Fault-tolerant & Adaptive Stochastic Routing Algorithm

for Network-on-Chip

Team CoheVer:

Zixin Wang, Rong Xu, Yang Jiao, Tan Bie

Idea & solution to be investigated by the project

There are some options available for the implementation of stochastic routing algorithm, including probabilistic flood, directed flood, random walk, etc. Firstly, we need to analyze the algorithm essence and choose one of them that can be suitably adapted into the NoC system. In order to accommodate adaptive features, the basic stochastic routing algorithm cannot be too complex, otherwise large area or power overhead will emerge. Thus, at current stage, we prefer to utilize random walk based stochastic algorithm due to its advantage in low complexity and relatively small area overhead.

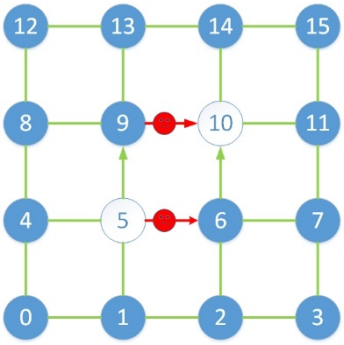
Even though stochastic routing algorithm gains advantage in fault-tolerance, its inherent drawback in performance (due to its oblivious routing strategy) restricts its popularity in NoC system. So we plan to add some adaptive features to our implementation of stochastic routing algorithm for high performance. Specifically, some modules that are responsible for collecting and analyzing real-time data of NoC will be added, by comparing probability of each route, the optimal route can be determined. Of course, adaptive features cannot do harm to fault-tolerance, some evaluations will be conducted not only for the performance improvement, but for its reliability.

In all, we plan to develop a fault-tolerant & adaptive stochastic routing algorithm for NoC in our project.

Progress so far:

1. A basic random-walk routing algorithm has been implemented in C++ and added into the 'routefunc.cpp' in Booksim.
   - For the routing algorithm we designed, dimension-order will be used if there is no fault in the network to achieve a smaller latency. While a packet enters a router with fault links, the router will change to use random-walk routing and randomly chose other ports to avoid the fault links.

2. Inserted fault tables into the Booksim to simulate a network with permeant links faults. So we could verify our routing algorithm in a faulty environment.

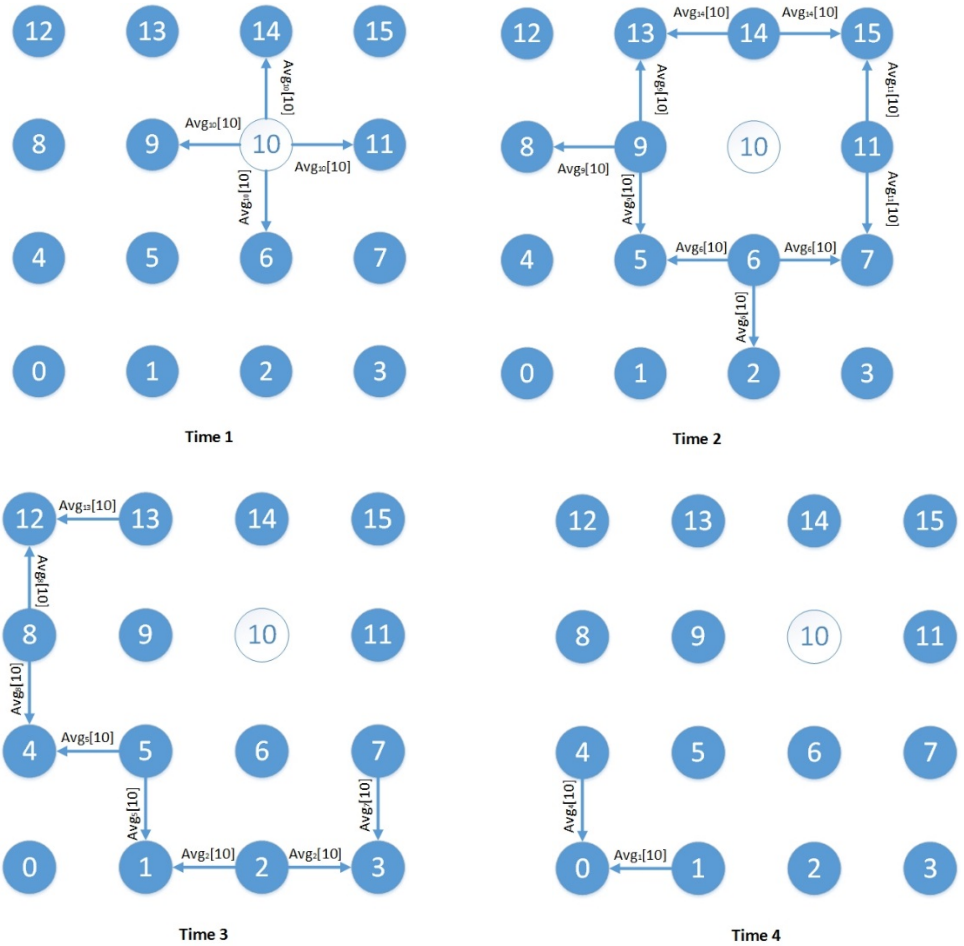


Here we use a 2D vector to construct a fault network and import to Booksim

Noticed for checkpoint 3: Right now, by improving the code, our fault table could be generated during the simulation process to emulate a real-life situation that hard link error occurs in runtime.

3. Based on the simulation results, we found that the average latency of this random-walk routing was too long and the performance wasn't satisfying. After the discussion with GSI, we decided to modified our baseline design.

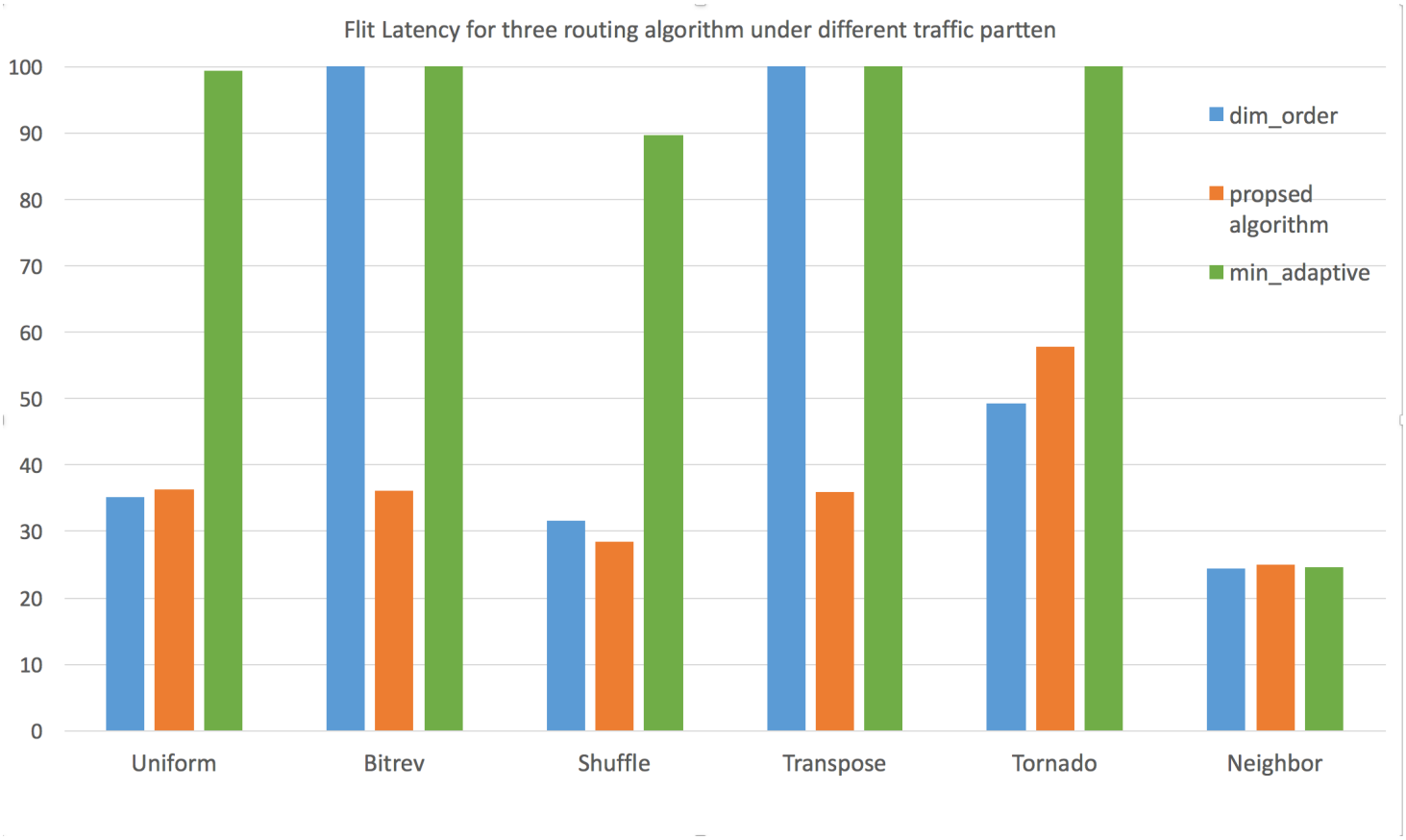
4. An adaptive feature has been added:
   - we implemented a global congestion table to store the estimated delay from each node to every possible designation in the network. The global congestion table is destination-based and upgraded periodically.
   - Then by using this table, we could generate port selection radio based on destinations for each router, which indicate the corresponding path delays. And all packets destined for the same destination are distributed in the same radio to the downstream routers so that balancing the network congestion and achieving a smaller over transmission latency.
   - The delay measurement and propagation is shown below.

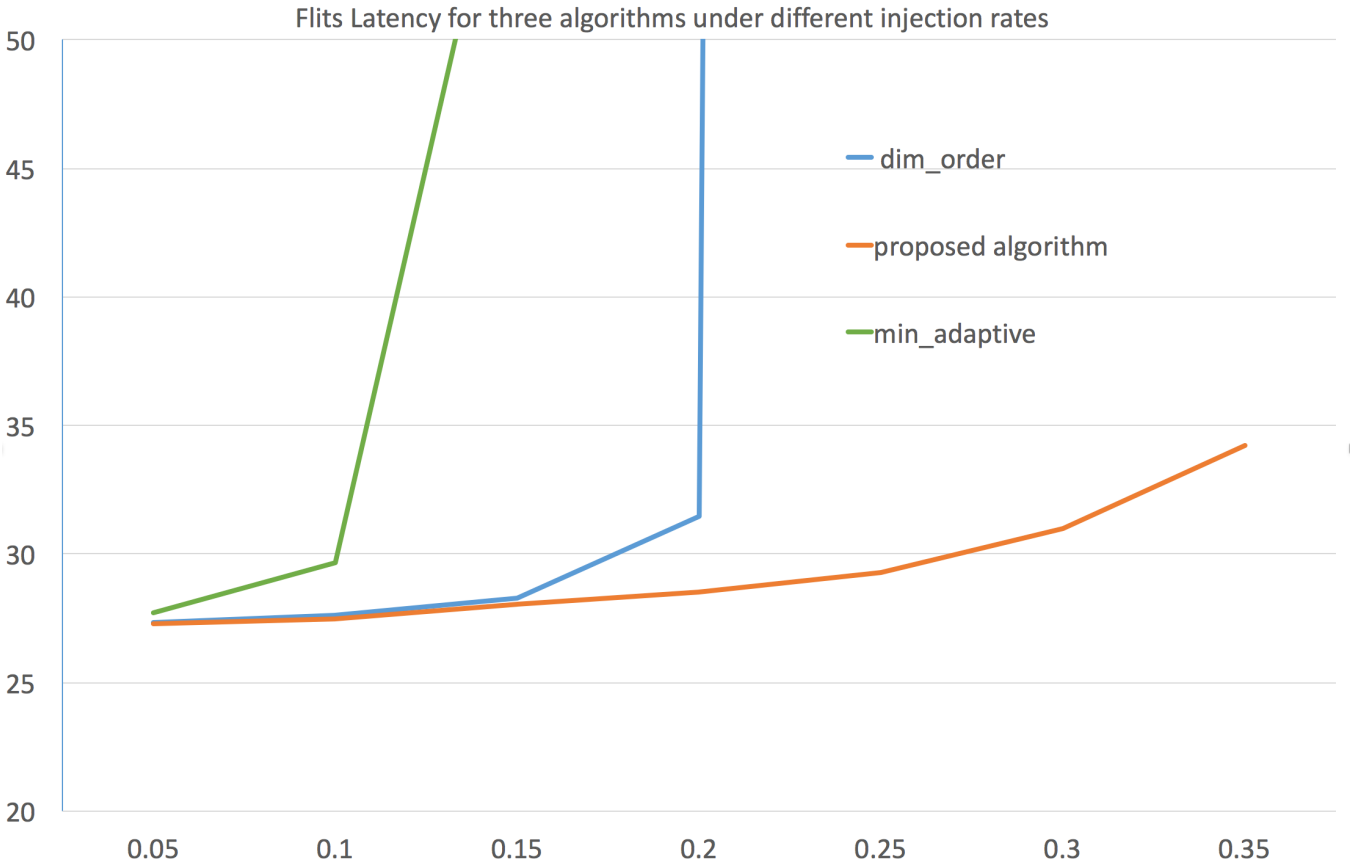


- And here is a part of the pseudo-code we implemented.

```
foreach i = 1:gNodes; //Destination
        foreach j = 1:2(gK-1);
                foreach k = 1:gNodes; //Source
                        if(abs(node[i].x-node[k].x)+abs(node[i].y-node[k].y) == j)
                                update_average[k];
                                update_w[k];
                        endif
                end
        end
end
```

5. Booksim simulation results based on our proposed adaptive routing without faults:





6. Fault-tolerant features have been implemented and simulated in Booksim:
   - For our project, we focus on hard link errors (permeant errors).
     The permeant link faults always mean a topology change. Thus, reconfiguration of the routing table is necessary to ensure the complete reachability.
     There are two families based on their methods to reconfiguration.
     i. One is deploying the routing tables and logic that are updated upon each fault occurrence in runtime.
     ii. The second solution based on the offline software to complete the reconfiguration upon any fault link detected and then communicate with surviving topology with a central node.

   - In this stage, we propose two possible mechanisms to cope with the hard link errors in the network both on the above two families.

A. Offline Software routing table reconfiguration

- Trigger:
  The packet arrives at a node and find its output port connect to a permanent link error. Then this defect information will be send to the central node to trigger the offline software and the fault node pipeline enter 'reconfiguration' stage.
- Reconfiguration:

The software then calculates and generates an updated routing table for all nodes in the network for all destination based on the link error arised.

- Table forward:
  The central node forward the reconfiguration results to each node in the network, invalid the original routing table. The fault node finish 'reconfiguration' stage and process the flits inside.
- For this mechanism, the latency a critical factor and mainly consisted of three parts.
  1) Delay of transfer fault to software.
  2) The computation time of the software.
  3) the delay of result forward.

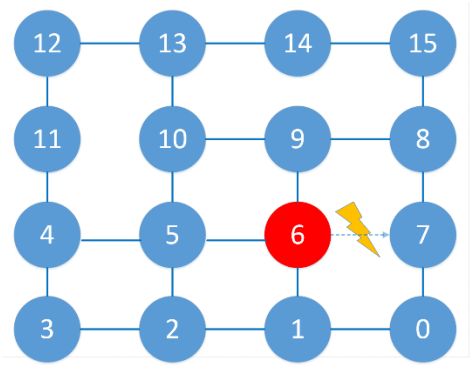
B. On-chip routing table reconfiguration combined global congestion analysis.

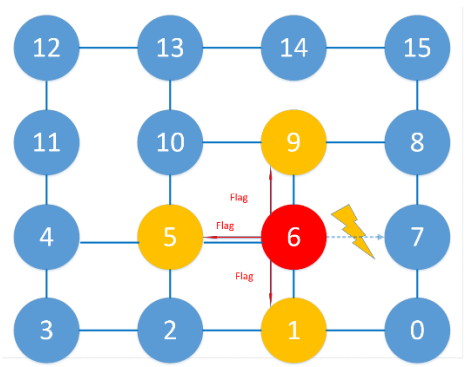
- Here we assume that every node could be able to check whether there is a fault link connected to its port by Virtual channel allocator maintaining the state information of its adjacent nodes. Then when a fault link is detected, the network would start route table reconfiguration.
- The reconfiguration process is described in details as blow.
  a. The node with fault links become to a root node. The root node broadcast and propagate a 1-bit reconfiguration flag to all nodes in the network only through healthy links hop-by-hop. Meanwhile the network delay Avg[i] are also transmitted with the flag signal to update network congestion information stored in each node.
  b. For each node received the reconfiguration flag.
     i. Update the routing tables. For ports receiving the reconfiguration flag, calculate and store W[x][i] according to the propagated delay from downstream nodes. For ports not receiving the flag, invalid current W[x][i] and set to zero. Then the average delay of current node are also calculated.

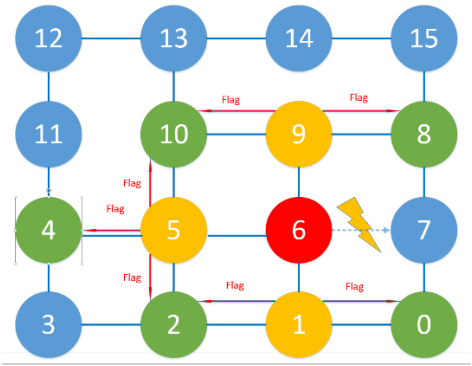
For example

Destination (i)	West	North	East	North
Radio (W)	0.6 → 0.55	0.4 → 0	0 → 0.45	0
Flag received	Yes	No	Yes	No

     ii. Flag forwarding. Nodes send reconfiguration flag only to those ports which didn't receive a flag or connect to a fault link. In this process, up/down restriction is applied to ensure a deadlock-free routing.







c. For each node detecting a hard link error, repeat the process 1 and 2 to obtain an updated routing table with safe path from each other node to this faulty node as the destination as well as the network congestion information to decide these safe path adaptively.

7. A router architecture used in our adaptive fault-tolerant routing algorithm is also analyzed and under implementing in RTL design to estimate the area overhead of our algorithm. Right now, the hardware design of global congestion table inside a node has been done.

8. Deadlock and livelock resolution are under implementing. We plan to use escaped virtual channel so that a safe path through the whole network is reserved. Upon a deadlock or livelock happens, in-flight flits will be sent to escaped virtual channel and get to the destination in the end

Issues/showstoppers:

1. On-chip reconfiguration is still under implementing and further analysis is necessary.
2. The deadlock & livelock resolution is under testing, thus some scenarios during Booksim simulation are still unable to finish.

Further Works:

1. Improve the performance of our algorithm both in fault-free and fault networks, may investigate some new features as well.
2. Debug the potential corner cases lead to failures in the simulation.
3. Implement and test deadlock & livelock resolution in Booksim
4. Fully test and analyze our final routing algorithm in different scenarios like stressful condition to compare the latency performance.
5. Implement the router architecture in RTL design and synthesis in VCS to estimate the area overhead for our routing algorithm.