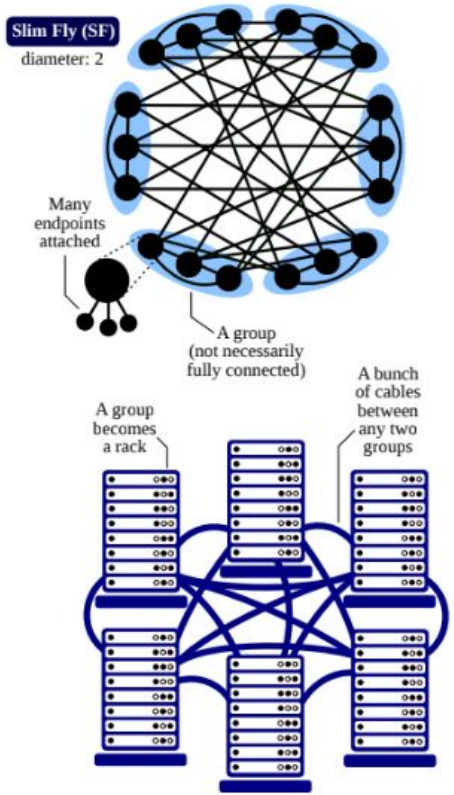
Real World Evaluation of the Slim Fly Network

Edward Brees

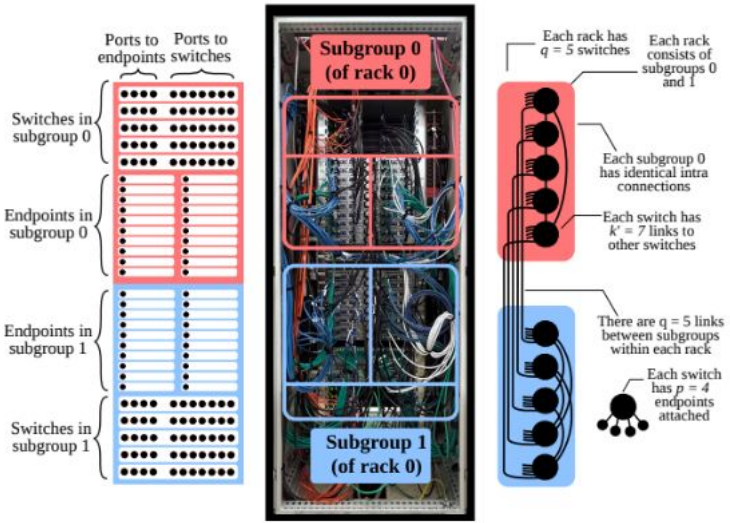
Bet You Haven't Heard of This Before

- Network Topologies
  - The need for efficient data transfer and communication in high-performance computing
  - Latency, cost, and power consumption
- Slim Fly (SF)
  - Proposed solution with theoretical (key word) benefits of low-diameter
  - This implies low latency and cost savings



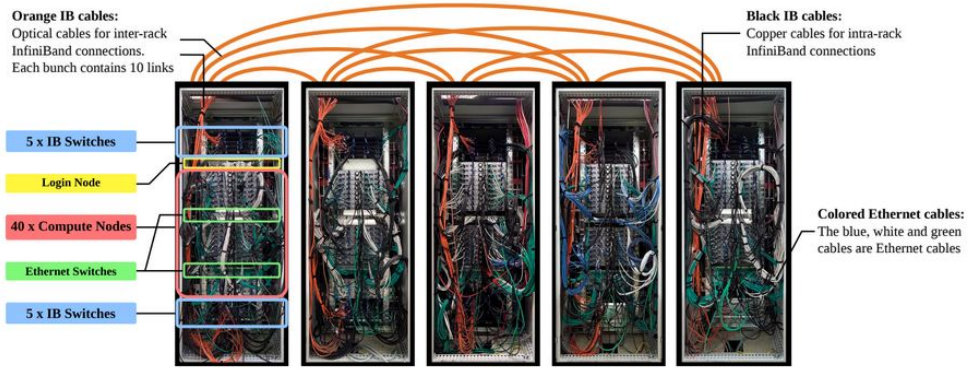
From Theoretical to Reality

- Slim Fly's benefits have not been empirically tested in real-world environments.
- Research paper objective
  - To deploy an at-scale SF network and rigorously evaluate its performance, cost, and power efficiency.
- Approach
  - Implementation of open-source cabling and physical layout routines to facilitate practical deployments.



Deployment Challenges and Implementation

- Challenges
  - Ensuring accurate cabling to maintain SF's low-diameter benefits
  - Managing cost and power efficiency while keeping practical scalability
- Implementation
  - Physical layout was optimized to reflect Slimfly setup.
  - Developed open-source cabling tools to assist in error-free installation.



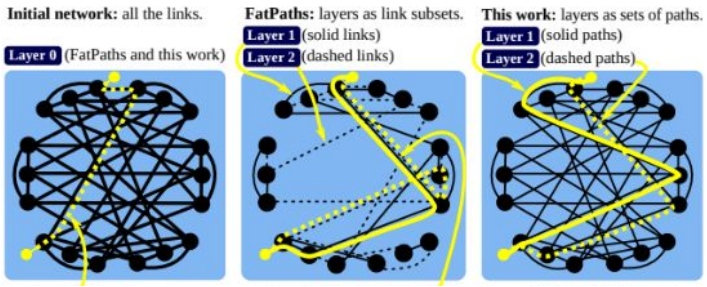
Proposed Multipathing

- FatPaths as a basis
  - Utilizes multiple paths to avoid congestion and evenly distribute traffic.
- Improvements made upon it (Layering)
  - Achieves improved load balancing through layered routing approaches.
  - Incorporates adaptive routing algorithms that are aware of network state.

Algorithm 1: Construct routing layers; details are in § 4.3

Input : Network topology $G = (V, E)$, number of layers $|L|$
Result : A set of L routing layers
// $W \in \mathbb{R}^{N_r \times N_r}$ contains weights of links; p is a priority queue, with entries being pairs of nodes

```
W = init_link_weight_matrix() // Set all matrix entries to 0
p = init_p_queue(G) // Each node pair gets the same priority
L = {E} // Layer 0 contains all the links (E)
for l = 1 to |L| − 1 do
    init_layer(l) // Initialize the next layer as empty
    node_pairs = copy_pairs(p)
    while node_pairs ≠ ∅ do
        pair = node_pairs.dequeue()
        path = find_path(G, W, pair, l)
        if valid(path) then
            update_priorities(path, p)
            update_weights(path, W)
            add_path_to_layer(path, G, l)
        end
    end
    L = L ∪ {l} // Add a new layer to finalized layers
end
```

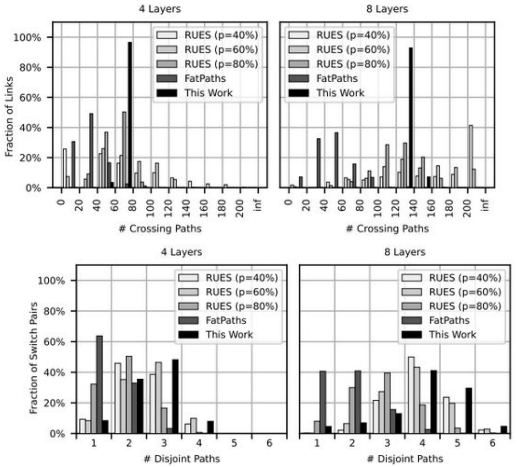


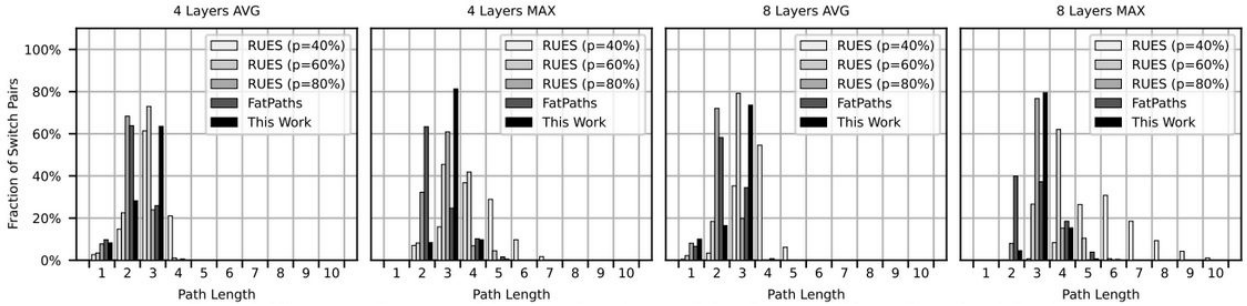
Implementing Physical Multipathing

- Final tweaks and adjustments
  - Accounted for practical problems such as link failures, deadlocks and traffic variations
  - Tailored FatPaths routing for optimized performance with SlimFly characteristics
- Physical Implementation
  - Load balancing is achieved by distributing traffic over multiple equal-cost paths.
  - Systematic approach to physical and logical network design enabled efficient routing.
  - Validated the FatPaths approach through real-world measurements and experiments.

Theoretical Analysis of Paths

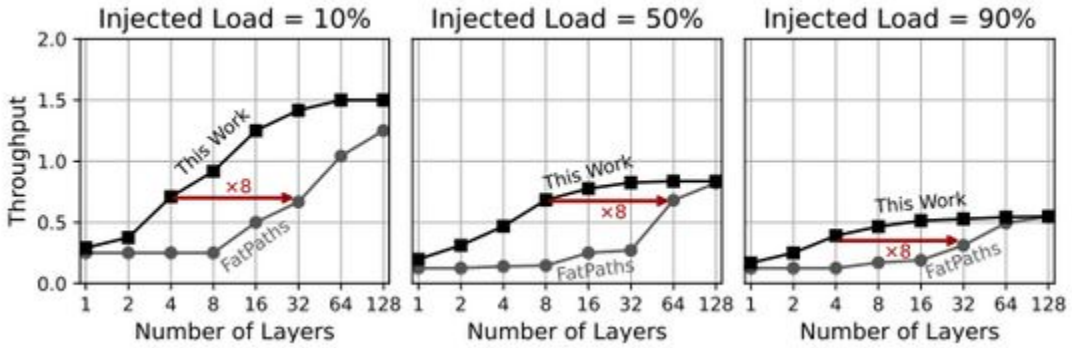
- RUES and p%
  - RUES (Random Uniform Edge Selection) is used to measure theoretical equitable traffic handling
  - p% indicates the fairness in path usage at equilibrium
  - ‘This Work’ is unadjusted physical observed data
- Path Lengths
  - Theoretical shortest path lengths remain consistent, facilitating predictability in performance.
- Path Distribution
  - Analyses show a favorable distribution of paths, which is essential for load balancing.
- Path Diversity
  - High path diversity is inherent in SF topology, contributing to robustness against failures and congestion.





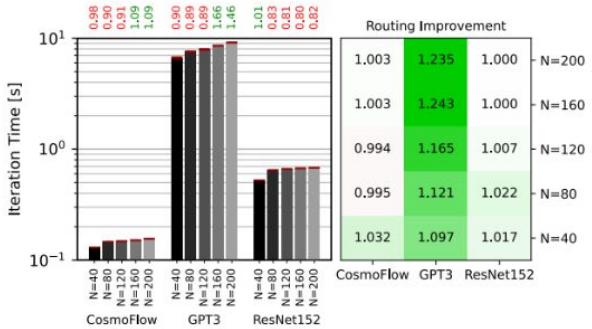
Results and Insights From Analysis

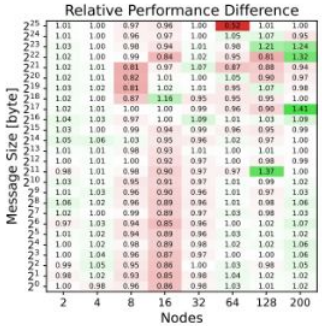
- Throughput
  - Total amount of data transmitted through the network per unit time.
- SF's max throughput outperforms standard FatPaths in most cases, indicating potential for high efficiency in data handling.



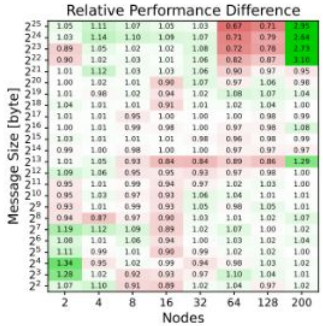
Empirical Evaluation

- Comparisons
  - Slimfly was shown to surpass traditional 2-Level Fat-Trees in both latency and throughput.
  - Slimfly was tested across diverse HPC and deep learning workloads.
- Microbenchmarks were used to present SF's raw performance on basic network tests.

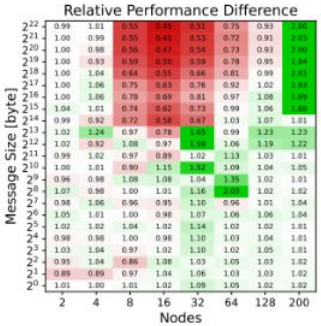




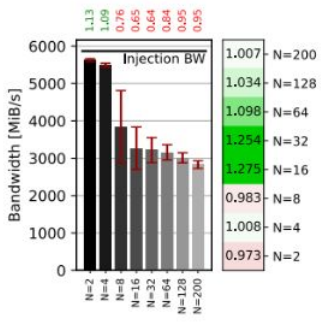
(a) MPI Bcast - SF L vs. FT



(b) MPI Allreduce - SF L vs. FT



(c) Custom Alltoall - SF L vs. FT



(d) eBB - SF L vs. FT

Empirical Results and Conclusions Made

- Comparison Benchmarks
  - SF showcases general improvements in common HPC tasks and scientific simulations over other methods
- Deep Learning Uses
  - Showed significant speedups in distributed deep neural network training over traditional topologies.
- Scalability & Cost Analysis
  - Offers generally higher scalability and cost-effectiveness, especially in larger installations, but becomes disproportionate as size and scale increase.

Questions?