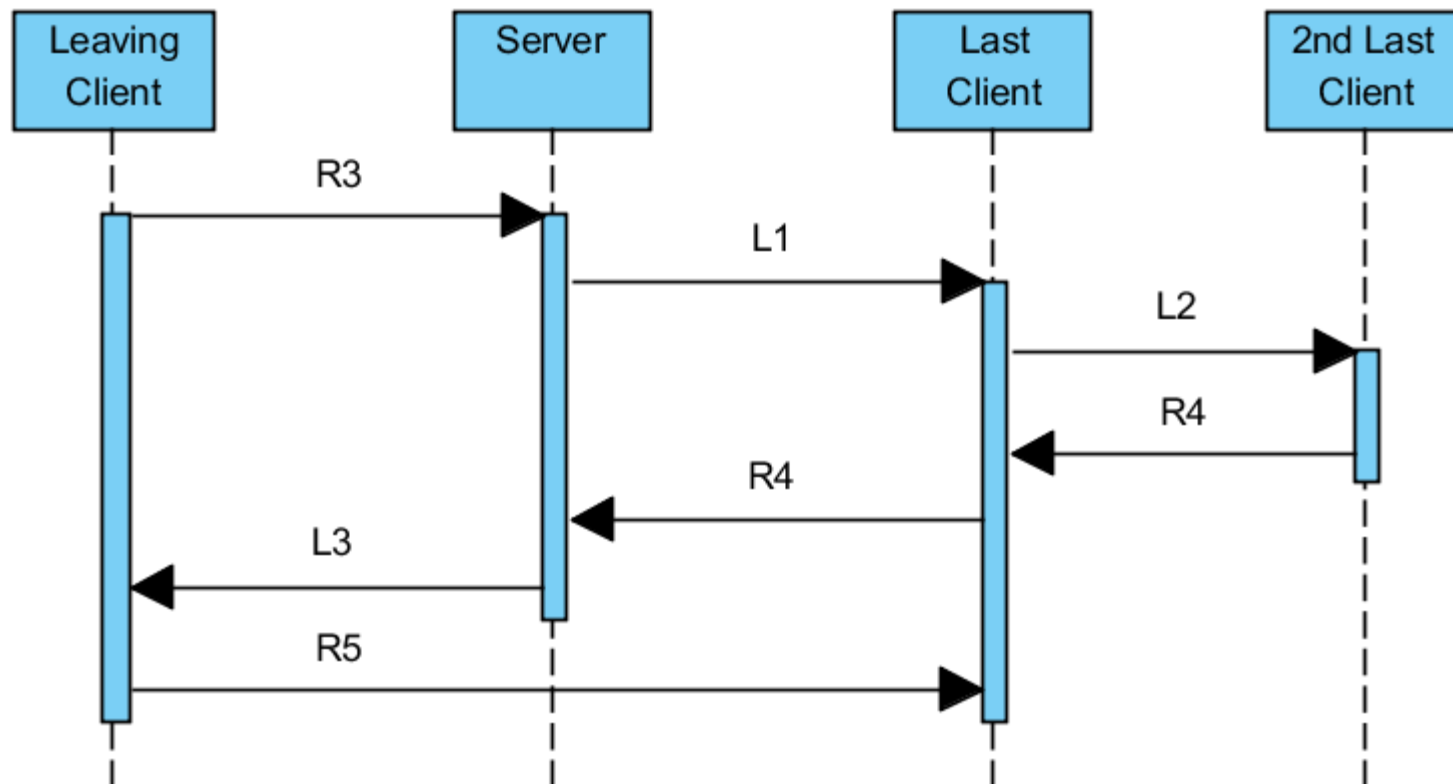
Message Passing

Message	Direction	Payload	Description
R1 Next Clock	S→C	new clock number	Inform all the clients to start new computation round
R2 Border	C→C	border status vector	Send border information to clients that is in charge of adjacent area
R3 Report	C→S	whether is leaving, top and left coordinate, status bit map	Report finished computation, and return the computation result to the server in interactive mode
R4 Confirm	A→A	none	Confirm received last message
R5 New Outfit	C→C	c o n n e c t i o n information, outfit information	Send all the computing information to a new added client
R6 Neighbor Update	C→C	c o n n e c t i o n information, relative position	Inform all the neighbors about the position change
J1 Request Join	C→S	connection information	Send request to server for joining the computation system
J2 Split	S→C	c o n n e c t i o n information, new client id, split direction	Inform the right client to split its area, construct the new outfit and send to the new client
L1 Last Merge	S→C	connection information	Inform the last two client to merge into one, the idle one waits for new command
L2 Outfit Merge	C→C	c o n n e c t i o n information, outfit	Send outfit to pair client for merging
L3 Leave Permit	S→C	connection information	Send outfit to the specified client

Leaving



Leaving (cont.)

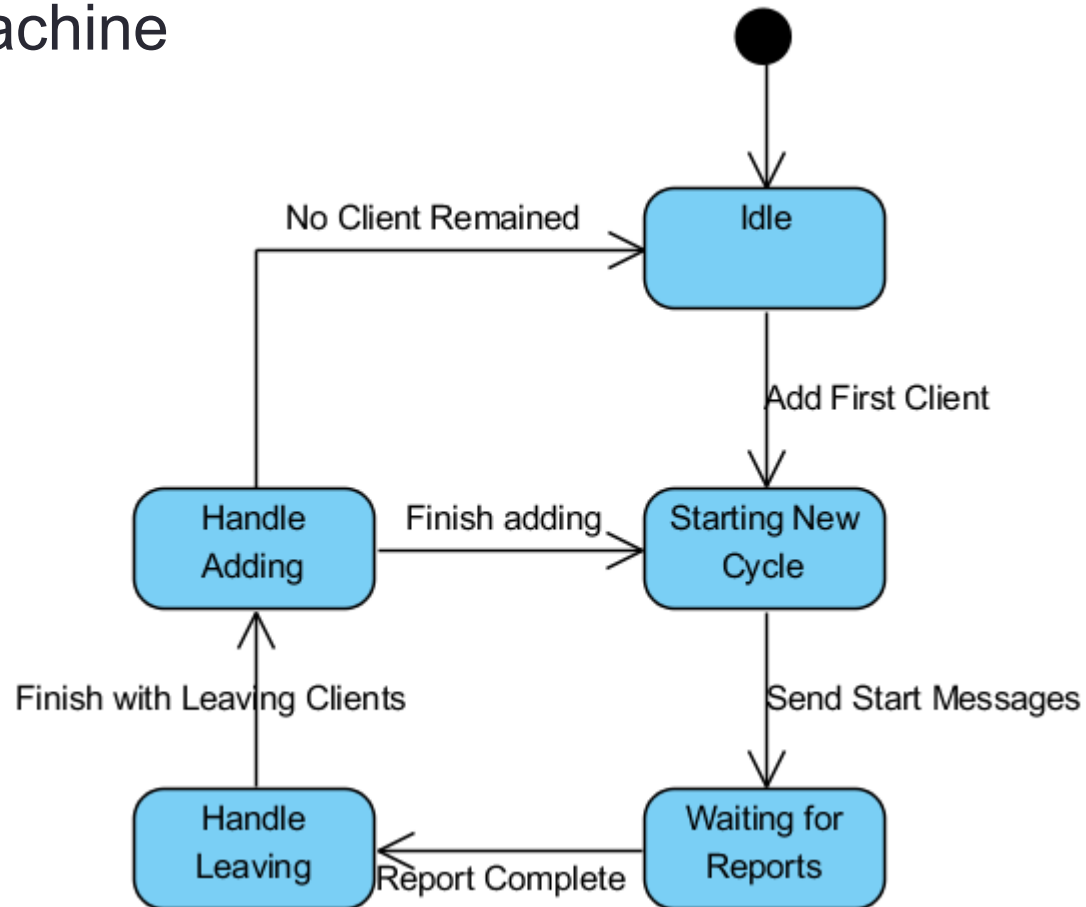


Message Passing

Message	Direction	Payload	Description
R1 Next Clock	S→C	new clock number	Inform all the clients to start new computation round
R2 Border	C→C	border status vector	Send border information to clients that is in charge of adjacent area
R3 Report	C→S	whether is leaving, top and left coordinate, status bit map	Report finished computation, and return the computation result to the server in interactive mode
R4 Confirm	A→A	none	Confirm received last message
R5 New Outfit	C→C	c o n n e c t i o n information, outfit information	Send all the computing information to a new added client
R6 Neighbor Update	C→C	c o n n e c t i o n information, relative position	Inform all the neighbors about the position change
J1 Request Join	C→S	connection information	Send request to server for joining the computation system
J2 Split	S→C	c o n n e c t i o n information, new client id, split direction	Inform the right client to split its area, construct the new outfit and send to the new client
L1 Last Merge	S→C	connection information	Inform the last two client to merge into one, the idle one waits for new command
L2 Outfit Merge	C→C	c o n n e c t i o n information, outfit	Send outfit to pair client for merging
L3 Leave Permit	S→C	connection information	Send outfit to the specified client

Server

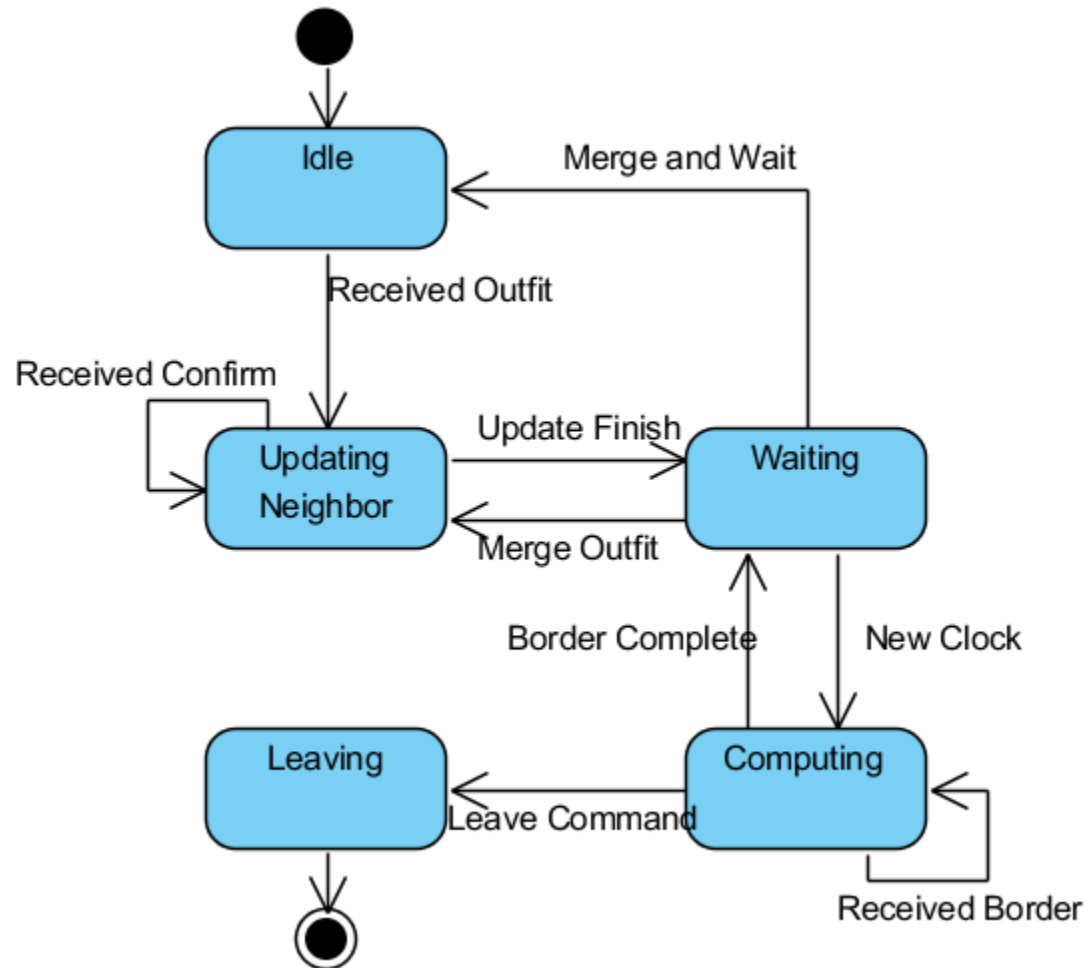
- State machine



Work Cycle

- Server
 - Cycle start
 - Synchronization
 - Wait for results & Handle missing
 - Merge & Refresh
 - Respond to requests(join, leave)
 - Cycle end

Client



Work Cycle (cont.)

- Client
 - Send joining application to server
 - wait for accepting
 - Cycle start
 - Respond to server's request of sibling change, replacement or new sibling
 - Synchronization
 - Compute
 - Compress if needed & Send
 - Back up if necessary
 - Send leaving application if necessary
 - Cycle end

Neighbor Updating


- Every node has up to 12 neighbors
- Send message to notice all neighbors

1	2	3	4
12			5
11			6
10	9	8	7


Message Queue




- Java NIO
- Multi-thread
 - Monitoring new connection
 - Monitoring new message and push to a queue with thread protection

Github Management



[Explore](#) [Gist](#) [Blog](#) [Help](#)

 leafpicker

 leafpicker / LifeGameSim

[Admin](#) [Unwatch](#) [Fork](#) [Pull Request](#) [2](#) [2](#)

[Code](#) [Network](#) [Pull Requests 0](#) [Issues 0](#) [Wiki 1](#) [Stats & Graphs](#)


No description or homepage.

[Clone in Mac](#) [ZIP](#) [SSH](#) [HTTP](#) [Git Read-Only](#) [Read+Write access](#)



[branch: master](#) [Files](#) [Commits](#) [Branches 1](#) [Tags](#) [Downloads](#)

Latest commit to the **master** branch

demo jar

 leafpicker authored 42 minutes ago [commit 4d77e15624](#)

LifeGameSim /

name	age	message	history
 .metadata	an hour ago	new sever address [Yao Xie]	
 BSDSocket	a month ago	addmessage [qddpx]	

Performance Testing

Ensure the system will perform well and be reliable:

- a) Under high user load
- b) With a large dataset
- c) Over an extended period time
- d) As the load/dataset increases

Test Objectives

- With the increase of clients, we want to observe the updates(cycles) per second
- The longest computing time in each round
- The average computing time of each client in each round
- The network flow (including sending message and receiving message)

Test Strategy

- Append incoming clients to a pending list
- Add a new client per 20 cycles as a new round
- $\text{cycles / second} = 20 * 1000 / (\text{endTime} - \text{startTime})$
- Each round:
 - Weakest-link principle (Cannikin Law)
 - compared with the longest computing time.
 - Monitor the sender and receiver and record the size of each package it send or receive
 - Total network flow = $\sum \text{sender} + \sum \text{receiver}$

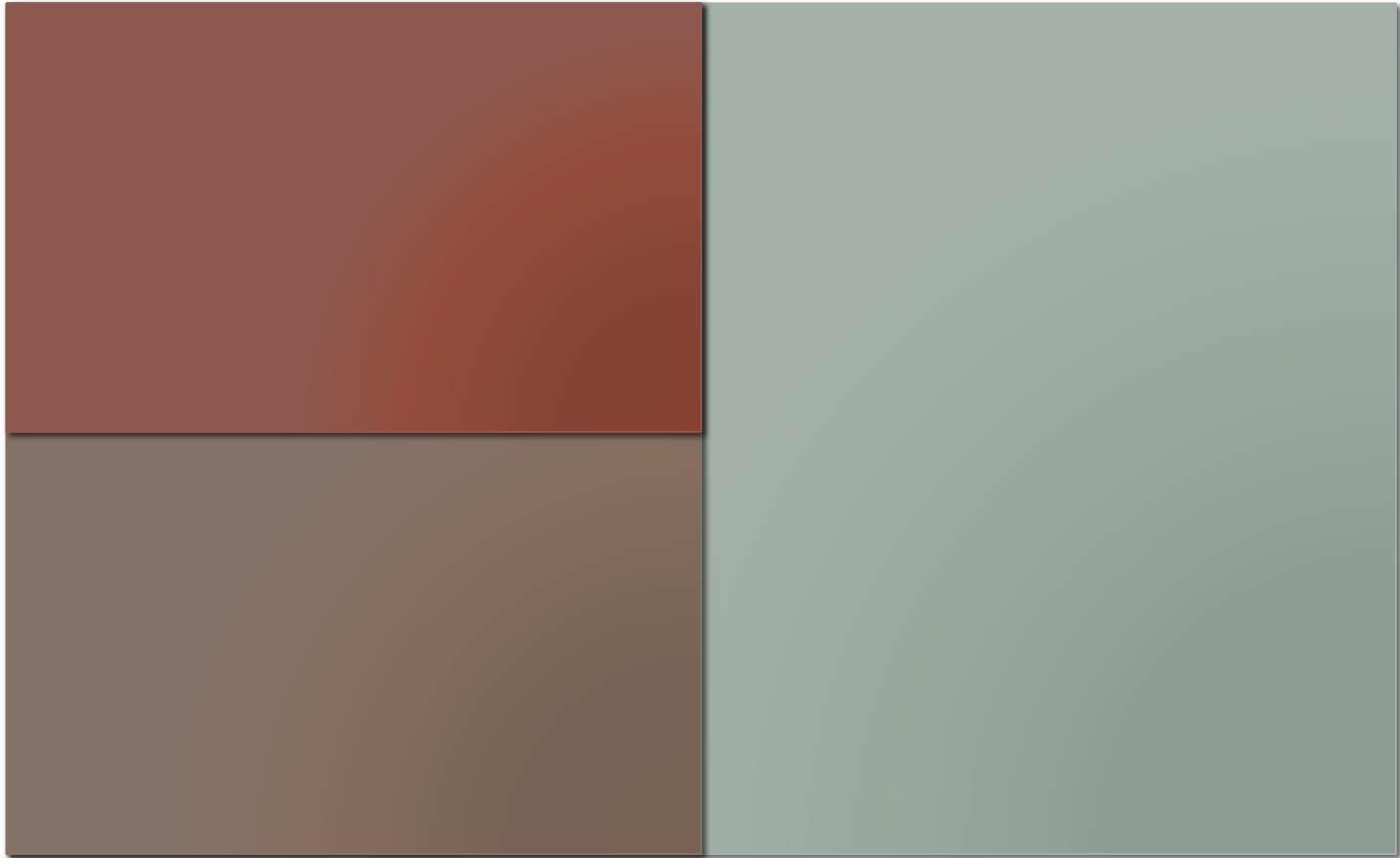
Cannikin Law



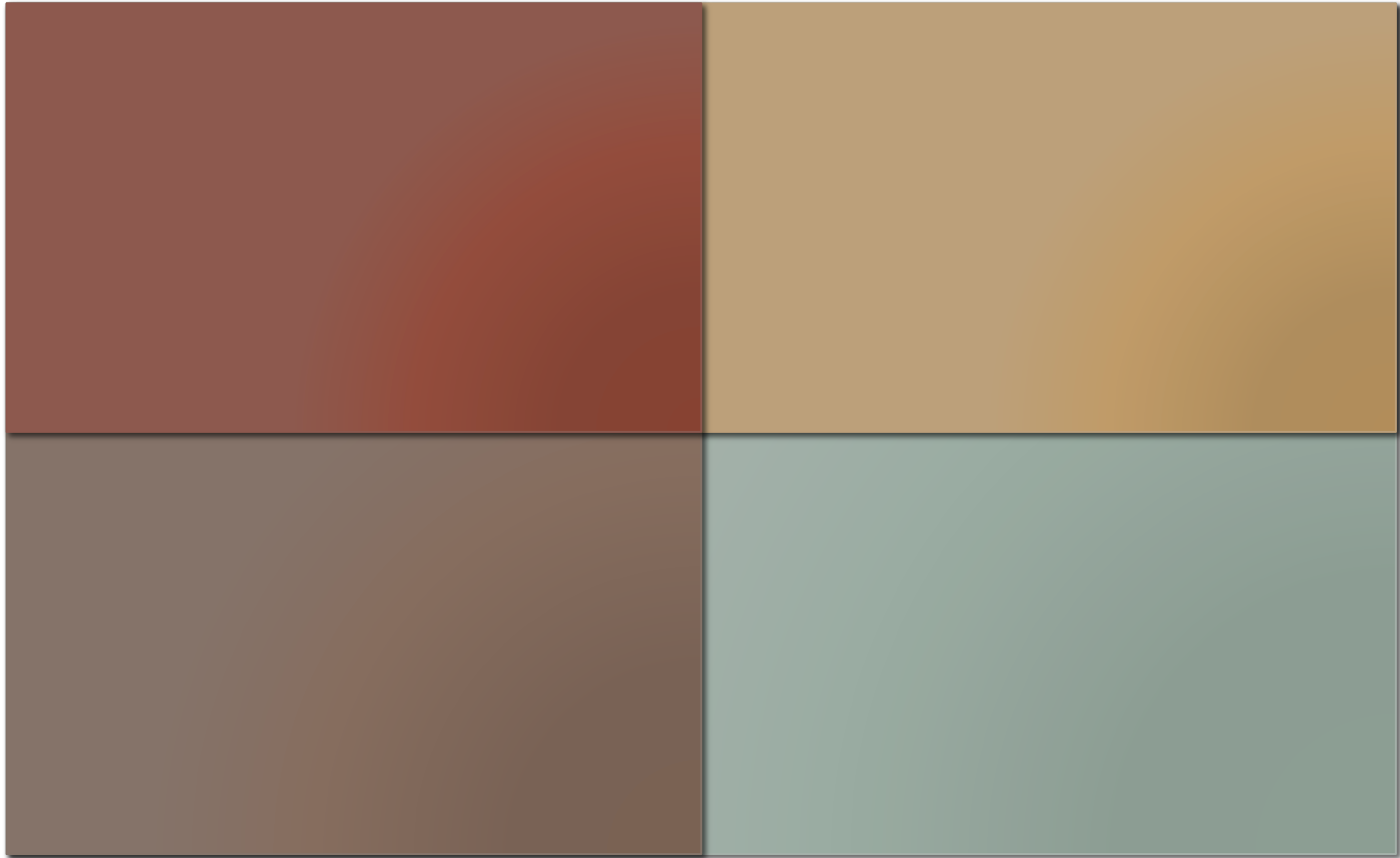
Cannikin Law



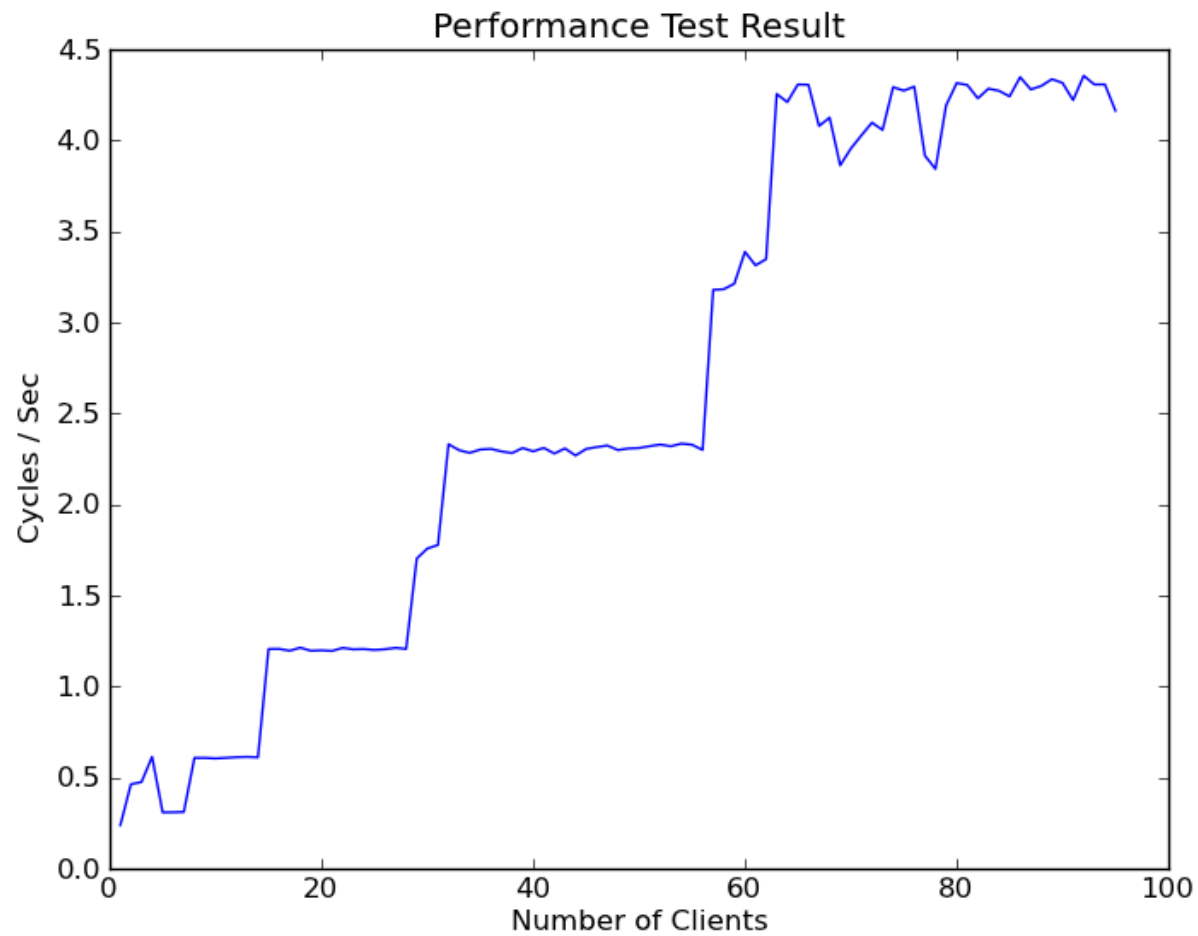
Cannikin Law



Cannikin Law



Sample Run 1



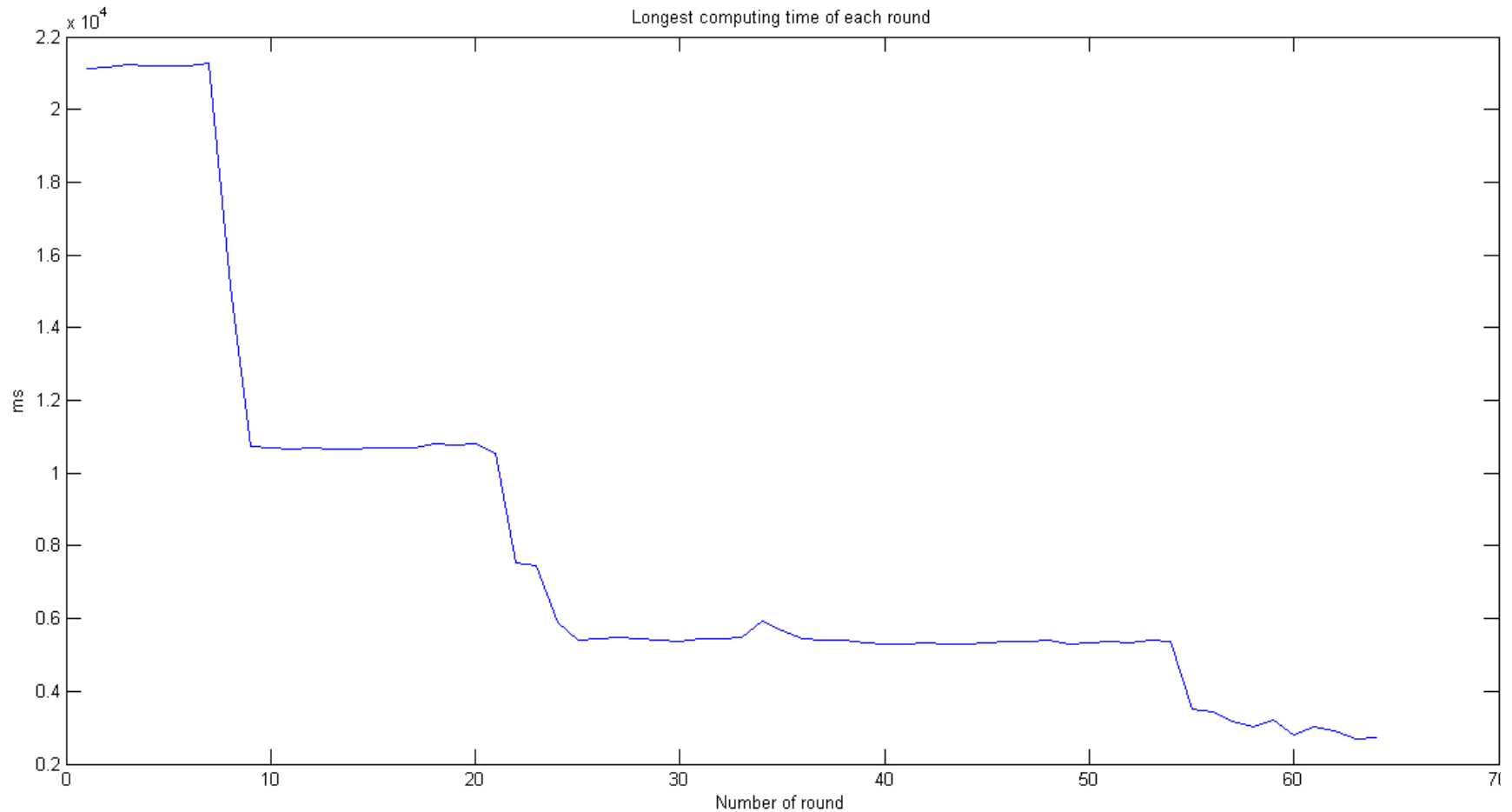
Sample Run 2



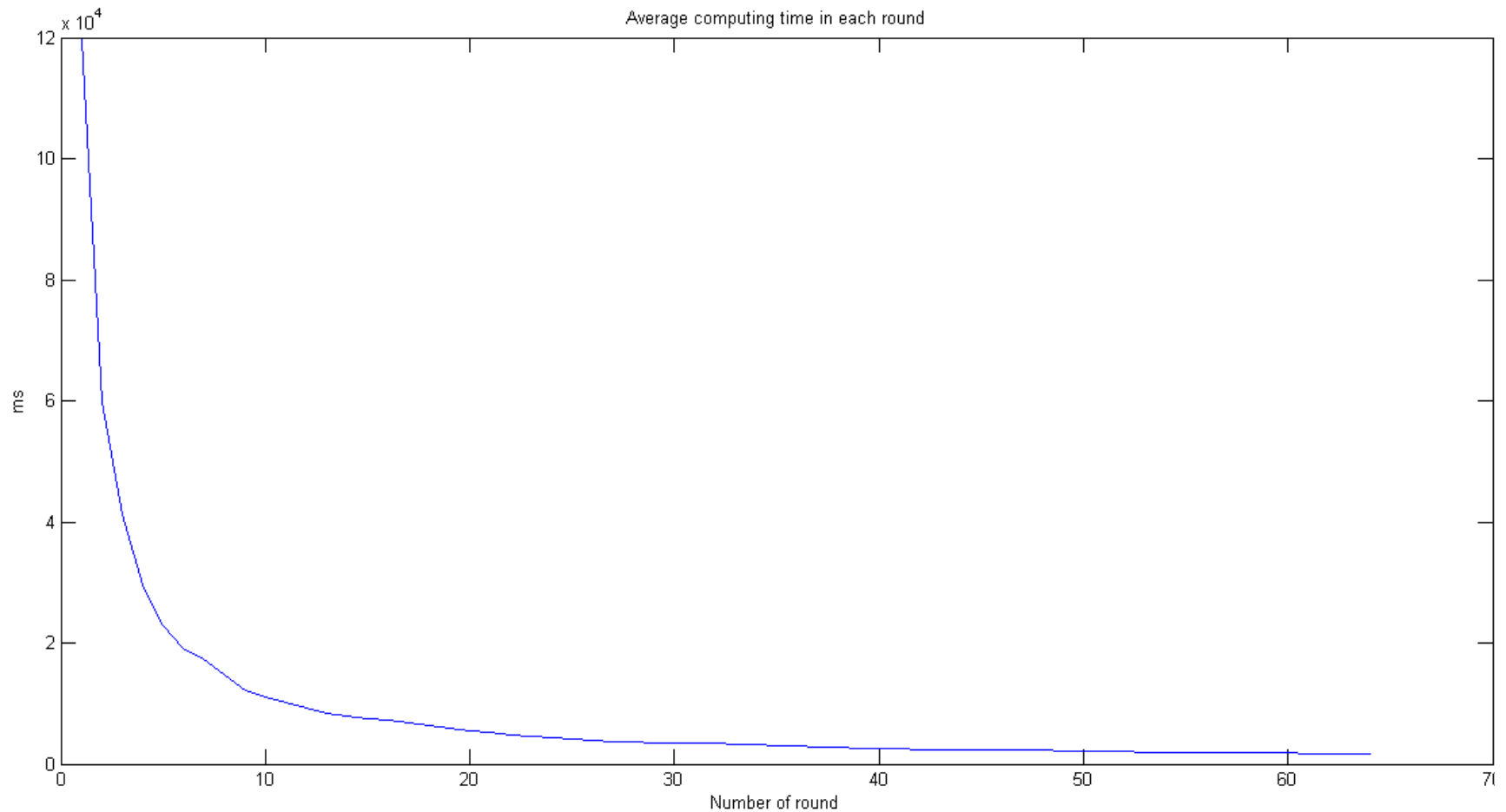
Sample Run 3



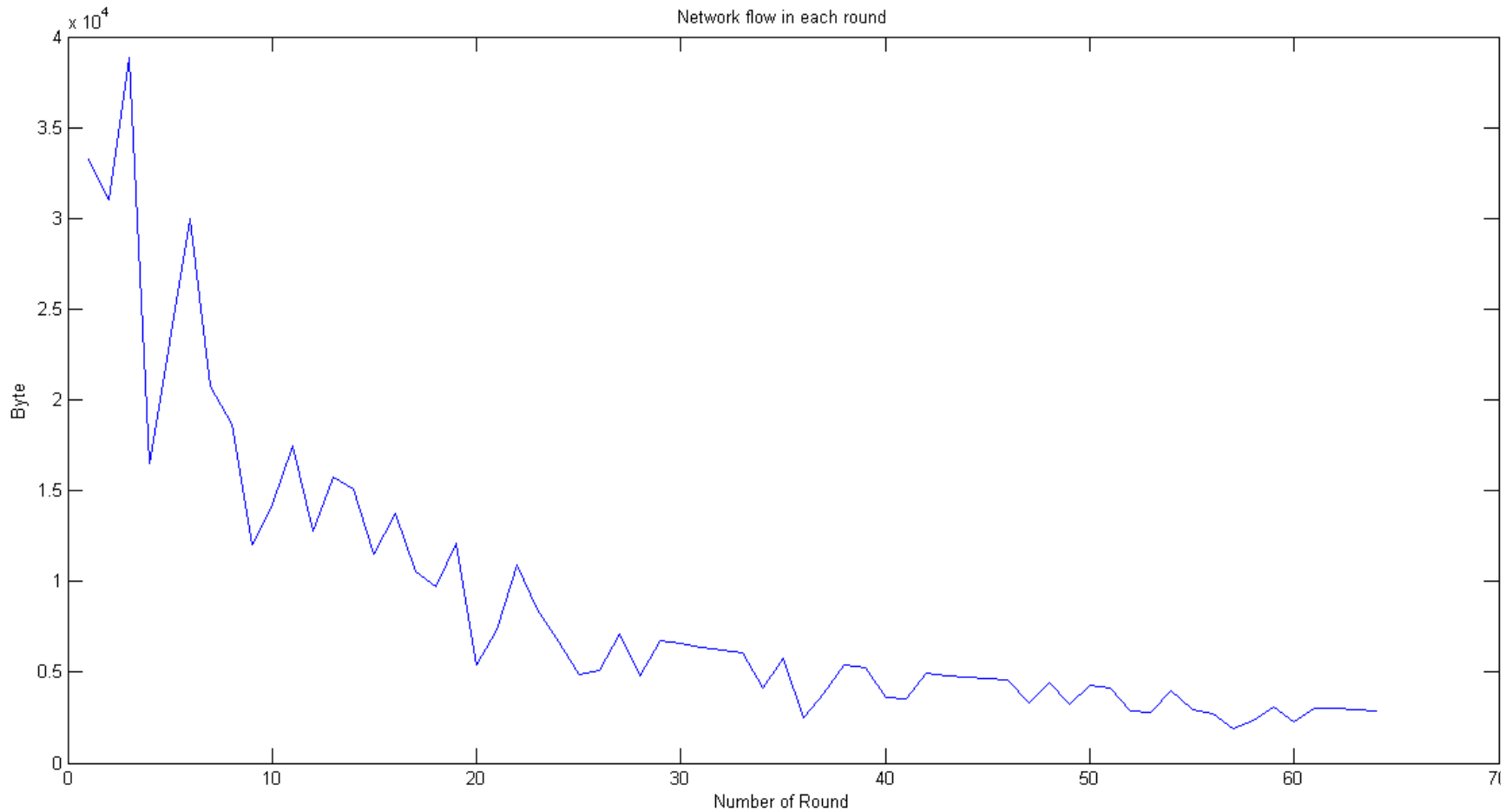
Longest Time



Average Time



Network Bandwidth



Conclusion

- The system can support dynamically adding and deleting computing nodes.
- It can support over 100,000,000 cells and 10,000,000 lives.
- With 100,000,000 cells, the system finishes about 4.5 cycles / sec, better than our expectation.

Questions?