AI+X Computing Acceleration: From Algorithms Development, Analysis, to Deployment: Final Report

Yuqi Ren 2022011332
Jinfan Lu 2022010741

1 Introduction

In this project, we have implemented two advanced network topologies, SlimFly and DragonFly, alongside their corresponding routing algorithms and a deadlock-avoidance technique that utilizes virtual channels. SlimFly and DragonFly are among the most efficient topologies, with diameters of two and three, respectively.

...

2 Architecture

2.1 Topology

- **SlimFly**: The SlimFly topology is characterized by a highly symmetric internal structure, consisting of two subgraphs, each composed of identical subgroups of routers. As depicted in Figure 1, the first subgraph consists of routers $(0, x, y)$, while the second is formed by routers $(1, m, c)$. These two subgraphs comprise q subgraphs that do not directly connect with each other, with each subgraph containing q routers. Router $(0, x, y)$ and router $(1, m, c)$ are connected if and only if $y = mx + c$.
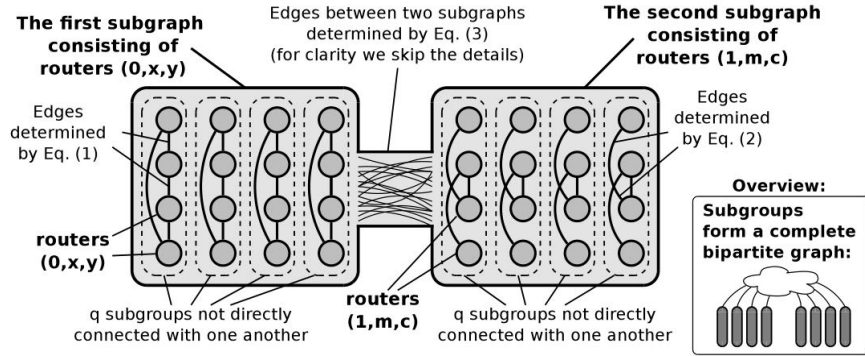

Figure 1: SlimFly topology

  Based on this topology, we can derive several key metrics: Diameter = 2, Degree = $O(q)$, Average Distance ≈ 2, and Cost = $O(q^3)$.

- **DragonFly**: The DragonFly topology is a hierarchical network organized into three levels: system, group, and router. At the group level, n routers within the same group are fully interconnected. At the system level, there are m groups, and each pair of groups is connected by a single link. Specifically, the i-th group and the j-th group are connected via global channels between the $(i \bmod n)$-th router in group j and the $(j - 1 \bmod n)$-th router in group i. From this topology, we can compute the following

metrics: Diameter $= 3$, Degree $\leq n - 1 + \lceil m/n \rceil$, Average Distance ≈ 3, and Cost $= O(n^2 m)$. An example of a special DragonFly topology ($m = 2n + 1$) is shown in Figure 2.
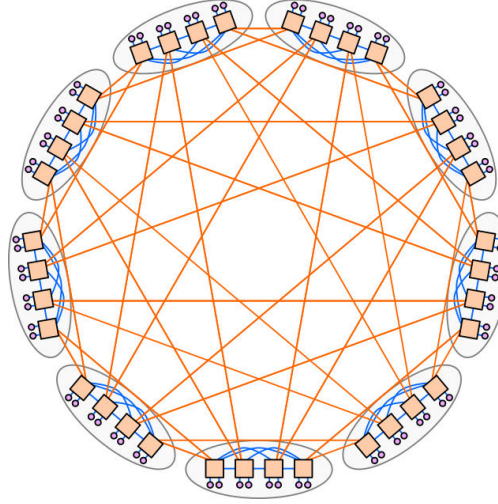

Figure 2: DragonFly topology

2.2 Routing

We have implemented two fundamental routing methods for the DragonFly and SlimFly topologies:

- **Minimal Static Routing**: In minimal routing, a packet is forwarded along the shortest path from the source router R_s to the destination router R_d.

- **Valiant Random Routing**: In Valiant routing, the protocol first randomly selects an intermediate router R_r. The packet is then routed from the source router R_s to the intermediate router R_r, and finally from R_r to the destination router R_d. The resulting paths may consist of 2, 3, or 4 hops.

2.3 Deadlock Freedom

To ensure deadlock avoidance in SlimFly, we implement a technique that utilizes virtual channels. When sending a packet from router R_a to R_b, if they are directly connected, virtual channel VC_0 is used. Otherwise, as the maximum distance in SlimFly is two hops, virtual channel VC_0 is used for the first hop and VC_1 for the second hop. This approach effectively breaks cycles into different sets of buffers, thus preventing deadlocks.

3 Implementation

3.1 Topology

- **SlimFly Topology**: The implementation of SlimFly involves two main steps: adding intra-group links and inter-group links. First, we divide routers into groups based on their indices. Then, we connect router $(0, x, y)$ with router $(0, x, y')$ if $y - y' \in X$, and connect router $(1, m, c)$ with router $(1, m', c')$ if $c - c' \in Y$, where $X = \{1, \xi^2, \dots, \xi^{q-3}\}$, $Y = \{\xi, \xi^3, \dots, \xi^{q-2}\}$, and ξ is an element of the Galois Field $\mathbb{F}_q$ that generates $\mathbb{F}_q$. Finally, we add inter-group links by connecting router $(0, x, y)$ with router $(1, m, c)$ if and only if $y = mx + c$.

  In our implementation, considering a configuration with 64 CPUs, we set $q = 5$, resulting in 50 routers. Therefore, we have $X = \{1, 4\}$ and $X' = \{2, 3\}$.

- **DragonFly Topology**: The implementation of DragonFly follows a similar approach as SlimFly. We first add intra-group links and then inter-group links. Denoting the i-th router in group j as (i, j), we add inter-group links as follows: connect router $(j - 1 \mod n, i)$ with router $(i \mod n, j)$.

3.2 Routing

- **Minimal Static Routing**: To find the shortest path, we assign a weight of 1 to all links in the network. Since the function `lookupRoutingTable` in `RoutingUnit.cc` will select the path of smallest weight, we can directly call it and then get the shortest path.

- **Valiant Random Routing**: To select the intermediate router, we define a random selection mechanism that balances traffic. Specifically, we add a member variable `mid_router` to class `RouteInfo` in `src/mem/ruby/network/garnet/CommonTypes.hh` and decide the path in `RoutingUnit.cc`.

3.3 Deadlock Freedom

In our implementation, we set the number of virtual channels to 2. When sending a packet, we check whether the destination router is directly connected. If it is, we use VC_0; otherwise, we use VC_0 for the first hop and VC_1 for the second hop.

This modification mainly involves changes in `src/mem/ruby/network/garnet/OutputUnit.cc`, specifically in the logic for the functions `has_free_vc` and `select_free_vc`.

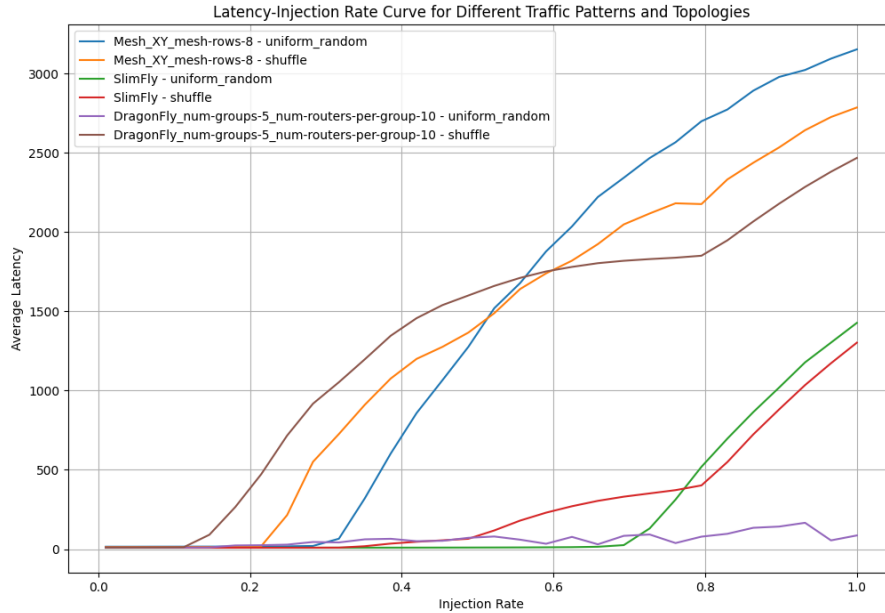# 4 Experiments and Analysis

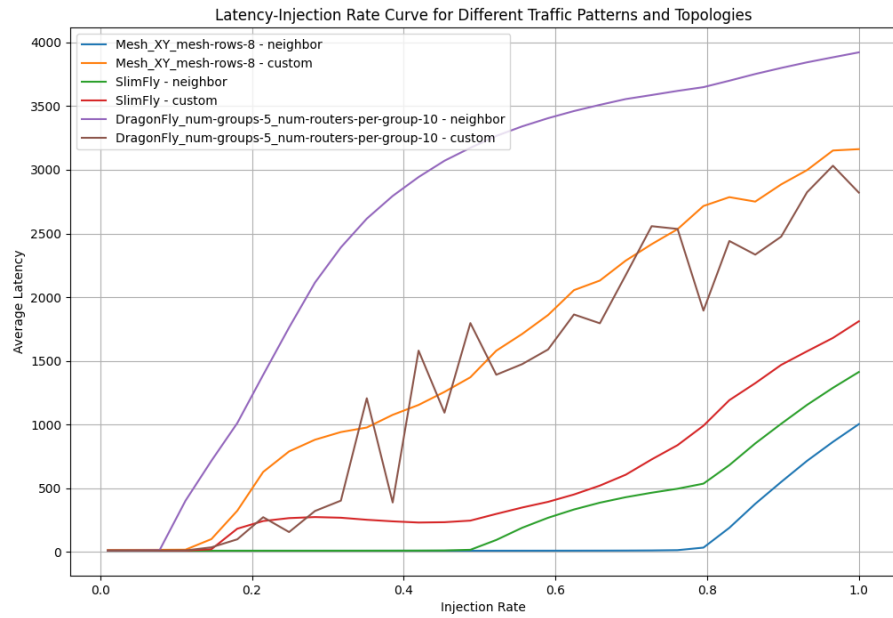## 4.1 Topology

Figure 3: Experiemnts of topology

Figure 4: Experiemnts of topology

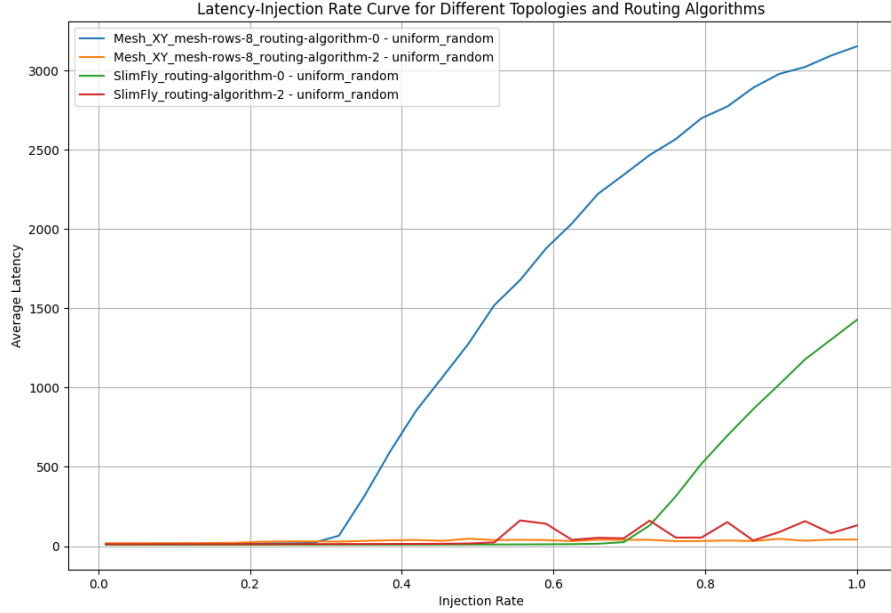As shown in Figure 3 and Figure 4, ...

4.2 Routing

Figure 5: Experiemnts of routing

4.3 Deadlock Freedom

To evaluate the effectiveness of the deadlock-avoidance technique, we conducted simulations with and without the deadlock prevention algorithm. The built-in synthetic traffic patterns in Garnet are not suitable for detecting deadlocks in SlimFly, so we implemented a custom synthetic traffic pattern. In this pattern, most packets follow a specific path (R_x, R_y), which is clearly illustrated below.
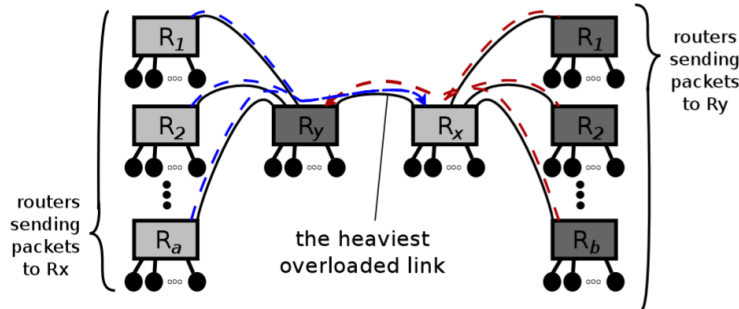

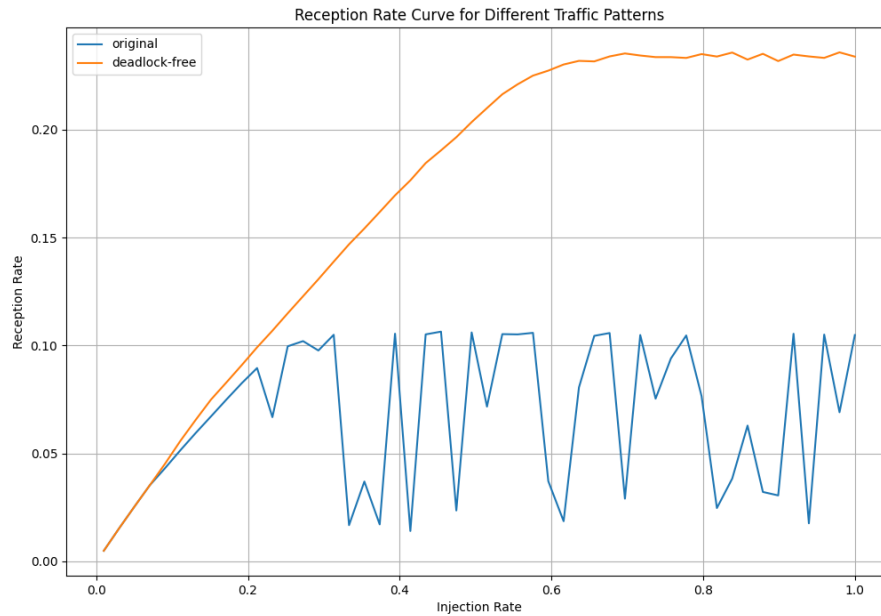Figure 6: Illustration of the worst-case scenario for SlimFly

Figure 7: Experiemnts of deadlock free algorithm

We use reception rate to indicate the occurrence of deadlocks in the minimal static routing algorithm. The original algorithm shows some points with extremely low reception rates, indicating deadlocks. Due to the inherent randomness in our synthetic traffic pattern, some areas may still exhibit higher reception rates.

5 Labor distribution

Yuqi Ren (50%):

- Paper reading (SlimFly, DragonFly)

- DragonFly topology

- Minimal routing, Valiant random routing

- Experiments (50%)

- Report (Topology, Routing)

- PPT, Pre

Jinfan Lu (50%):

- Paper reading (SlimFly, deadlock)

- SlimFly topology

- Valiant random routing

- Experiments (50%)

- Report (deadlock)

- PPT, Pre

In summary, we complete the entire project with equal contributions.

6 Reference

- Besta, Maciej, and Torsten Hoefler. "Slim fly: A cost effective low-diameter network topology." SC'14: proceedings of the international conference for high performance computing, networking, storage and analysis. IEEE, 2014.

- Kim, John, et al. "Technology-driven, highly-scalable dragonfly topology." ACM SIGARCH Computer Architecture News 36.3 (2008): 77-88.

- Besta, Maciej, et al. "Slim noc: A low-diameter on-chip network topology for high energy efficiency and scalability." ACM SIGPLAN Notices 53.2 (2018): 43-55.

Appendix (How to Reproduce Experimental Results)

```
# 1: Install Dependencies
sudo apt install build-essential git m4 scons zlib1g zlib1g-dev \
libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev \
python3-dev libboost-all-dev pkg-config

# 2: Build
git clone https://github.com/gem5/gem5
cd gem5
git checkout v23.0.0.1
# replace /path/to/lab4.patch with the actual path of lab4.patch
git apply /path/to/lab4.patch
pip install -r requirements.txt
scons build/NULL/gem5.opt PROTOCOL=Garnet_standalone -j $(nproc)

# 3: Run
python ./experiemnts/topology_1.py      # Figure 3
python ./experiemnts/topology_2.py      # Figure 4
python ./experiemnts/routing.py         # Figure 5
python ./experiemnts/deadlock.py        # Figure 7
```