Report

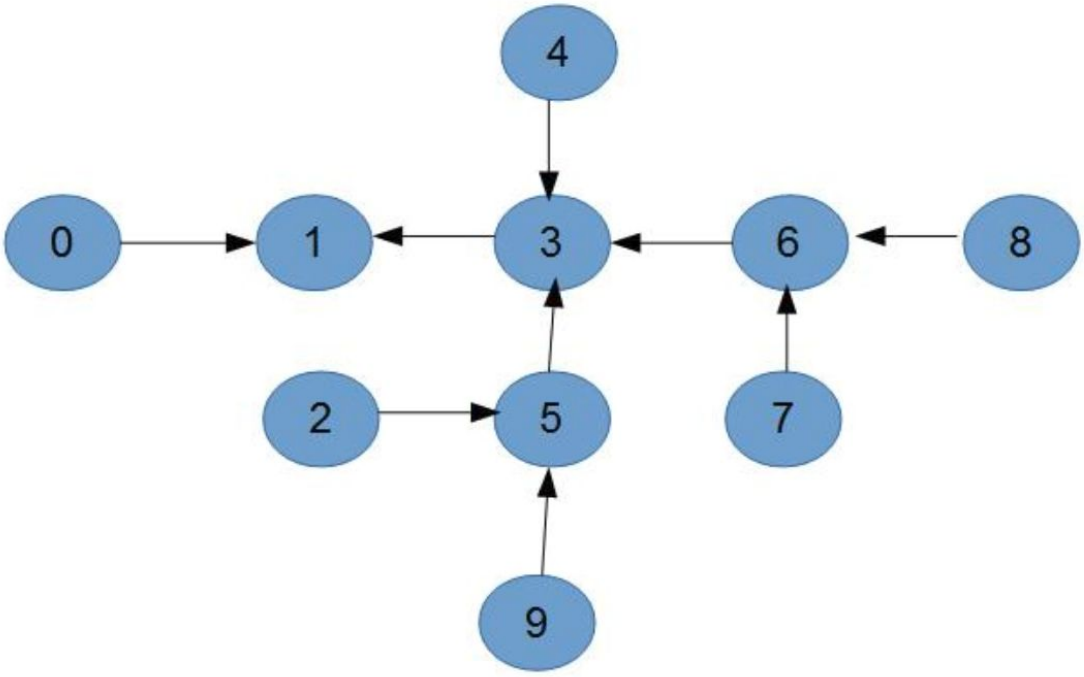
Design

- OpenMPI was used to simulate different nodes
- Each node have 2 threads running in parallel to simulate receiving of messages and working of critical section
- The following was treated as control messages: Request, Token messages
- Report messages were used to send the report containing the statistics of the algorithm
- The last process to have a critical section request pending collects the reports from all the other processes to print in the standard output, the statistics of the algorithm
- Termination messages were used for the programs to terminate gracefully
- The rest of the design was made in compliance with the details of the algorithm mentioned in the textbook
- Details of how to run the code and how to pass the input can be found in the Readme

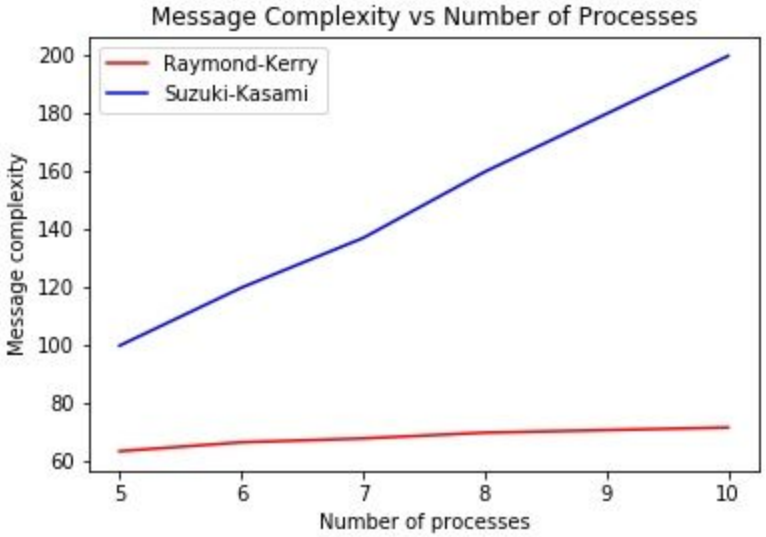
Graphs and Analysis

A fully connected graph was used for every run.

For Raymond-Kerry the following spanning tree was used over the fully connected graph. (remove the id from the graph in descending order to get the subgraphs)

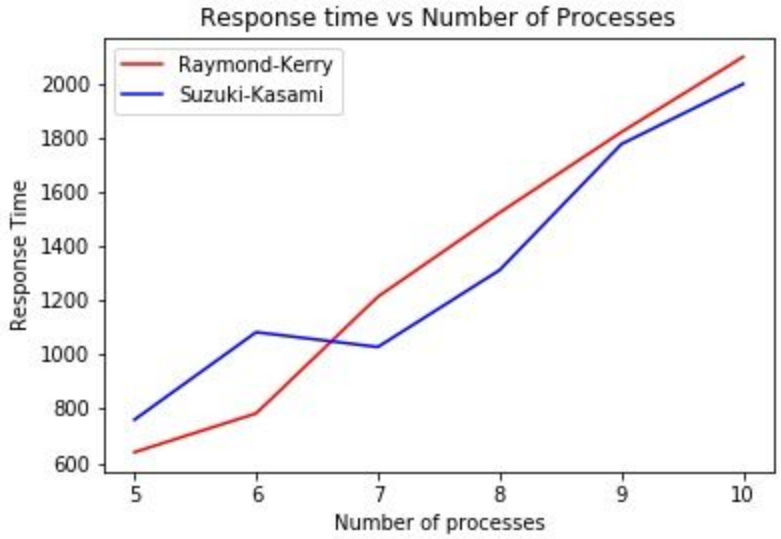


n = Number of processes, k=20, inital_node=1, alpha=2, beta = 2 was used



It can be inferred from the above graph that the message complexity of Raymond-Kerry stays almost a constant as the number of processes increase while that of Suzuki & Kasami increase almost linearly.

This is because we broadcast the request in the case of suzuki-kasami and with each increasing node, we need one extra message to broadcast where in Raymond-Kerry's algorithm no matter how many nodes are present, a node only requests its immediate neighbour for the token, so its average message complexity remains almost a constant.



It can be inferred from the above graph that both the algorithms take almost the same average time to serve the critical section request and increase almost linearly as the number of processes increase.

It can also be noticed that the time taken by Raymond Kerry is slightly higher than suzuki kasami, this is due to the fact that in Suzuki-Kasami's algorithm, the token gets directly sent to the node requesting the critical section whereas in Raymond-Kerry's algorithm, the token gets transferred along a spanning tree which causes the observed delay.

Anomalies observed

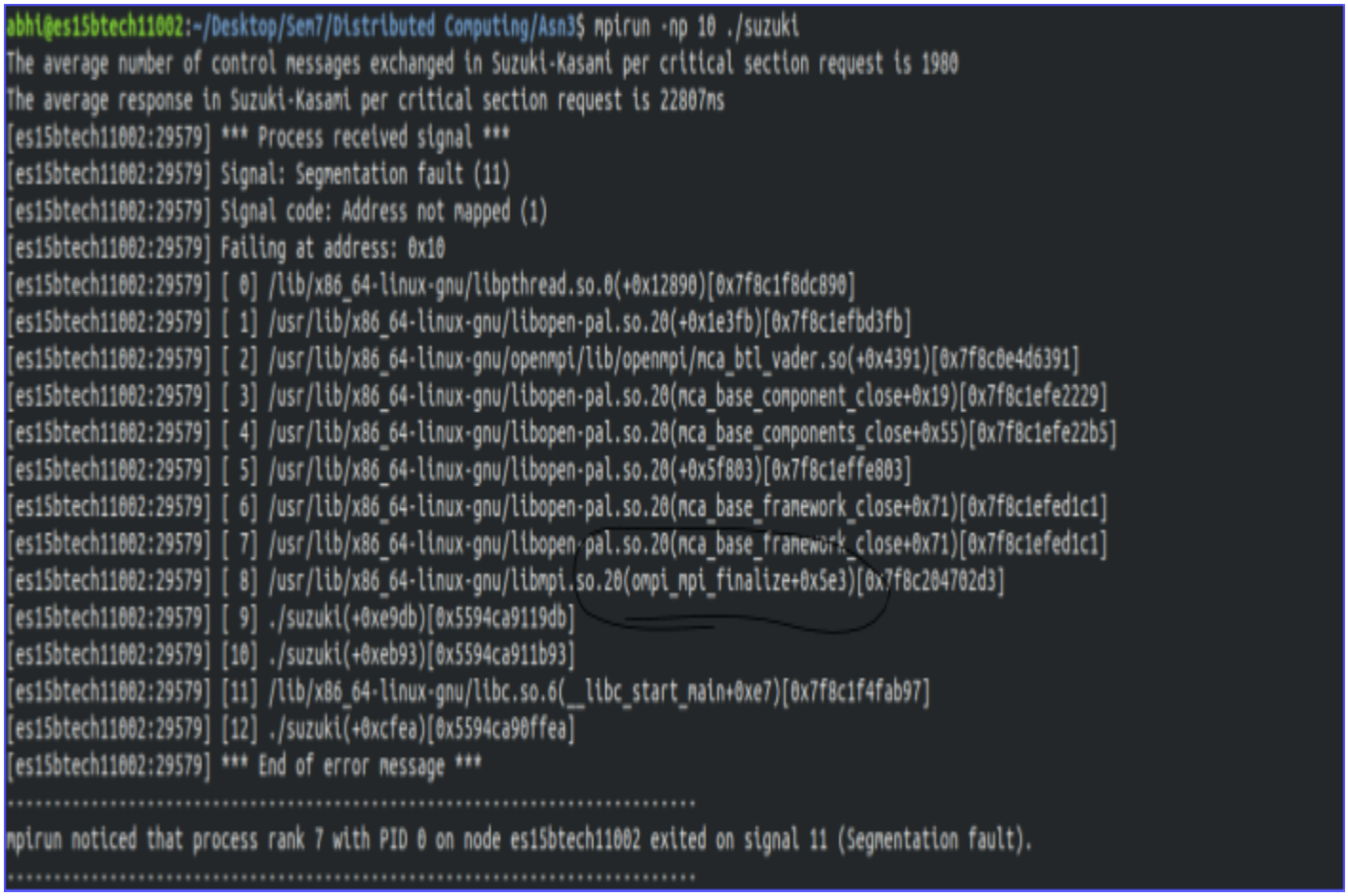
Initially we observe that Suzuki-Kasami takes more time to serve a critical section request than Raymond-Kerry, this is because we take into consideration the amount of time spent on sending the requests to all the processes via broadcast message if the node does not have the token, whereas in Raymond-Kerry only 1 message is sent in such a case.
This adds to the time and explains the anomaly observed.

Errors encountered

Sometimes(very rarely), there is a seg_fault which occurs when working with higher values of n.

It is because of an openMpi issue rather than a bug in the code

Following is the snapshot taken while debugging the seg_fault.



It can be observed above that the statistics get printed, this indicates all the processes have completed their critical section successfully.