THE UNIVERSITY OF CHICAGO

UNDERSTANDING NETWORK COMMUNICATION REQUIREMENTS FOR THE UPDOWN SYSTEM

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCE
IN CANDIDACY FOR THE DEGREE OF
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY
JIYA SU

CHICAGO, ILLINOIS
GRADUATION DATE

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The UpDown System is a novel fine-grained, event-driven architecture designed for irregular graph computation, that has demonstrated scalable performance to 16,384 nodes. Due to its unique event-driven mechanism, UpDown exhibits distinct computation and DRAM access patterns compared to conventional architectures such as CPUs and GPUs, thus, UpDown has different network communication characteristics. Understanding the network traffic patterns generated by UpDown applications—and how parameters such as latency and injection bandwidth impact performance—is critical for system designers to identify appropriate network configurations.

In this thesis, we simulate three irregular graph applications—Triangle Counting (TC), Breadth-First Search (BFS), and PageRank (PR)—on 1 to 512 UpDown nodes under varying network latencies and injection bandwidth constraints, and we project performance trends for a 16,384-node system. Results show that UpDown applications generates uniform traffic across nodes, with packet sizes ranging from 16 to 88 bytes. These applications produce approximately 0.5 to 2 bytes of network traffic per instruction, requiring 2–8 TB/s of injection bandwidth per node. Moreover, system performance remains stable under constrained injection bandwidth and increased network latencies of up to 4000 ns (8 times that of the hardware designed), indicating strong resilience to both transient bandwidth fluctuations and network designs that deliver longer communication delays.

CHAPTER 1

INTRODUCTION

Over the past several decades, the emergence of "internet-scale" and other big data workloads has driven a growing demand for computing systems capable of efficiently processing large-scale data, such as graphs with hundreds of billions of edges [7, 8]. To meet this growing demand, modern computing systems must scale both per-node performance and the total number of nodes. Consequently, many large-scale system [9, 10, 11, 12, 13, 14, 15, 16, 17] and cloud platforms have been built with thousands of nodes to deliver performance in the range of hundreds to thousands of PFLOPS.

Large-scale system architectures can broadly be classified into two categories: 1) Distributed memory architectures, which use MPI communication protocols between nodes —examples include Fugaku and Tianhe [9, 18]; 2) Shared memory architectures, which integrate RDMA technologies to enable high-performance memory access across nodes, such as the NVIDIA DGX GH200 system [19, 20, 21, 22, 23]. As shown in Figure 1.1, distributed memory architectures using MPI can demonstrate reasonable scalability at modest system sizes. However, as the number of nodes or cores increases, scalability bottlenecks emerge—primarily due to communication overhead. In distributed memory systems, MPI introduces substantial communication overhead, including eager polling for progress, inefficient global synchronization, and thread contention. These factors contribute to sub-linear performance scaling at scale [1, 9]. For applications with regular computation patterns, such as dense matrix multiplication—which exhibits coarse-grained parallelism and large per-thread workloads—MPI systems can scale efficiently by overlapping communication and computation through multi-threading. However, for irregular applications like Breadth-First Search (BFS) and PageRank (PR), the communication is fine-grained and data-dependent, leading to high synchronization costs and communication overhead. As a result, network traffic and thread contention dominate execution time, limiting scalability and overall per-

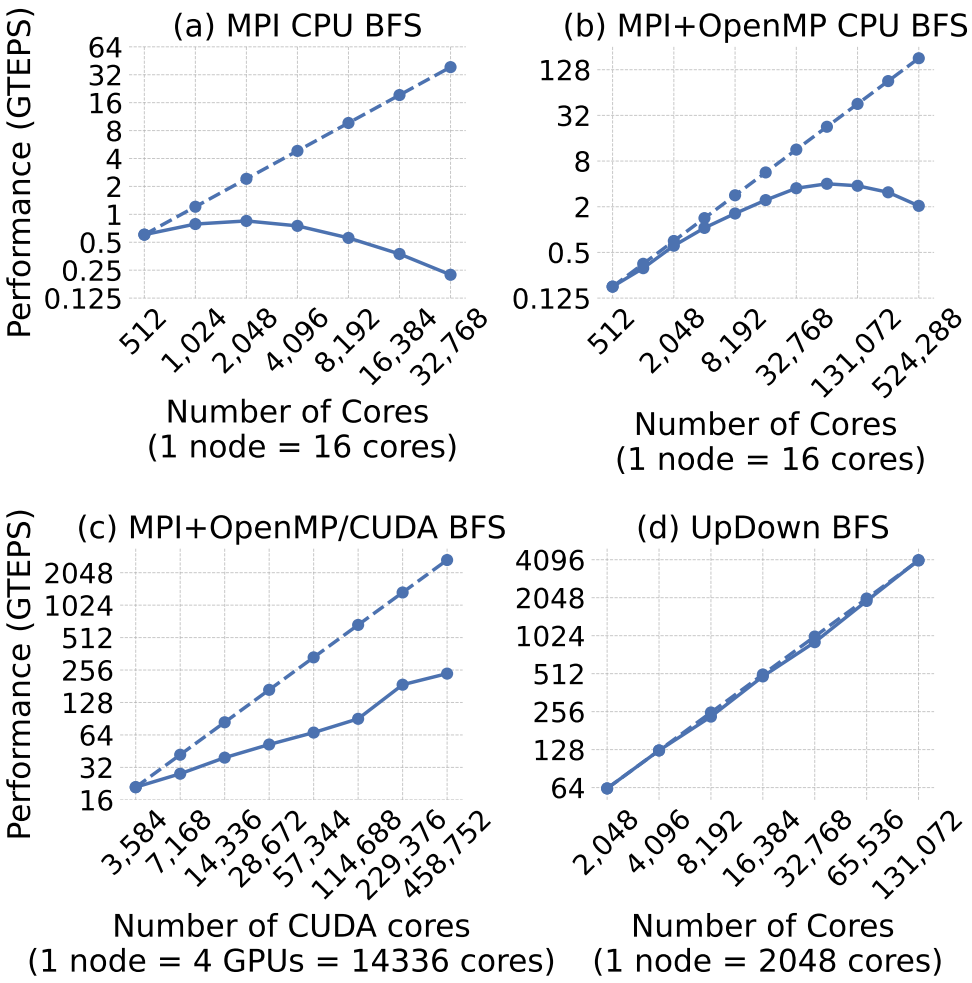

Figure 1.1: The solid line is the real performance, and the dashed line is the linear performance. (a) and (b): Weak scaling performance results using RMAT graphs of scale 25 to 35 on the Blue Gene/Q system. The MPI-CPU configuration utilizes 16 processes per node, while the hybrid configuration (MPI+OpenMP) employs 1 process per node and 16 threads per node [Adapted from [1]]. (c) Weak scaling performance with scale26 RMAT graphs on each Tesla P100 GPU up to 124 GPUs on an early access system (Ray) of LLNL's upcoming CORAL/Sierra supercomputer [Adapted from [2]]. (d) Weak scaling performance using RMAT graphs of scale 22 to 28 on the UpDown system.

formance [1]. Additionally, porting applications from single-node to distributed environments often requires substantial code modification to integrate MPI interfaces. While regular workloads such as transformer-based language models can achieve near-linear scaling across GPUs in shared memory systems with RDMA [24], training still demands significant inter-node communication, which can become a bottleneck when network bandwidth is limited [25].

Irregular graph algorithms—such as Breadth-First Search (BFS) and PageRank—further exacerbate scalability challenges due to their high inter-node traffic and page invalidation overheads [26], resulting in degraded scalability and sub-linear performance gains [27, 28].

In contrast, UpDown—a fine-grained, event-driven, globally addressable distributed memory architecture—can achieve linear scalable performance across multiple nodes (Figure 1.1(d)) without requiring any code modifications from the single-node implementation. To demonstrate this, we simulate three irregular graph applications—Triangle Counting (TC), Breadth-First Search (BFS), and PageRank (PR)—across 1 to 512 nodes, analyzing their scalability and network communication patterns. Furthermore, we conduct sensitivity studies on network latency and injection bandwidth, revealing that these applications exhibit strong resilience to both transient bandwidth fluctuations and prolonged communication delays on the UpDown system. Finally, we project performance results for a full 16,384-node UpDown system and compare its scalability and efficiency with existing large-scale systems.

The main contributions of this thesis are as follows:

1. We demonstrate that UpDown graph applications achieve scalable performance across multi-node systems with high resource utilization and no code modifications.

2. We analyze the network traffic patterns generated by UpDown applications, and show they exhibit uniform traffic across nodes with packet sizes ranging from 18 to 88 bytes—most commonly 16 B, 32 B, and 80 B—and generate 2–8 TB/s of network injection bandwidth per node.

3. We evaluate system robustness and show that UpDown maintains high performance under both transient bandwidth fluctuations and long communication latencies, demonstrating strong resilience across diverse network conditions.

4. We project the performance of a full-scale 16,384-node UpDown system, where Triangle Counting (TC), Breadth-First Search (BFS), and PageRank (PR) achieve 3,181,388

GTEPS, 1,041,354 GTEPS, and 1,542,398 GTEPS, respectively. We also estimate performance under different network injection bandwidth constraints.

We begin by introducing the background in Chapter 2, covering key concepts such as IO bandwidth, packet formats, frame forwarding rates, and large-scale system configurations. The design of the UpDown architecture and its network packet format is presented in Chapter 3. Scaling performance and communication characteristics across 1–512 nodes are evaluated in Chapter 4, followed by a sensitivity analysis of network latency and injection bandwidth in Chapter 5. We then project weak-scaling performance for configurations up to 16,384 nodes under different network assumptions in Chapter 6, and summarize key network requirements and future work in Chapter 7.

CHAPTER 2

BACKGROUND

This chapter provides an overview of the current landscape of network bandwidth capabilities, packet formats, and frame forwarding rates supported by network devices in large-scale computing systems.

2.1 I/O Channel Bandwidth and Emerging Technologies

Input/Output (I/O) Channel Bandwidth refers to the data transmission rate of communication channels, typically measured in gigabits per second (Gbps). It represents the rate at which data is transmitted through chip-to-chip interconnects, such as Ethernet, InfiniBand, Slingshot, and NVLink. The current progress in I/O channel bandwidth primarily depends on advancements in serial/parallel signal transmission technology, modulation methods, and packaging techniques.

Table 2.1: Different I/O Channel Bandwidth

Name	**Year**	**Channel Bandwidth**	**Channel latency**	**Packet Size**	**Systems**
Ethernet	2014	100-400 Gbps	2-10 us	64B-9KB	NVIDIA DGX A100/ROCEv2
InfiniBand	2018	200-400 Gbps	0.5-1 us	64B-9KB	Summit, Fugaku
Slingshot	2019	100-200 Gbps	1 us	32B-9KB	El Capitan, Frontier, Aurora
NVLink	2017	200-400 Gbps	0.5 us	16-288B	DGX GH200, GB200 NVL72

As bandwidth demands increase, I/O systems face growing challenges, including signal attenuation, elevated power consumption, and physical limitations at the package level. When transmission rates exceed thresholds such as 112 Gbps using PAM4 signaling, signal integrity deteriorates sharply, restricting PCB trace lengths to only a few centimeters. Simultaneously, each additional SerDes channel in high-speed transmission contributes significantly to power consumption. As channel bandwidth increasing, I/O channel power becomes a critical bottleneck for system-level energy efficiency. Moreover, the limited area of chip packages imposes constraints on the number of I/O channels due to wiring density

and thermal design limitations [29]. Consequently, despite differences in network architecture and protocols, the per-channel bandwidth across Ethernet, InfiniBand, and Slingshot is not substantially different, as all are subject to similar physical and power constraints.

2.1.1 Co-Packaged Optics (CPO)

Co-Packaged Optics (CPO) addresses key limitations in traditional electrical interconnects, such as bandwidth bottlenecks, and excessive power consumption. By integrating optical modules directly into computing chips or switching chips, CPO effectively overcomes signal attenuation and crosstalk issues. It also eliminates the performance degradation and energy inefficiencies associated with long copper links, thereby significantly increasing bandwidth density.

Ayar Labs has introduced the TeraPHY optical chiplet, capable of achieving 8 Tbps using a 16-wavelength SuperNova light source [30]. Broadcom has announced plans to integrate CPO into its compute ASICs, targeting 12.8 Tbps (6.4 Tbps per direction) by 2025, with an ambitious roadmap to scale up to 102.4 Tbps (51.2 Tbps per direction) by 2028 [31]. This breakthrough technology is expected to meet the growing demands of AI, HPC, and large-scale data centers. By leveraging CPO, these systems can achieve high-bandwidth, low-power, and low-latency interconnects—surpassing the physical and energy efficiency limitations of conventional electrical signaling.

2.2 Network Packet Format

Networks implement different protocols and packet formats across various OSI layers: Ethernet operates at the data link layer, IPv4/IPv6 at the network layer, and UDP/TCP at the transport layer. To reduce protocol overhead and improve data transmission efficiency, modern high-performance interconnects—such as HPE Slingshot and NVIDIA NVLink—typically employ lightweight, customized packet formats that bypass or simplify the traditional net-

work and transport layers.

2.2.1 Ethernet Packet Format

Table 2.2: Ethernet II Packet Format [4]

Layer	**Field**	**Size**	**Description**
Physical layer	Inter frame gap	Min 12B	An inter frame gap is required between Ethernet frames, meaning that a device must wait for a specified period before sending each subsequent frame. This is necessary to give the frame receiver enough time to process each frame. The length can be reduced to 64 bits for GE interfaces or 40 bits for 10GE interfaces.
	Preamble	7B	Synchronizes the receiver before actual data transmission
	Start Frame Delimiter (SFD)	1B	Indicates the start of the Ethernet frame
Link layer	Dest MAC Address (DMAC)	6B	Address of the receiving device
	Source MAC Address (SMAC)	6B	Address of the sending device
	EtherType	2B	Specifies the protocol encapsulated in the payload (e.g., IPv4 = 0x0800)
	Payload (Data)	46B–1500B	Actual data being transmitted
	Frame Check Sequence (FCS)	4B	CRC checksum for error detection

The traditional Ethernet packet format adheres to a standardized frame structure that is widely adopted in LAN and Internet communication. An Ethernet II frame consists of a 13–20 byte physical layer header and an 18-byte data link layer header, resulting in a total header size of 31–38 bytes. The data payload typically ranges from 46 to 1500 bytes, yielding a payload efficiency of approximately 54% (with a 46-byte payload) to 98% (with a 1500-byte payload), depending on the payload size.

To enhance transmission efficiency for large data transfers, Ethernet networks support *jumbo frames*—packets with a larger Maximum Transmission Unit (MTU) than the standard Ethernet frame. These frames are widely supported and commonly enabled in high-

performance NICs and switches, particularly in HPC environments. By reducing the number of packets required to transmit a given volume of data, jumbo frames help lower CPU utilization and protocol processing overhead. An MTU size of 9 KB is generally recommended for Ethernet-based HPC networks, as it enables more efficient frame handling, improved throughput, and reduced packet overhead [32]. In systems using HPE Cray interconnects, when InfiniBand is configured as the high-speed network, the default MTU is typically set to 64 KB. This configuration supports jumbo frame traffic and is considered optimal for IP over InfiniBand (IPoIB), maximizing bandwidth utilization for large-message communication [33].

2.2.2 HPE Slingshot Packet Format

Ethernet was originally designed for local and wide area networks, rather than the tightly coupled communication required in supercomputing environments. Its relatively large packet overhead, lower switching rates compared to HPC standards, and reliance on packet drops under congestion make Ethernet poorly suited for latency-sensitive protocols such as remote direct memory access (RDMA), widely used in HPC and storage systems.

HPE Slingshot [34, 35] enhances standard Ethernet by extending the Link Layer Discovery Protocol (LLDP) and introducing a packet format optimized for HPC fabrics. Slingshot supports standard Ethernet at the edge while employing optimized HPC-specific Ethernet on internal network links.

Compared to Ethernet, Slingshot packets support smaller minimum frame sizes of 32 or 40 bytes [35, 36], making them well-suited for single-word (8 B) remote memory access operations. While Slingshot's public documentation does not specify a maximum MTU, it has demonstrated effective performance with message sizes up to 128 KB [35]. To support Slingshot's flexible packet structure and associated protocols, the packet header typically ranges from 24 to 32 bytes—larger than the 13–20 byte header used in standard Ethernet II frames. As a result, the payload-to-header ratio varies significantly depending on the

message size, with payload efficiency ranging from approximately 20% for small payloads (e.g., 8 bytes of data) to nearly 99.975% for large payloads (e.g., 128 KB).

2.2.3 NVLink Packet Format

NVLink [5] provides fine-grained, high-bandwidth, and low-latency communication between processors and memory. A typical NVLink transaction consists of a request and its corresponding response. NVLink packets vary in size, ranging from a single 128-bit flit (16 bytes) to as many as 18 flits (288 bytes), enabling data transfers of up to 256 bytes.

Table 2.3: NVLink Packet [5]

Field	**Size**	**Description**
Cyclic redundancy check (CRC)	25 bits	Detect data errors
Transaction layer header	83 bits	Request type, address, flow control credits, and tag identifier
Data link layer heder	20 bits	Acknowledge identifier, packet length information, and application number tag
Address extension (AE, optional)	128 bits	AE flit contains information that should be relatively static from request to request (sticky bits), information that is command-specific, or information that alters the default value for a command type. Sticky information is transmitted when it changes and is stored on the receiver side for reuse for non-AE packets.
Byte enable (BE, optional)	128 bits	A BE flit is used for write or atomic commands, and 128 enable bits indicate the data bytes to be written on writes up to 128 bytes.
Data payload	0-256B	Between 0 and 16 128-bit data payload flits

NVLink packets transmit data in units of 16 bytes (128-bit flits). Excluding AE (Address Extension) and BE (Byte Enable), the packet header occupies only 16 bytes, with data payloads ranging from 0 bytes up to 256 bytes. Consequently, the payload can constitute up to approximately 94% of the packet (16-byte header plus 256-byte data payload). NVLink employs a highly compact packet header format relative to other contemporary network protocols.

2.3 Packet Frame Forwarding Rate

The packet forwarding rate of a network NIC or switch is a key indicator of its processing capacity. It represents the number of packets the device can handle per second, typically measured in millions of packets per second (Mpps) or millions of messages per second (Mmps).

2.3.1 HPE Slingshot CASSINI NIC and Rosetta Switch

Cassini [35] is a 200 Gbps HPC NIC ASIC developed by HPE. The host interface is PCIe Gen4 and supports extended speed mode. The network link port conforms to the Ethernet standards for 200 Gbps (4×50 Gbps PAM4) or 100 Gbps (4×25 Gbps NRZ).

Figure 2.1 illustrates remote memory access (RMA) bandwidth as a function of data size in message for 64 processes per NIC, measured using a SHMEM performance test on a single Cassini NIC in the Shadow system. For 8-byte data payloads, a Cassini NIC achieves a packet forwarding rate of 189 million packets per second (Mpps), corresponding to an effective port data bandwidth of 12.1 Gbps (60.5 Gbps including a 32-byte packet header). For 64-byte frames, the built-in frame generator records a forwarding rate of 155 Mpps, yielding a port data bandwidth of 79.4 Gbps (119 Gbps with 32-byte headers). For 1024-byte data payloads, the forwarding rate is 22.8 Mpps, corresponding to 186 Gbps effective bandwidth (192 Gbps with headers included).

2.3.2 NVLink4 and NVLink4 Switch

NVSwitch4 is an NVLink switch chip that integrates 18 NVLink4 ports, internally organized as an 18×18 fully connected crossbar [37]. This architecture enables any port to communicate with any other at the full NVLink speed of 50 GB/s, yielding an aggregate switch bandwidth of 900 GB/s. Each NVLink4 port provides 25 GB/s per direction. Each GPU can be connected via up to 18 NVLink4 interfaces to switches or directly to other GPUs, offering a

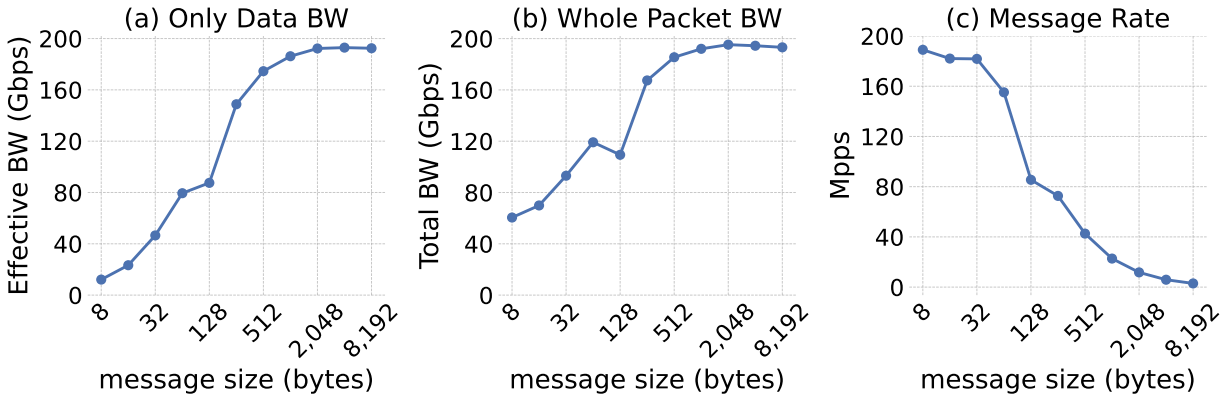

Figure 2.1: Remote memory access bandwidth and message rate as a function of data payload size in message for 64 processes per NIC on the Shadow system. [Adapted from [3]].

total bidirectional bandwidth of 900 GB/s (450 GB/s per direction).

As publicly available data on NVSwitch frame forwarding rates is limited, the theoretical forwarding rate is estimated based on NVLink's port bandwidth and packet sizes. For the smallest packet size of 16 bytes, each port can achieve a maximum forwarding rate of approximately 3,125 million packets per second (Mpps). For 64-byte packets, the estimated forwarding rate is approximately 781 Mpps per port. For 256-byte packets, the estimated forwarding rate is approximately 195 Mpps per port.

2.3.3 51.2T TH5-Bailly

Broadcom's TH5-Bailly [31] is a state-of-the-art 51.2 Tbps Ethernet switch that leverages Co-Packaged Optics (CPO) technology to address the growing demands for high-bandwidth, low-power network interconnects in domains such as artificial intelligence (AI) and high-performance computing (HPC). The switch supports 128 $\times$ 400 Gbps FR4 optical connections. Assuming a packet size of 64 bytes, the theoretical packet forwarding rate per port is approximately 781 million packets per second (Mpps), resulting in an aggregate packet forwarding rate of 100 billion packets per second (Bpps) for the entire switch.

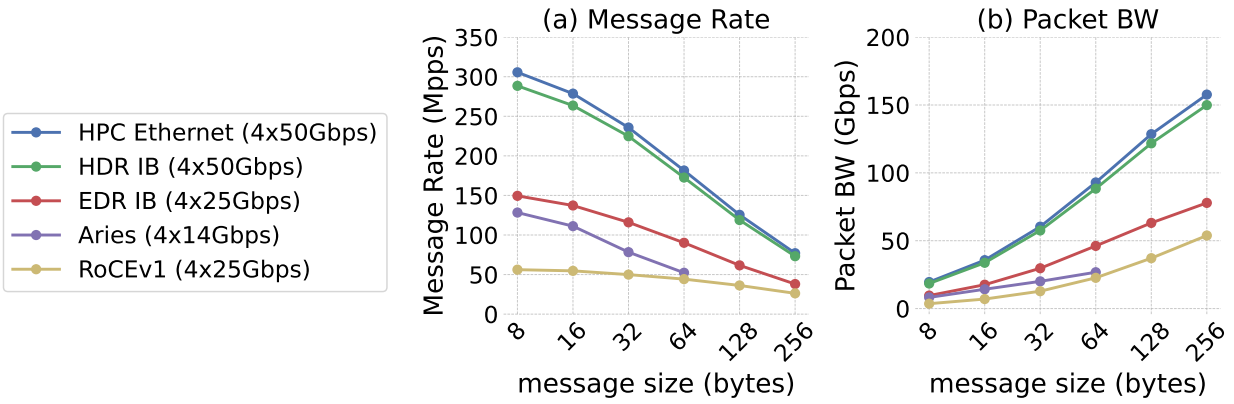

Figure 2.2: Message rate vs. message size for various high-speed interconnect technologies as of 2019. [Adapted from [3]]. **HPC Ethernet** refers to enhanced Ethernet (e.g., Slingshot) optimized for HPC with adaptive routing and low-latency transport. **HDR InfiniBand** delivers 200 Gbps using PAM4 signaling and low-latency RDMA, suitable for modern HPC and AI workloads. **EDR InfiniBand** provides 100 Gbps using NRZ, widely adopted in earlier HPC systems. **Aries** is Cray's proprietary interconnect, utilizing a Dragonfly topology with adaptive routing for low-latency communication. **RoCEv1** enables RDMA over Layer 2 Ethernet, requiring a lossless fabric and typically constrained to a single subnet.

2.3.4 Other HPC Interconnect technologies

We also evaluate message rate and packet-level bandwidth across other HPC interconnect technologies. As shown in Figure 2.2, their message forwarding rates range from 25 to 300 million packets per second (Mpps). Beyond link-level bandwidth constraints, message rate efficiency is also influenced by message size. As illustrated in Figure 2.2 and Figure 2.1(b), smaller message sizes do not lead to proportionally higher message rates and instead result in substantially lower packet bandwidth. This demonstrates that bandwidth utilization efficiency—defined as packet bandwidth (i.e., the product of message rate and message size)—declines for small messages due to protocol overhead, serialization delays, per-packet processing costs, and so on.

2.4 Large-scale systems

In this section, we summarize the network configurations and interconnect technologies used in various large-scale high-performance computing systems.

El Capitan [10, 11], Frontier [12], and Aurora [13] are the top three supercomputing systems in the world as of 2024. Fugaku [14, 15] remains the highest-performing non-GPU-based system and achieves the best Breadth-First Search (BFS) performance according to the TOP500 list [6]. Table 2.4 summarizes key architectural parameters of these systems, including core count, peak FP64 performance, memory bandwidth, and network injection bandwidth—both per node and at system scale. In addition, NVIDIA offers several GPU-based rack-scale systems with significantly higher inter-node bandwidth, as listed in Table 2.5. The top three supercomputing systems—El Capitan [10, 11], Frontier [12], and Aurora [13]—all employ HPE Slingshot interconnect technology. Fugaku utilizes Fujitsu's custom Tofu-D interconnect, while NVIDIA's GPU-based rack-scale systems use NVLink for intra-rack communication.

2.4.1 One Node Configuration

The following section compares the ratios between compute resources—measured as peak FP64 performance (excluding tensor cores)—memory bandwidth, and network injection bandwidth on a single compute node across several large-scale systems. Two key metrics are used for comparison: (1) the ratio of network to memory bandwidth, and (2) the amount of network traffic generated per 1,000 FP64 operations.

As shown in Table 2.4, El Capitan, Frontier, and Aurora deliver between 177 and 250 TFLOPS of peak FP64 performance, with memory bandwidths ranging from 13 to 21 TB/s per node. Despite their substantial computational and memory capabilities, these GPU-based systems provide only 100–200 GB/s of network injection bandwidth per node —representing just 0.5% to 1% of local memory bandwidth. As a result, the bandwidth

available for remote memory access is significantly lower than that of on-node memory, potentially constraining scalability for data-intensive applications. In addition, the average network traffic per 1,000 FP64 operations is 0.4 B, 0.5 B, and 1.13 B for El Capitan, Frontier, and Aurora, respectively. These values reflect a relatively low volume of network communication per unit of computation, underscoring the risk of network bottlenecks in both compute-bound and memory-bound workloads.

Fugaku is a purely CPU-based system, delivering 36.9 TFLOPS of FP64 performance per node. Due to its lower per-node compute and memory bandwidth, combined with a comparatively higher network injection bandwidth, Fugaku exhibits a higher ratio of network-to-memory bandwidth (4%) and a significantly higher network traffic rate—13 B per 1,000 FP64 operations. Nonetheless, the overall injection bandwidth remains relatively low compared to on-node resources.

NVIDIA's scale-up GPU-based platforms provide much higher network injection bandwidth per node: 450 GB/s with NVLink 4 and 1.8 TB/s with NVLink 5. The ratio of network-to-memory bandwidth improves from 5% (Grace Hopper) to 11% (Blackwell), and the network traffic per 1,000 FP64 operations is approximately 15–20 B—higher than current top500 supercomputer systems, but still lower than required for high-communication workloads

2.4.2 Whole Machine System Comparison

Most Top500 scalable machine systems adopt the Dragonfly network topology using 64-port HPE Slingshot network switches. Table 2.4 presents the system injection bandwidth, bisection bandwidth, BFS performance and energy for various systems. For Frontier, although the switch port design allows up to 50% bisection bandwidth, not all inter-group ports are utilized, resulting in an effective bisection bandwidth of only 30% of the system injection bandwidth. Aurora achieves a bisection bandwidth of 0.69 PB/s, which corresponds to 32.5% of its total system bandwidth. According to the Fugaku system slides [17], Fugaku reaches

only 94 TB/s of bisection bandwidth, accounting for merely 1.5% of the system bandwidth. But the network bandwidth inside one group is much higher. For NVIDIA systems, both DGX GH200 and GB200 NVL72 employ an all-to-all fat-tree topology, in which the bisection bandwidth is effectively half of the system's total network traffic capacity.

In addition, we incorporate Breadth-First Search (BFS) performance data from Graph500 [38]. Fugaku has consistently held the top BFS performance on RMAT scale 43 since 2020. Frontier currently ranks fourth, while Aurora ranks sixth. Despite delivering only one-quarter of the FP64 performance of Frontier and Aurora, Fugaku achieves 10× higher BFS performance, highlighting its architectural efficiency for graph-centric workloads.

Table 2.4: Top500 HPC Systems [6]

	El Capitan [10, 11]	Frontier [12, 39]	Aurora [13]	Fugaku [14, 15, 16, 17]
Compute Engines	4 AMD APU MI300A	1 AMD EPYC CPU 4 AMD MI250X GPU	2 Intel Xeon Max 9470 CPU 6 Intel Max 1550 GPU	12 Fujitsu A64FX CPU
Node CPU cores	96 @1.8GHz	64 @2GHz	104 @2GHz	576 @2GHz
Node GPU CUs	912 @2.1 GHz	880 @1.7 GHz	768 @900MHz	
Peak FP64 Perf	250.8 TFLOPS	212 TFLOPS	176.9 TFLOPS	36.9 TFLOPS
Node Memory	512GB HBM3	512GB HBM2e(GPU) 512GB DDR4(CPU)	768GB HBM2e(GPU) 128GB HBM2e(CPU) 1TB DDR5(CPU)	384GB HBM2
Node Memory Bandwidth	21.2 TB/s	12.8 TB/s HBM2e(GPU) 205 GB/s DDR4(CPU)	19.66 TB/s HBM2e(GPU) 2.87 TB/s HBM2e(CPU) 0.56 TB/s DDR5(CPU)	12 TB/s
Network Type	Slingshot	Slingshot	Slingshot	Infiniband
Node Ports	4	4	8	72
Node Injection Bandwidth	100 GB/s	100 GB/s	200 GB/s	490 GB/s
Network/Memory Bandwidth	0.46%	0.76%	0.99%	3.98%
Network Traffic Per 1k FP64	0.40 B	0.47 B	1.13 B	13.28 B
Nodes	11,136	9,408	10,624	13,248
CPU cores	1,069,056 @1.8GHz	602,112 @2GHz	1,104,896 @2GHz	7,630,848 @2GHz
GPU CUs	10,156,032 @2.1GHz	8,279,040 @1.7GHz	8,159,232 @900MHz	
Peak FP64 Perf	2,793 PFLOPS	1,994 PFLOPS	1,879 PFLOPS	488 PFLOPS
System Memory	5.4PB HBM3	4.6PB HBM2e(GPU) 4.6PB DDR4(CPU)	8.16PB HBM2e(GPU) 1.36PB HBM2e(CPU) 10.9PB DDR5(CPU)	4.85PB HBM2
System Ports	44,544	37,632	84,992	953,856
System Injection Bandwidth	1.09 PB/s	0.90 PB/s	2.12PB/s	6.19 PB/s
Bisection Bandwidth		0.27 PB/s (30%)	0.69 PB/s (32.5%)	94 TB/s (1.5%)
Network Topology	Dragonfly	Dragonfly	Dragonfly	Tofu Interconnect D (6D mesh/torus)
Network Switch	HPE Slingshot (64 ports @ 200 Gbps)	HPE Slingshot (64 ports @ 200 Gbps)	HPE Slingshot (64 ports @ 200 Gbps)	Tofu Network Router (10 ports with 2 lanes)
Power	34.8 MW (Peak)	8-30 MW	60 MW	30-40 MW
BFS GTEPS		29,654.6	24,250.2	204,068

Table 2.5: NVIDIA HPC Systems and UpDown

	NVIDIA DGX GH200 [19, 20, 21, 22]	NVIDIA GB200 NVL72 [40, 41]	UpDown
Compute Engines	1 Grace CPU 1 Hopper H200 NVL GPU	1 Grace CPU 2 Blackwell B200 GPU	1 CPU 2048 UpDown Lanes
Node CPU cores	72 @3.1GHz	72 @3.1GHz	
Node GPU SMs/ UpDown Accelerators	132 @1365 MHz	528 @1665 MHz	32 @2GHz
Peak FP64 Perf	30 TFLOPS	90 TFLOPS	4.1 TFLOPS
Node Memory	144GB HBM3e(GPU) 96GB HBM3(GPU) 480GB LPDDR5X(CPU)	384GB HBM3e(GPU) 480GB LPDDR5X(CPU)	512GB HBM3e
Node Memory Bandwidth	4.9 TB/s HBM3e(GPU) 4TB/s HBM3(GPU) 0.5TB/s LPDDR5X(CPU)	16 TB/s HBM3e(GPU) 0.5TB/s LPDDR5X(CPU)	9.6 TB/s HBM3e
Network Types	NVLink	NVLink	
Node Ports	18	36	
Node GPU Injection Bandwidth	450 GB/s	1.8 TB/s	6.4 TB/s (assume)
Network/Memory Bandwidth	4.94%	11.25%	66.7%
Network Traffic Per 1k FP64	15 B	20 B	1561 B
Nodes	256	36	16,384
CPU cores	18,432 @3.1GHz	2,592 @3.1GHz	
GPU SMs/Accs	33,792 @1365 MHz	19,008 @1665 MHz	524,288 @2 GHz
Peak FP64 Perf	7.68 PFLOPS	3.24 PFLOPS	67.11 PFLOPS
System Memory	36TB HBM3e(GPU) 24TB HBM3(GPU) 120TB LPDDR5X(CPU)	13.5TB HBM3e(GPU) 17PB LPDDR5X(CPU)	8PB HBM3e
System Ports	4608	1296	
System GPU Injection Bandwidth	112.5 TB/s	65 TB/s	102.4 PB/s
System Bisection Bandwidth	56.25 TB/s	32.5 TB/s	51.2 PB/s
Network Topology	2-level Fat Tree	1-level Fat Tree	
Network Switch	NVLink 4 Switch	NVLink 5 Switch	
Power	256 KW	86 KW	9.64 MW
BFS GTEPS			935,000

CHAPTER 3

UPDOWN SYSTEM

UpDown [42, 43, 44, 45, 46] is a large-scale parallel system design initiative funded under IARPA's Advanced Graphic Intelligence Logical Computing Environment (AGILE) program [47]. The overall objective of the program is to achieve breakthrough performance on irregular applications and graphs characterized by extreme skew and low data reuse.

In this chapter, we first provide an overview of the UpDown design, highlighting its unique architectural features that are critical for the subsequent network performance analysis. We then extend the discussion from a single-node design to a multi-node system and introduce the UpDown network packet format. Next, we briefly introduce the UDKVMSR [44] (UpDown Key-Value pair Map Shuffle Reduce) framework, which is used for graph application mapping, along with the implementations of Triangle Counting (TC), Breadth-First Search (BFS), and PageRank (PR) built on UDKVMSR. Finally, we describe *Fastsim2*, a cycle-accurate, fast, multi-threaded simulator capable of simulating up to 256 UpDown nodes efficiently.

3.1 UpDown Architecture

The UpDown system is composed of multiple UpDown nodes, where each node consists of one CPU, 32 UpDown accelerators, and 8 HBM3e DRAM stacks, as illustrated in Figure 3.1. Each UpDown accelerator is positioned between the CPU and the DRAM, enabling both the CPU and the UpDown accelerators to directly access the memory. An individual UpDown accelerator contains 64 event-driven programmable lanes, with each lane equipped with a 64 KB software-controlled scratchpad memory. These lanes are capable of generating unlimited event messages and issuing direct DRAM accesses.

Events are first-class primitives in the UpDown ISA, represented using *event-words*, which

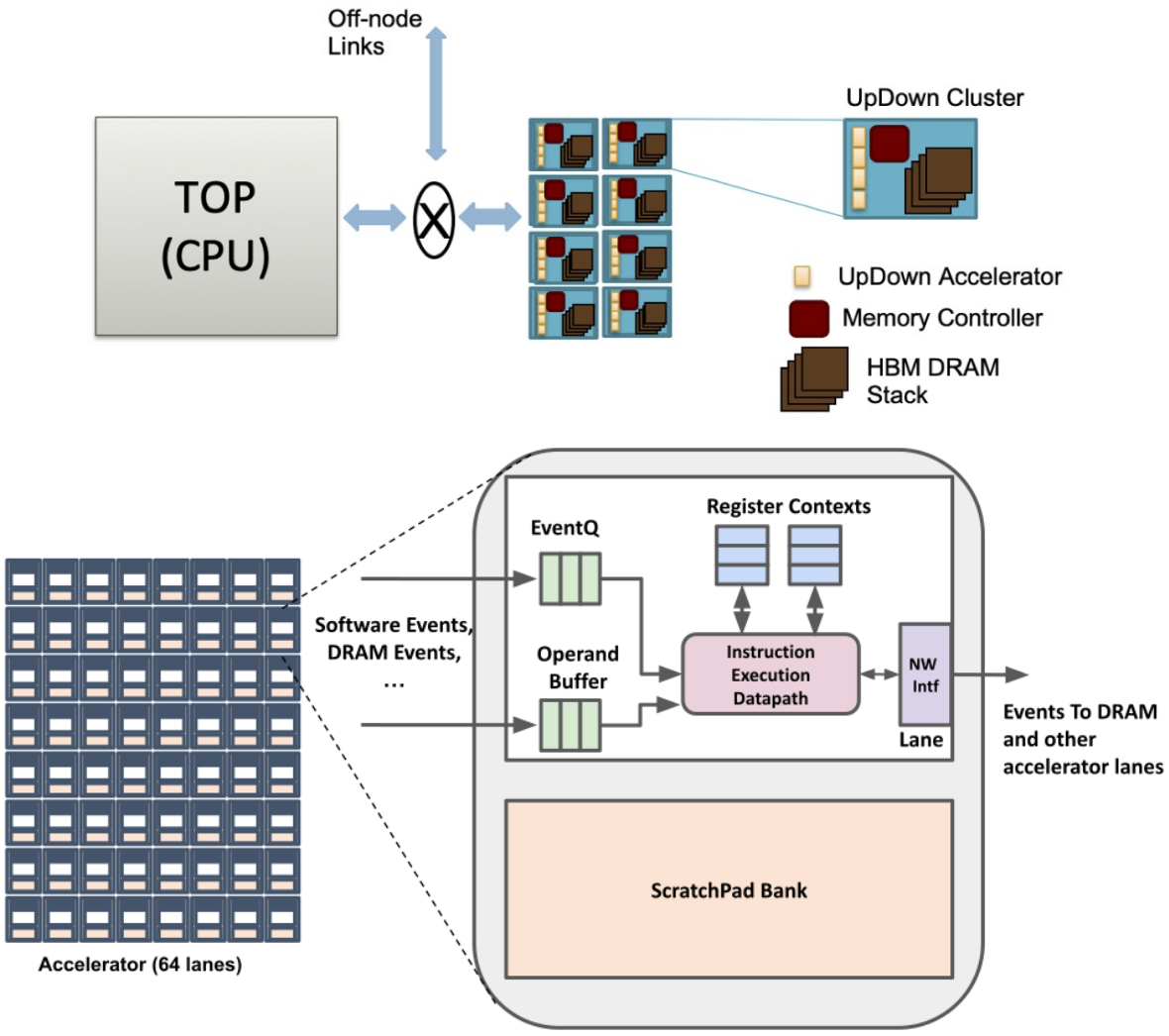

Figure 3.1: UpDown Node has 1 CPU, 32 UpDown accelerators with uniform access to 8 HBM stacks. Each UpDown accelerator has 64 event-driven programmable lanes.

combine traditional hardware events such as `DRAM_load` and `DRAM_load_ack` and custom software-defined events under a unified programmable event framework. As illustrated in Figure 3.2, an *event-word* encodes the following information: compute name (event_label, 20 bits), thread ID (8 bits), compute location (network ID, 32 bits), and message size (num_operands, 3 bits). Both software and hardware events are captured through this unified interface and managed using specific ISA instructions [42], as summarized in Table 3.1. These event messages facilitate communication between UpDown lanes and the

DRAM.

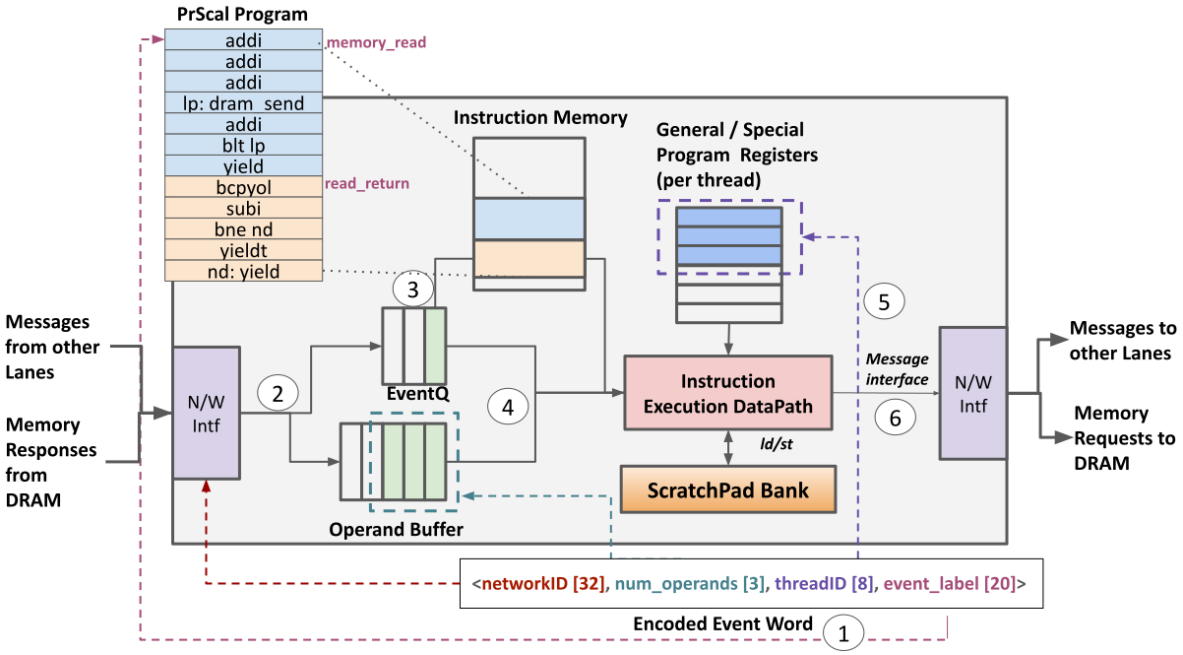

Figure 3.2: Event Driven Execution on UpDown lane

Table 3.1: UpDown Instruction Set Architecture Highlights

Category	**Execution Mechanisms and Instructions**
Low-latency non-blocking thread/event	Thread spawn/invoke: Msg arrival event, send msg; Thread yield/destroy: (`yield, yieldt`); Register naming for Msg Operands, use in add-class, send, and load-store (scratchpad memory) operations
Split-transaction DRAM	DRAM send instructions (`sendm, sendmops, sendmr`); DRAM responses come to software-defined event label
Messaging (incl DRAM)	Single instruction sends of 1-8 words: `send, sendr, sendops`, `sendm` (DRAM), `sendmops` (DRAM)

Specifically, the `send` instruction carries an *event_word*, a *continuation_word*, and a payload of up to nine data words (one word = 8B). Using the `send` instruction, a new thread (with `threadID` 0xFF) can be launched on a specified lane (designated by `networkID`) to execute from a given `event_label`, all encoded within the *event_word*. The *continuation_word* specifies continuation-passing information using the same encoding format. DRAM accesses, performed using the `sendm` instruction, follow similar semantics; however, the *event_word* is replaced by a physical memory address, while the *continuation_word*

specifies the `event_label` for response synchronization. The details of *event_ word*, physical DRAM address, and `networkID` encoding are summarized in Table 3.2.

Table 3.2: Event Word and Address Encoding

Event Word (64 bits)		
Name	**Bit Field**	**Description**
networkID	[63:32]	Lane identifier where the event should be sent.
threadID	[31:24]	Thread ID to be activated in the lane. A value of 0xFF creates a new thread.
thread_mode	[23]	Defines the register map to be used (regular/streaming mode).
num_ops	[22:20]	Number of operands in the current event.
event_label	[19:0]	Address offset of the instruction block to be executed.
Physical Address (64 bits)		
Name	**Bit Field**	**Description**
reserved	[63:60]	Reserved bits.
nodeID	[59:45]	Node identifier associated with the address.
physical_addr	[44:0]	Physical DRAM address.
NetworkID (32 bits)		
Name	**Bit Field**	**Description**
top/ud	[31]	Always 0 to indicate UpDown Accelerator networkID.
reserved	[30]	Reserved bit.
send_policy	[29:27]	Policy for dynamic management of message targets.
NID	[26:11]	Node ID.
UDID	[10:6]	Accelerator ID.
LID	[5:0]	Lane ID.

Due to previous unique architecture features, UpDown include the following key features that generate short event message with low overhead:

- **Low Latency Event / Thread:** hardware event queues, hardware event-driven thread scheduling, hardware-supported thread create;schedule;terminate.
- **Short Message:** Direct ISA messaging for 1-8 words operands by registers with 2 event words for destination and continuation.
- **Non-blocking Event Scheduler:** Each DRAM access send one to eight-word load/store operations to DRAM, asynchronous return and trigger another event by continuation word. So there is no blocking inside event.

- **Flexible Programmability:** An instruction set that realizes customized and general computation efficiently

3.2 UpDown System

The UpDown system consists of a compute array comprising 16,384 UpDown nodes, along with a separate set of input/output (I/O) nodes responsible for external networking and storage. The compute array is organized into 32 racks, with each rack housing 512 nodes. In addition to the compute racks, there are 8 I/O and system management racks, containing a total of 28 I/O nodes.

In this paper, we focus exclusively on the compute array and its internal system network. Memory within an UpDown node is divided into local memory and global memory. Local memory is accessible only by the node itself, whereas global memory is shared across the entire UpDown system. Consequently, there are two types of network messages: computation messages, where a lane sends an event to a remote lane, and remote global memory accesses, where a node accesses shared global memory residing on a remote node.

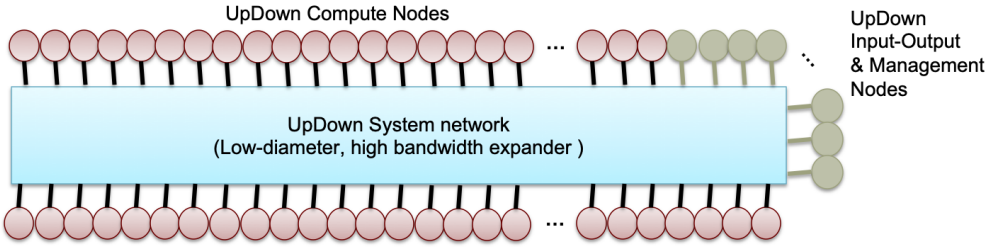

Figure 3.3: UpDown System consists of a compute array (16,384 UpDown nodes) and a separate set of input/output nodes that connect external networking and storage.

3.3 UpDown System Network Design

UpDown accelerators and TOP CPUs can send messages directly to the system network, as illustrated in the node diagram (Figure 3.2), and can also receive incoming messages from the

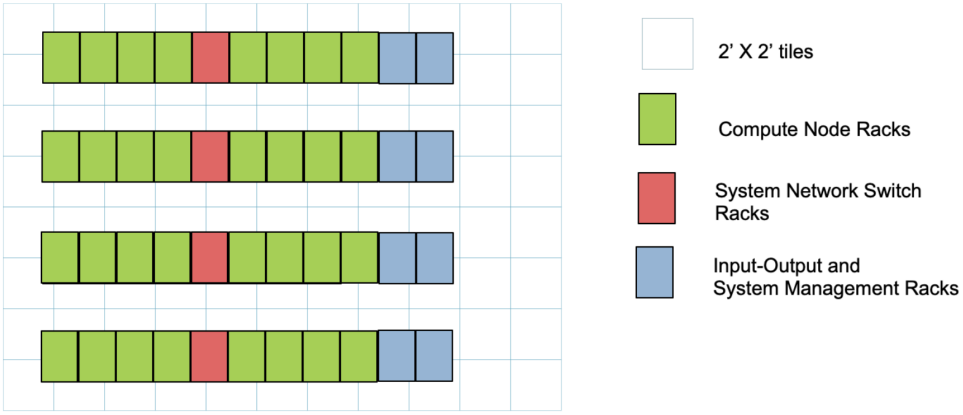

Figure 3.4: Floor Plan for UpDown System: Compute and Input/Output nodes and approximate layout. The system has 44 racks and occupies 256 sqft.

network. Because both system-level and intranode packet formats are standardized, packets can flow directly from the lanes of one UpDown node to those of another without requiring format conversion.

The UpDown network infrastructure adds only speed matching and flow control to handle longer internode and interrouter link delays, as well as to manage the transition from short electrical links to longer optical internode links. As a result, direct messaging enables communication within a node, between nodes, and to global DRAM locations anywhere in the UpDown system.

3.3.1 Network Packet Size

UpDown message packets consist of two main types of communication: cross-node lane-to-lane compute communication and remote DRAM access. Remote DRAM operations include DRAM load, DRAM load acknowledgment, DRAM store, and DRAM store acknowledgment. Further details are summarized in Table 3.3. The destination node ID can be extracted either from the event word or from the DRAM address, as described in Table 3.2.

Table 3.3: UpDown Packet Description

Packet Type	**Packet Size (B)**	**Description**
Compute	32-88	destination event (8B) + continuation event (8B) + 2-9 operands (16B–72B)
DRAM load	16	destination DRAM address (8B) + continuation event (8B)
DRAM load ack	24-80	destination DRAM address (8B) + continuation event (8B) + 1-8 read data (8B–64B) (usually 64B when reading continual data)
DRAM store	24-80	destination DRAM address (8B) + continuation event (8B) + 1-8 write data (8B–64B) (usually 64B when writing continual data)
DRAM store ack	16	destination DRAM address (8B) + continuation event (8B)

3.4 UDKVMSR

The Key-Value Map-Shuffle-Reduce (UDKVMSR) model is a key-based programming framework implemented on UpDown. It enables programmers to explicitly control the binding of `map` and `reduce` tasks to specific compute resources.

Triangle Counting (TC), Breadth-First Search (BFS), and PageRank (PR) are all implemented using UDKVMSR. As shown in Table 3.4, each vertex is randomly assigned to a thread on a lane. During the `map` phase, the neighbor list of the current vertex v is accessed, and each edge (v, u) is sent to a designated `reduce` lane based on a hash function—typically using $hash(u)$ or $hash(v, u)$ as the reduce key.

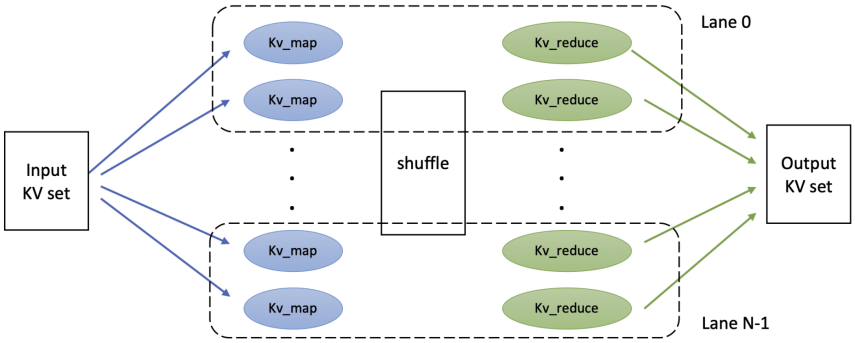

Figure 3.5: Key-value map-shuffle-reduce framework on UpDown

Table 3.4: Benchmark Applications.

App	Implementation			Lines Of Code
	Function	Map	Reduce	
PR	*for v in Graph G :* *for u in v.neighbor :* $u.value \mathrel{+}= 1$	**Input:** v *for u in v.neighbor :* *send event* $(u, 1)$ *to lane hash(u)*	**Input:** $u, 1$ $u.value \mathrel{+}= 1$	1184
BFS	*for* $(v, parent)$ *in Queue* q_d : *if not v.visit :* *update* $v.parent, v.distance, v.visit$ *for u in v.neighbor :* $q_{d+1}.push(u, v)$	**Input:** $v, parent, d$ *if not v.visit :* *update* $v.parent, v.distance, v.visit$ *for u in v.neighbor :* *send event* (u, v) *to lane hash(u)*	**Input:** $u, parent$ $q_1.push(u, parent)$	1291
TC	*for v in Graph G :* *for u in v.neighbor :* *if* $v > u$: $Intersect(v.neighbor, u.neighbor)$ $(v > u > intersect.vid)$	**Input:** v *for u in v.neighbor :* *if* $v > u$: *send event* (v, u) *to lane hash(v,u)*	**Input:** v, u $Intersect(v.neigh, u.neigh)$ $(v > u > intersect.vid)$	1448

3.5 UpDown Simulator

Fastsim2 is a cycle-accurate simulator of the UpDown accelerator elements, incorporating a simple memory model that includes both latency and bandwidth constraints. It separates computation and communication phases into two distinct parallel loops and parallelizes the simulation of UpDown instructions and communication using OpenMP across UpDown accelerators. This design achieves a simulation speed of approximately 150 MIPS, which is about 300× faster than GEM5. This allows us to execute application runs of 1.38e13 instructions.

Fastsim2 also includes a simplified HBM memory model, which imposes realistic limitations on DRAM access latency and DRAM channel bandwidth. Validation against GEM5 shows a simulation accuracy difference of less than 10%. As a result, Fastsim2 can accurately simulate millions of UpDown lanes, providing a solid foundation for studying UpDown network behavior at multi-node scales.

For the UpDown injection network study, we extend Fastsim2 to add a fixed network latency for network messages and constrain the total outgoing network traffic per node

during each communication phase. The UpDown system network design achieves a one-way maximum latency (across three hops) of 550 ns. The latency of each individual network link and switch is estimated at 183 ns, including time-of-flight delays of up to 120 ns.

To model network latency and throughput limitations, a queue is implemented on each node to temporarily store outgoing network messages. In each cycle, only messages that have been queued for at least the specified network latency (in cycles) are eligible for transmission. Furthermore, the total size of transmitted network packets in each cycle must not exceed the injection bandwidth limit; otherwise, messages will remain queued until sufficient bandwidth becomes available. This mechanism ensures that both latency and bandwidth constraints are accurately enforced at the packet send-out stage.

Table 3.5: UpDown System Configuration

Component	**Specification**
UpDown Lane	256 HW threads, 64KB scratchpad, in-order core @2GHz
DRAM Memory Stack	8x channels HBM3e @1.2TB/s (100ns access latency)
UpDown Node	2048 UpDown lanes + 8 DRAM Stacks @9.6TB/s
UpDown System	16,384 UpDown nodes + 550ns network latency

CHAPTER 4

SYSTEM NETWORK TRAFFIC CHARACTERISTIC

4.1 Summary of Network Traffic Characteristic

Network traffic characteristics encompass communication pattern and network capacity both of which play a crucial role in overall system performance and scalability. Point-to-point network traffic and bisection traffic ratio define the requirements of an UpDown network topology and routing, influencing data flow efficiency and load balancing across network links. The network traffic injection bandwidth determines the necessary port and link capacity to ensuring smooth data transmission and minimizing congestion from compute part to network. Additionally, the network packet injection rate and packet size influence the frequency of routing decisions, impacting line speed decision requirements, queuing behavior in switches and overall network efficiency.

Table 4.1 summarizes the key performance metrics used in the following sections to evaluate both program and network behavior.

Table 4.1: Metrics and Description

Application and System Metrics	**Description**
Inst Cycles/Time	Number of cycles or simulated time in which each lane actively executes instructions
Exe Cycles/Time	Number of total cycles/time during the program simulation
Normalized Exe Time	Execution time on multiple nodes normalized by one-node time
Utilization	Fraction of instruction runtime spent in execution, formula 4.1
Network Metrics	**Description**
Normalized Network Traffic	Network traffic normalized by average, formula 4.2
Coefficient of Variation (CoV)	Ratio of the standard deviation to the mean, formula 4.4
Bisection Traffic	Network traffic crosses the midpoint of a system when it is divided into two equal halves
Ideal Bandwidth	The highest bandwidth program can achieve without any overhead, formula 4.8
Self-Inclusive Traffic	Include both local and remote network traffic, formula 4.9. Detail explanation in Section 4.5.1
Instantaneous Bandwidth	Average bandwidth in a really short of time (such as 100 cycles, 50ns)

4.2 System-level workload

In this chapter, the workload is defined along two dimensions: three graph applications and different input graph sizes. We report both per-application results and average performance. Section 4.2.1 describes the characteristics of the input graphs, while Section 4.2.2 outlines the properties of the applications. All experiments simulate the applications across various input sizes (RMAT s19-s28) and various system sizes (1 to 512 UpDown nodes) using Fastsim2.

4.2.1 Input Dataset Characteristics

Each application is evaluated on following input graphs generated using the Graph500 RMAT specification [48], with dataset details provided in Table 4.2. RMAT graphs are synthetic and designed to exhibit consistent sparsity and skewed degree distributions. We adopt the standard Graph500 generation parameters ($a = 0.57$, $b = c = 0.19$, and an average degree 16), which are commonly used to benchmark graph application performance at scale [38].

Table 4.2: Input Graph Dataset

Graph-Datasets	**Apps**	**#Vertices**	**#Edges**	**#Avg Deg**	**#Connected Vertices**	**#Avg Connected deg**
RMAT-scale19	TC	525K	14.5M	27.6	291K	49.8
RMAT-scale20	TC	1.05M	29.5M	28.1	557K	53.0
RMAT-scale21	TC	2.1M	60.1M	28.6	1.1M	54.6
RMAT-scale22	TC, BFS, PR	4.2M	122M	29.0	2.0M	61.0
RMAT-scale23	TC, BFS, PR	8.4M	247M	29.5	3.9M	63.8
RMAT-scale24	TC, BFS, PR	16.8M	500M	29.8	7.4M	67.7
RMAT-scale25	TC, BFS, PR	33.6M	1011M	30.1	14.1M	71.7
RMAT-scale26	BFS, PR	67.1M	2039M	30.4	29.4M	69.4
RMAT-scale27	BFS, PR	134.2M	4108M	30.6	56.8M	72.3
RMAT-scale28	BFS, PR	268.4M	8269M	30.8	108.9M	75.9

RMAT [48] graphs generated with probabilities a=0.59, b=0.19, c=0.19, deg=16

Given their similar structural properties, graph applications are expected to exhibit comparable performance across these RMAT inputs. Previous large-scale system studies, such as [1], have employed RMAT graphs in weak scaling experiments.

Compared to other synthetic graph models such as Erdős–Rényi and Forest Fire [49], as well as large-scale real-world graphs like Twitter, RMAT graphs demonstrate a higher degree of skew in their degree distributions. This skew poses a significant challenge for load balancing in parallel graph processing and provides a useful stress case for studying network traffic characteristics in graph applications.

4.2.2 Application Characteristics

We evaluate three widely studied graph applications—Triangle Counting, Breadth-First Search (BFS), and PageRank—as representative workloads for analyzing network performance. These applications are implemented using code developed by researchers in the UpDown project [43, 44, 45, 46], as part of the AGILE U.S. Government research program [47]. A summary of each algorithm is provided in Table 3.4.

All applications are built on the UDKVMSR framework (described in Section 3.4), which is designed for scalable, irregular graph processing. These workloads exhibit irregular computation and input patterns, scale efficiently, and sustain high system utilization across deployments of up to 512 nodes. Moreover, the applications differ significantly in their network behavior—including packet size, communication-to-computation ratios, and network bandwidth requirements—making them suitable for evaluating a range of network performance characteristics.

Strong Scaling Performance.

Figure 4.1 shows the strong scaling performance of PageRank (PR), Triangle Counting (TC), and Breadth-First Search (BFS) on up to 512 nodes. Lines are color-coded by input graph size: green represents runs on the largest graph, while blue corresponds to the smallest.

All three applications demonstrate near-linear speedup, with larger input graphs achieving better scaling due to higher workload and reduced relative overhead. Interestingly, BFS

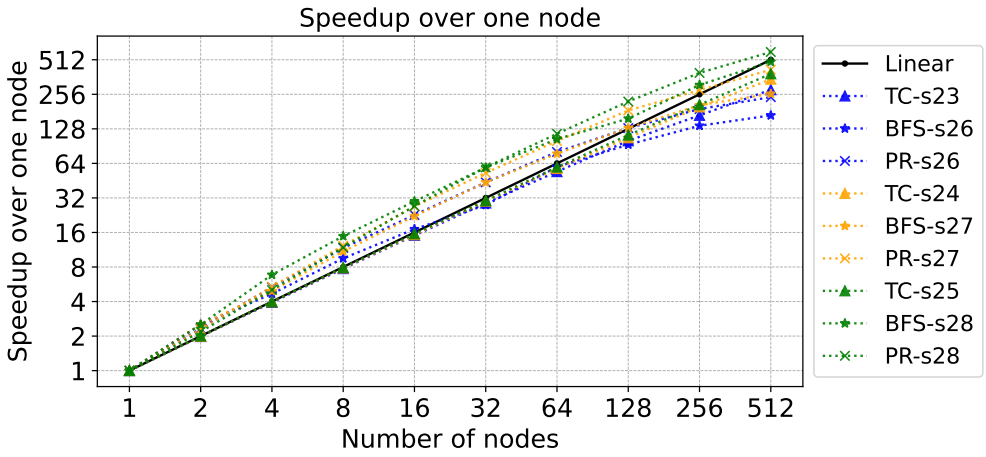

Figure 4.1: Application speedup over one nodes across 1-512 nodes (strong scaling).

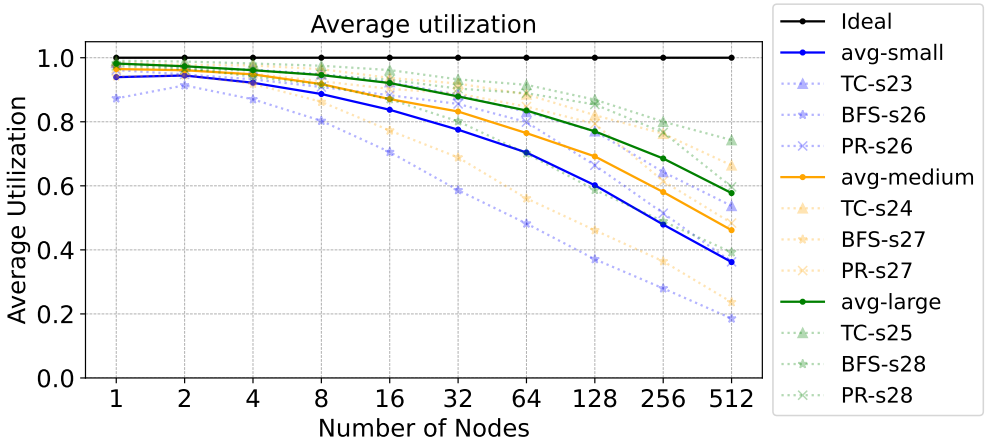

Figure 4.2: Application average lane utilization across 1-512 nodes (strong scaling).

and PR exhibit superlinear speedup at lower node counts. This behavior is attributed to increased effective software cache capacity: both applications leverage the scratchpad memory on each node as a cache for visited vertices and updated values. As the number of nodes increases, the aggregate scratchpad capacity grows proportionally, reducing cache conflicts and miss rates—eventually reaching a point where the entire working set can reside within the collective cache.

In addition to near-linear speedup, all three applications demonstrate high hardware utilization, quantified as the ratio of active execution cycles per lane to the total number of

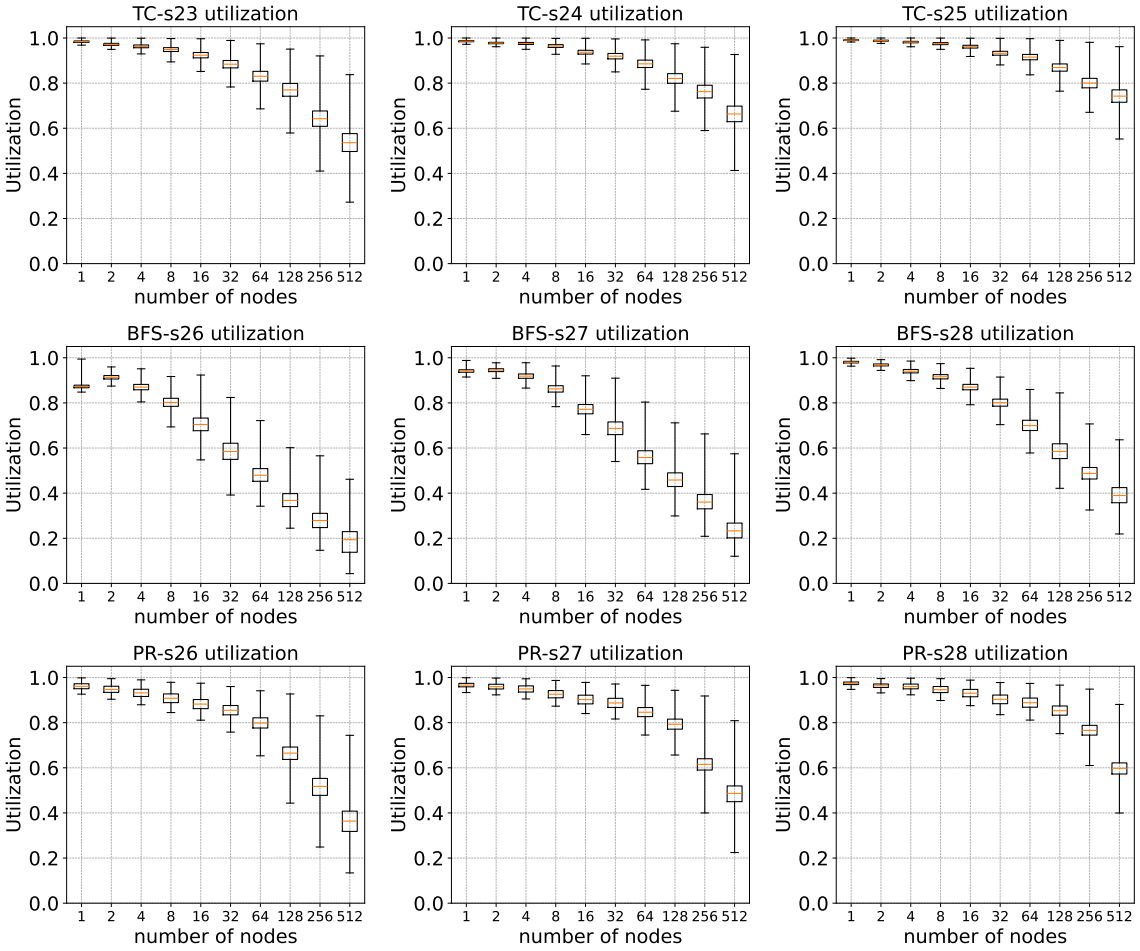

Figure 4.3: Strong Scaling Utilization on TC, BFS, PR.

simulated cycles.

$$\text{Utilization} = \frac{\text{Inst Cycles}}{\text{Exe Cycles}} \times 100\% \tag{4.1}$$

All applications achieve high compute utilization (over 80%) when running on 1–16 nodes. For larger machine configurations, average utilization remains above 50% on the largest input graph. The decline in average utilization at larger scales is attributed to two main factors: (1) reduced workload per node, which limits compute parallelism and makes it harder to hide latency, and (2) workload imbalance. To mitigate load imbalance, high-degree vertices

in the input graph are split into multiple sub-vertices. However, when the workload is not sufficiently large, imbalance still persists.

Figure 4.3 illustrates the utilization of PageRank (PR), Triangle Counting (TC), and Breadth-First Search (BFS) across three input graphs. The maximum utilization reflects performance loss caused by reduced workload per compute unit, which limits the ability to hide latency. The upper and lower bounds indicate the extent of load imbalance.

Both Figure 4.2 and Figure 4.3 demonstrate that increasing the input graph size leads to higher scaling speedup and utilization. The largest graphs simulated—TC-s25, BFS-s28, and PR-s28—are constrained by wall-clock time and memory limitations. It is expected that all applications could achieve over 90% utilization on 512 nodes if the input graph size were sufficiently large. Such larger studies are the topic of future research as UpDown simulation tools continue to improve. To demonstrate it, we also run a weak scaling on 1-64 UpDown nodes.

Weak Scaling Performance.

To isolate the effects of cache capacity and ensure sufficient workload per node, we also conduct weak scaling experiments. The key idea in weak scaling is to maintain a constant workload per node by keeping the number of vertices and edges mapped to each node fixed as the system scales. The corresponding input graph sizes and node counts used for these experiments are summarized in Table 4.3.

It is important to note that only PageRank and Breadth-First Search (BFS) demonstrate true weak scaling behavior. Triangle Counting (TC), by contrast, does not represent a strict case of weak scaling. While PageRank and BFS operate on workloads that scale linearly with the number of vertices and edges, TC involves computing set intersections between the neighbor lists of vertex pairs—a task whose computational complexity does not scale linearly with graph size. As a result, TC exhibits workload growth per node even under weak scaling

configurations, and therefore does not achieve ideal weak scaling.

Table 4.3: Weak Scaling Input Graphs

Nodes	**1**	**2**	**4**	**8**	**16**	**32**	**64**
TC	s19	s20	s21	s22	s23	s24	s25
PR	s22	s23	s24	s25	s26	s27	s28
BFS	s22	s23	s24	s25	s26	s27	s28

The performance results are presented in Figure 4.4, with corresponding utilization shown in Figure 4.5. Triangle Counting (TC), Breadth-First Search (BFS), and PageRank (PR) exhibit similar per-node performance as the machine size increases.

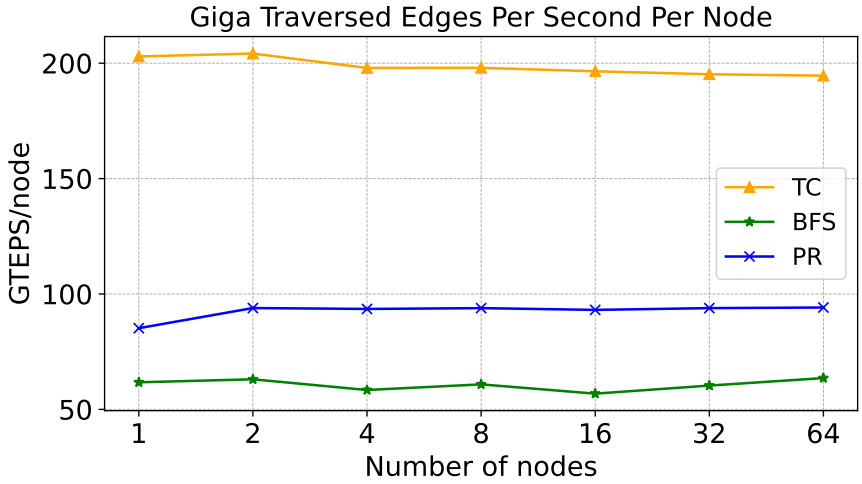

Figure 4.4: Weak Scaling Performance (Giga Traversed Edges Per Second Per Node).

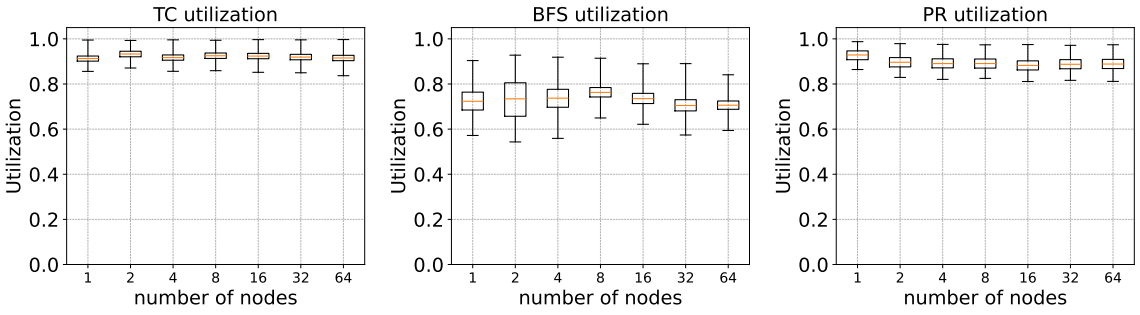

Figure 4.5: Weak Scaling Utilization on TC, BFS, PR.

All three applications achieve high hardware utilization. TC and PR maintain an average utilization of around 90%, while BFS achieves approximately 70%. These results demonstrate

the strong scalability of the UpDown architecture when each node is provided with sufficient workload. In Chapter 6, we will project the performance of TC, BFS, and PR up to 16,384 nodes, using the weak scaling results from 1 to 64 nodes as the foundation.

Different Network and Program Characteristics.

In addition to exhibiting good scaling performance, the three applications have distinct program characteristics. They generate different types of network messages, vary in packet size, and differ in network communication and memory usage ratios.

This section presents data collected from runs on 32 nodes, as the characteristics remain similar across other node sizes.

Figure 4.6 and 4.7 illustrates the distribution of different network message types. In Triangle Counting (TC), network traffic is primarily dominated by remote DRAM loads, whereas in Breadth-First Search (BFS) and PageRank (PR), lane-to-lane compute communication accounts for the majority of network traffic. As discussed in Section 3.3.1, the DRAM load packet size is 16B, while the acknowledgment packet size ranges from 24B to 80B. Assuming each DRAM load reads the maximum 80B, the average DRAM load packet size is approximately 48B ((16+80)/2), which aligns closely with TC's average packet size in Table 4.4. To minimize network traffic, most lane-to-lane compute communications in BFS and PR use the minimum 32B packet size, resulting in an average packet size close to 32B.

Although BFS and PR exhibit similar network message type ratios and average packet sizes, their program communication rate (communication instructions per kilo-instruction) and memory access rate (memory instructions per kilo-instruction) in Table 4.4 differ significantly. PR has a higher network communication frequency, with approximately one lane-to-lane compute communication generated for every 20 instructions on average. In contrast, BFS exhibits a higher memory access rate.

Figure 4.6: Network Message Types Ratio on 32 nodes (upper:count, lower:bytes).

4.3 Communication Pattern Analysis

This section analyzes the topological connectivity of UpDown applications, including point-to-point traffic and bisection traffic. Understanding these factors helps network designers optimize network topology and routing mechanisms to enhance performance and efficiency.

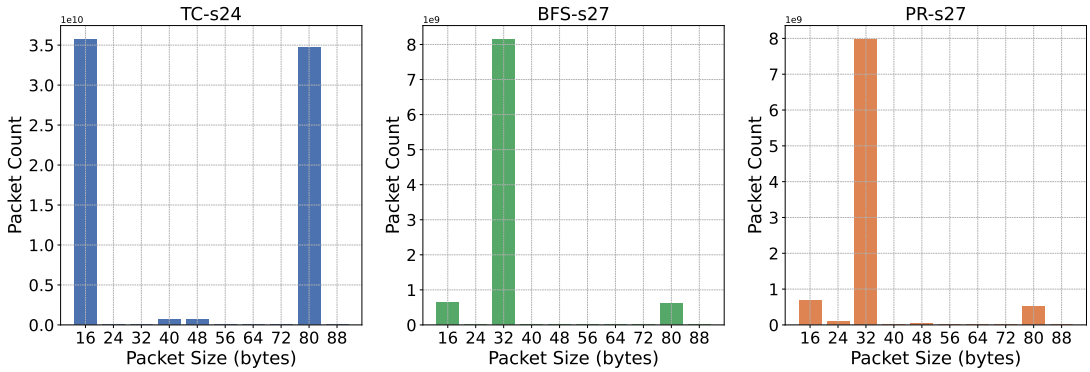

Figure 4.7: Packet Size Histogram on 32 nodes.

Table 4.4: Breakdown of Program Characteristics on 32 nodes. (KI: Kilo Instructions)

Applications	**Avg Packet Size**	**Commu Inst Per KI**	**Mem Inst Per KI**	**Traffic(B) Per Inst**
TC-s23	47.34 B	0.12	7.05	0.65
TC-s24	47.48 B	0.09	6.97	0.65
TC-s25	47.59 B	0.07	6.90	0.64
PR-s26	33.52 B	52.82	4.82	2.08
PR-s27	33.53 B	53.22	4.80	2.10
PR-s28	33.54 B	53.48	4.75	2.10
BFS-s26	33.94 B	33.51	9.50	1.32
BFS-s27	33.95 B	40.15	6.81	1.58
BFS-s28	33.96 B	41.83	6.17	1.65

4.3.1 Point-to-Point Traffic

The fine-grained architectural design and programming model of UpDown promote balanced task distribution across compute units, resulting in similar communication patterns among them. Furthermore, UpDown's two-level memory translation mechanism ensures that global memory is uniformly distributed across machines. As a result, random global memory accesses from compute units produce statistically uniform point-to-point traffic patterns.

Figures 4.8, 4.9, and 4.10 show the traffic matrices for 128 nodes, which are generally uniform. However, in the case of Triangle Counting (TC), over 10% of the average network traffic is concentrated between specific node pairs. This imbalance stems from skewed data access patterns—particularly, certain vertices and their neighbors are accessed much more frequently in TC.

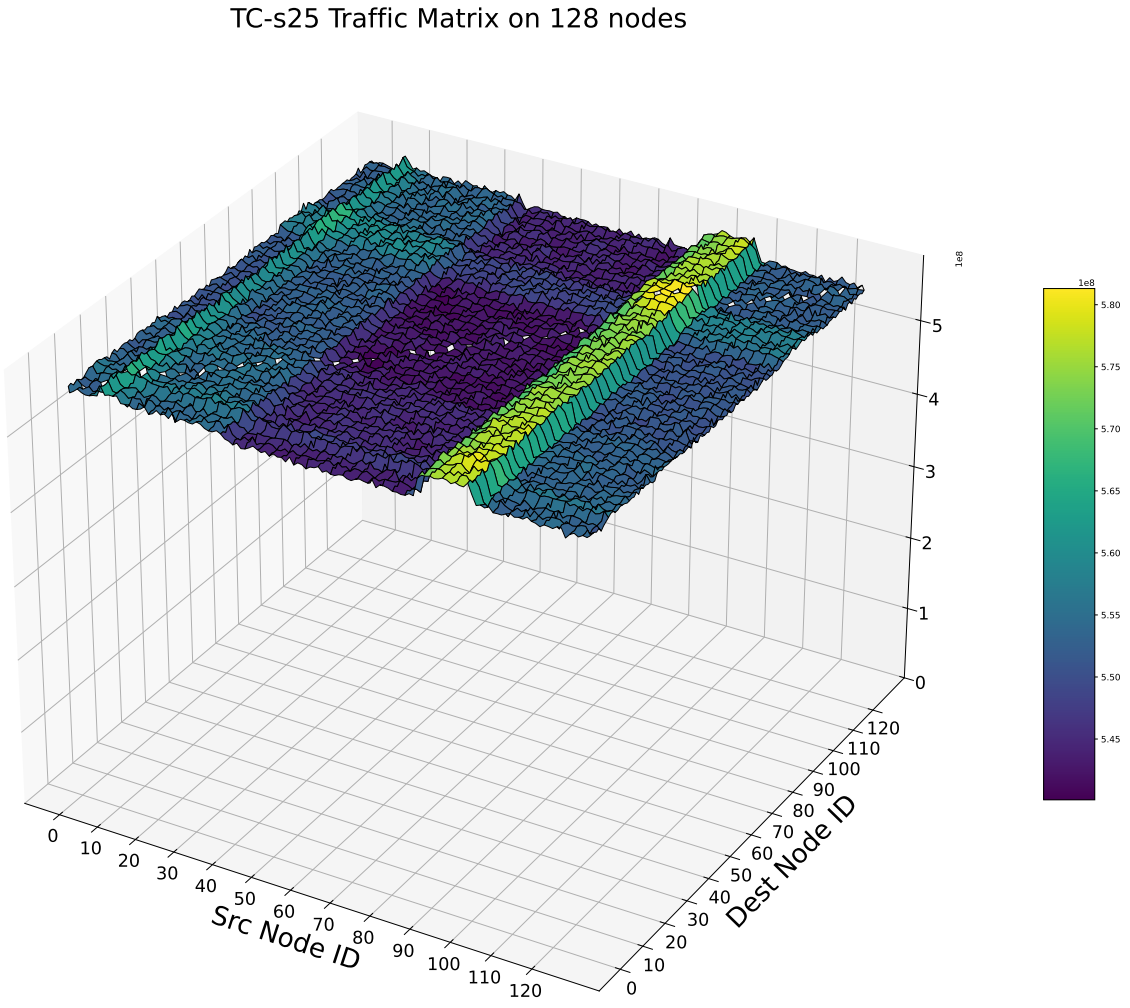

Figure 4.8: TC-s25 Network Traffic Matrix on 128 nodes.

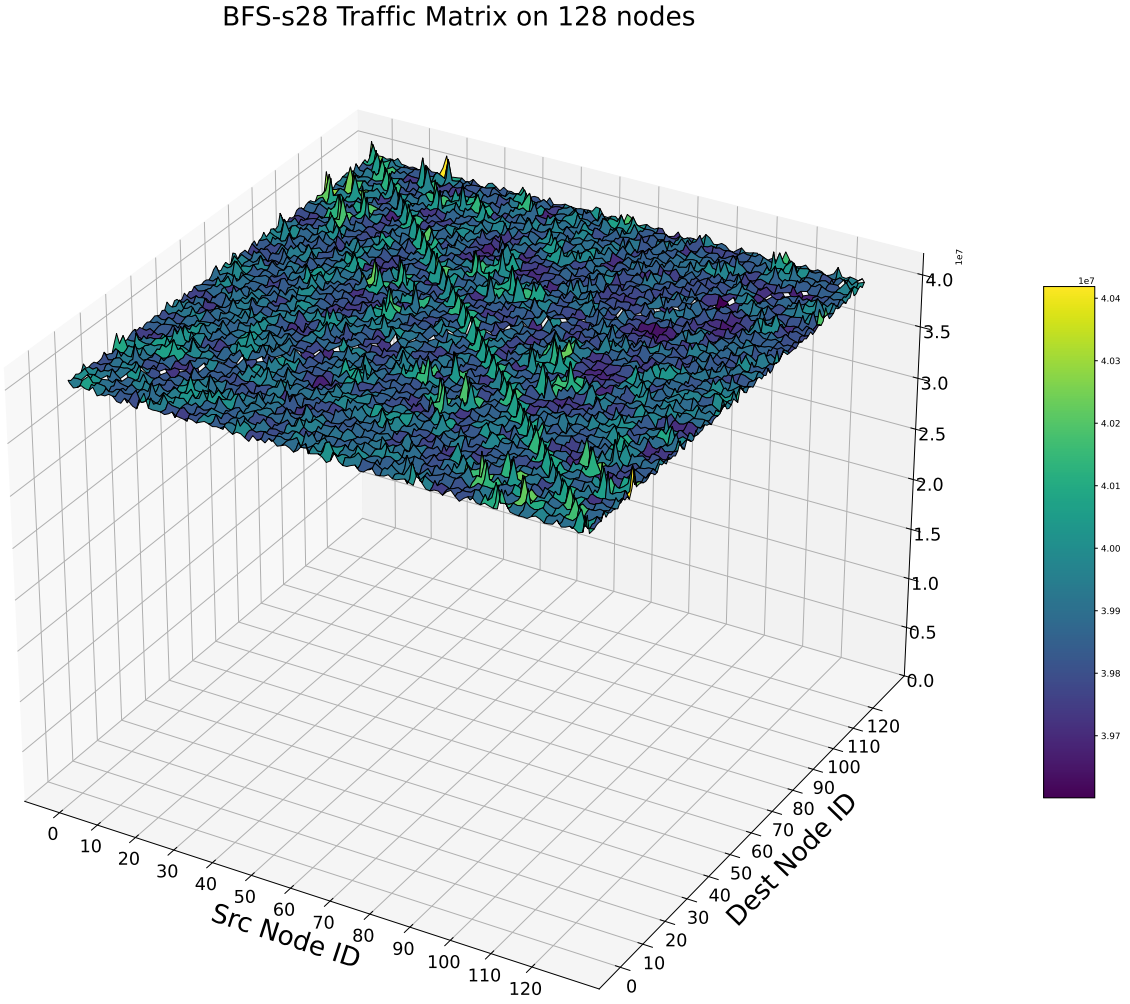

Figure 4.9: BFS-s28 Network Traffic Matrix on 128 nodes.

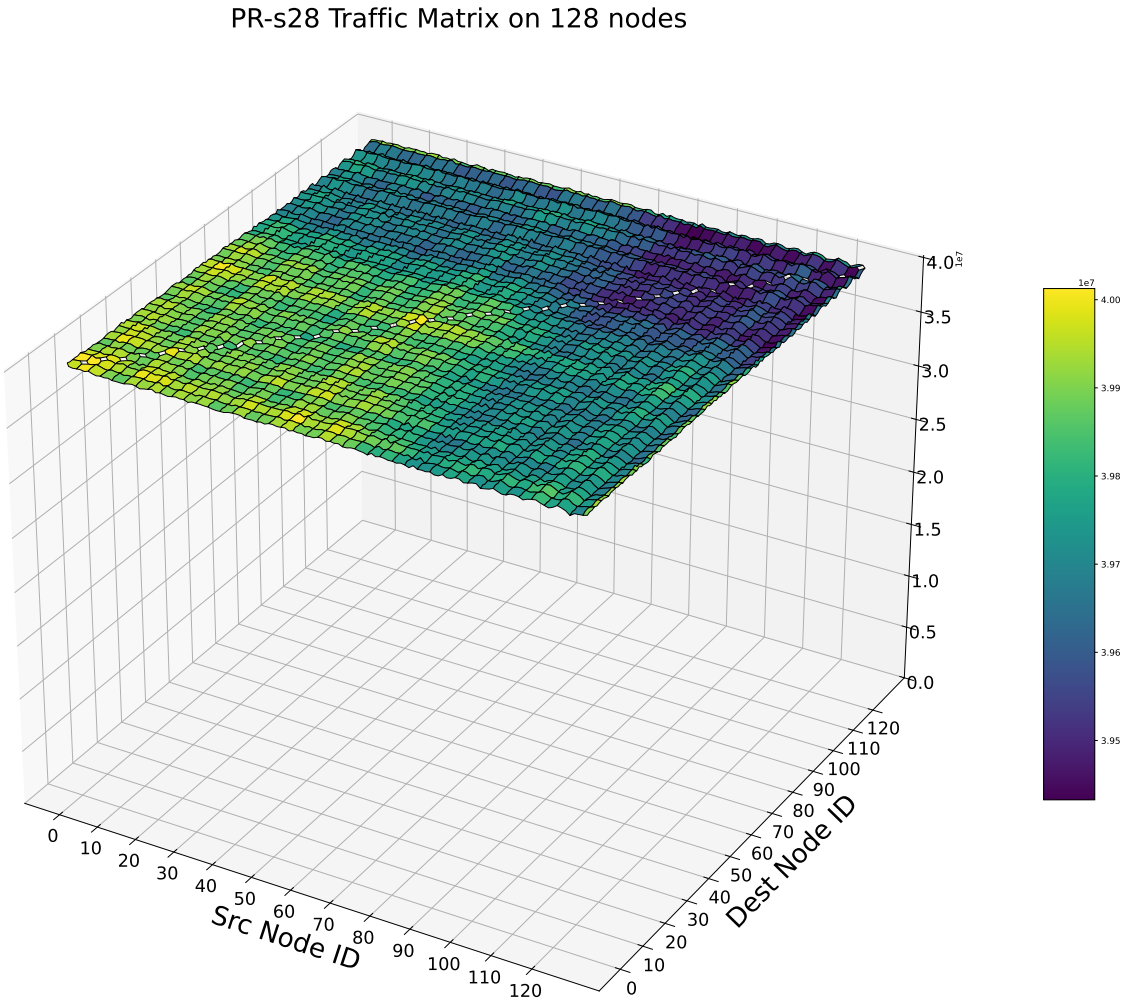

Figure 4.10: PR-s28 Network Traffic Matrix on 128 nodes.

A more detailed explanation reveals that TC's network traffic is dominated by remote DRAM reads and their associated read acknowledgments. In Figure 4.8, nodes 87 to 100 receive a disproportionately high volume of DRAM read requests and send more read acknowledgments, even though their instruction counts are similar to other nodes. This imbalance persists as the system scales from 128 to 512 nodes, with nodes 87 to 100 consistently experiencing higher volumes of remote DRAM reads.

To illustrate the distribution of point-to-point traffic across different machine sizes, we present a box plot of the normalized point-to-point traffic (as defined in Formula 4.2) under the strong scaling regime, spanning 1 to 512 nodes (Figure 4.11). In contrast, weak scaling shows a similar traffic distribution, with only minor variation (less than 5%), as it covers a narrower range—from 1 to 64 nodes—resulting in relatively stable communication patterns.

$$\bar{x_{ij}} = \frac{x_{ij}}{\mu} \tag{4.2}$$

$$\mu = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} x_{ij} \tag{4.3}$$

For strong scaling, as the number of nodes increases, the variance in point-to-point traffic also increases. This is primarily due to reduced workload and DRAM data per node, which exacerbates load imbalance. However, this imbalance can be mitigated by increasing the input graph size, which helps distribute the workload more evenly across nodes.

To more directly capture the variability in point-to-point traffic, we compute the Coefficient of Variation (CoV)—a normalized measure of dispersion defined as the ratio of the standard deviation to the mean, which is also the population standard deviation of normalized value.

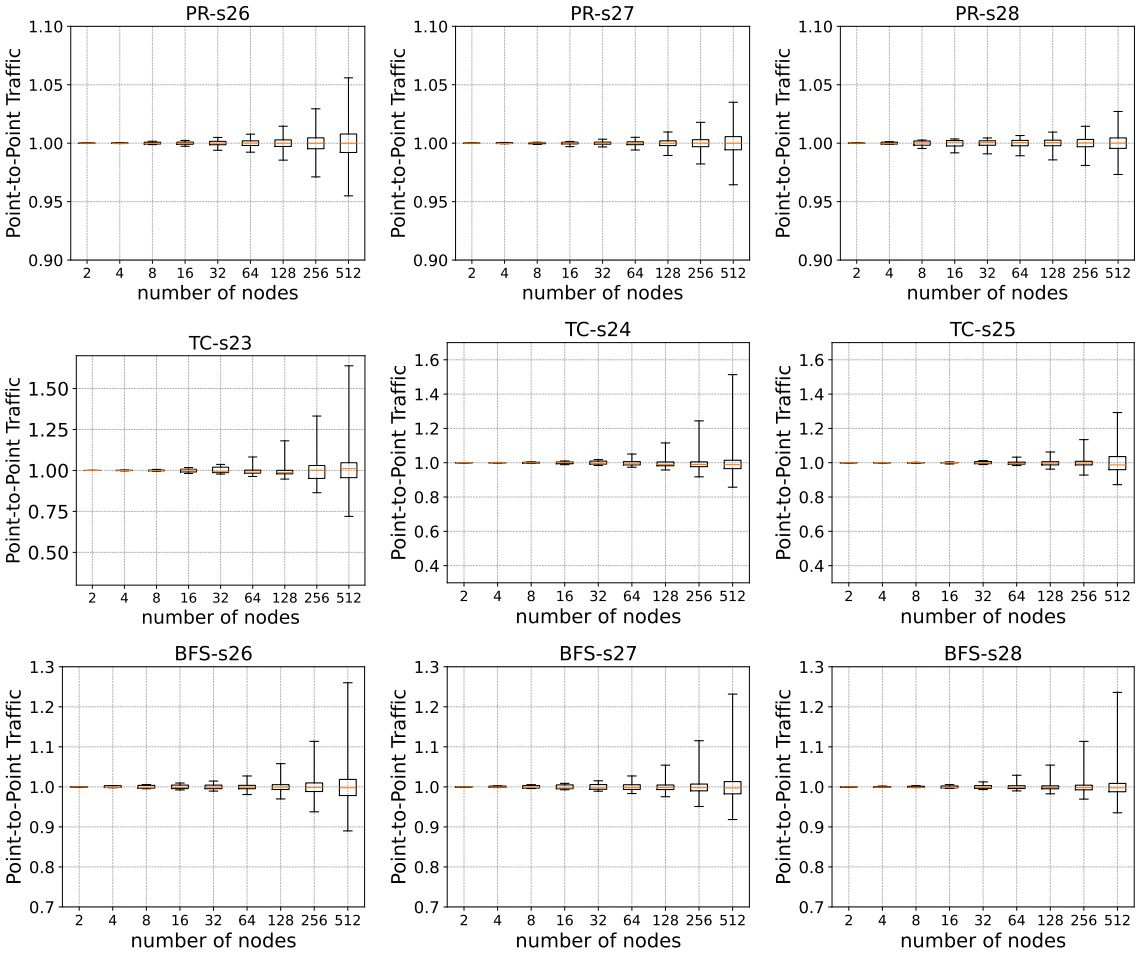

Figure 4.11: Normalized Point-to-Point traffic distribution (strong scaling).

$$CoV = \frac{\sigma}{\mu} \times 100\% \tag{4.4}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{4.5}$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{4.6}$$

The CoV provides a consistent basis for comparing variability across datasets with different mean values. In the context of network traffic, a low CoV indicates uniform distribution

across node pairs, while a high CoV signals significant imbalance or potential congestion.

Figure 4.12 presents the CoV of point-to-point communication traffic for different applications across varying input sizes and machine counts. Each application is evaluated using three input graph sizes. In the plot, the blue line represents the smallest inputs, the yellow line corresponds to medium-sized inputs, and the green line reflects the largest inputs. As input sizes increase, the CoV decreases and approaches zero (indicated by the black reference line) as the machine count grows. This trend suggests that, given sufficiently large input datasets, traffic distribution becomes increasingly uniform—approaching the ideal case where CoV = 0.

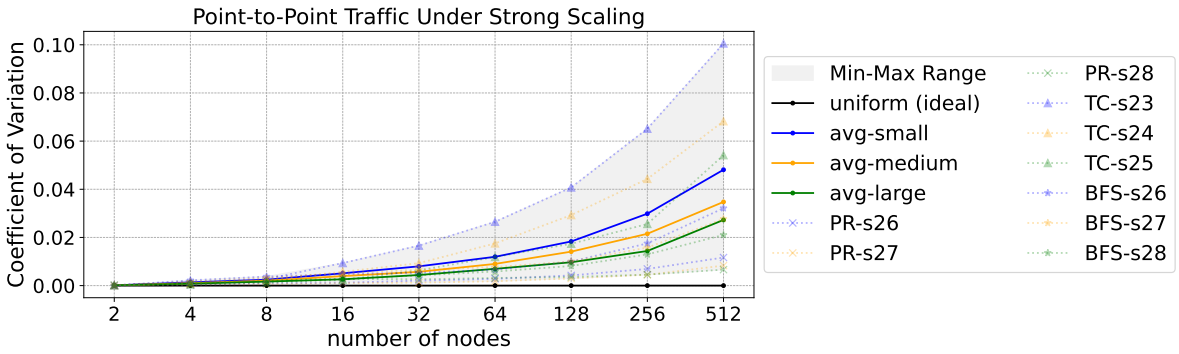

Figure 4.12: Coefficient of Variation of Point-to-Point Communication Traffic on 2-512 nodes.

From both Figure 4.12 and Figure 4.11, it is evident that point-to-point traffic becomes increasingly uniform as the input size grows.

4.3.2 Bisection Analysis

In addition to point-to-point traffic, bisection traffic is a crucial metric for evaluating and selecting the optimal network topology. Bisection traffic measures the total volume of communication that crosses the midpoint of a system when it is divided into two equal halves.

Figure 4.13 shows that the bisection bandwidth scales nearly linearly with increasing machine size under both strong and weak scaling scenarios. At 64 nodes on weak scaling, Triangle Counting (TC), Breadth-First Search (BFS), and PageRank (PR) achieve average

bisection bandwidths of 76.6 TB/s, 160.8 TB/s, and 236.9 TB/s, respectively. When scaled to 512 nodes, the corresponding average bandwidths increase to 435.5 TB/s for TC, 467.7 TB/s for BFS, and 929.0 TB/s for PR.

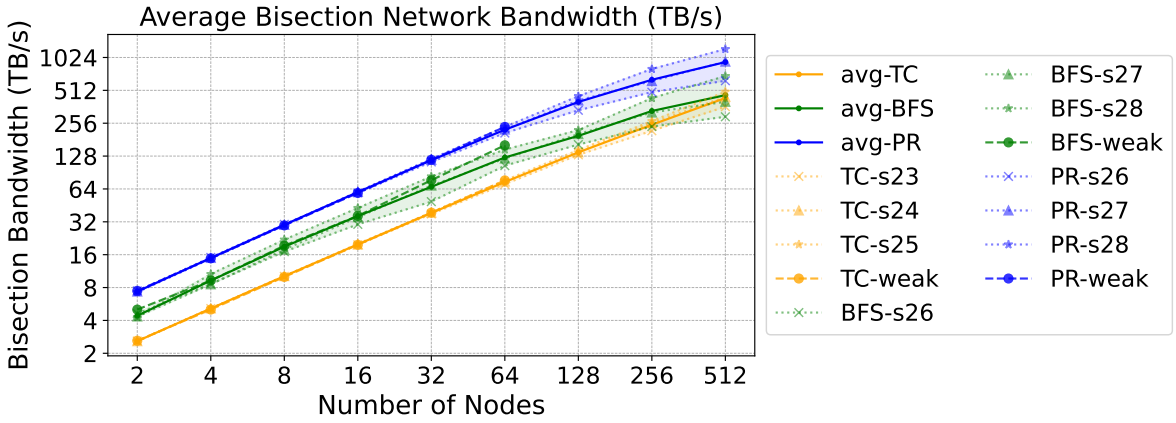

Figure 4.13: Application Use of Bisection Bandwidth.

4.4 Network Packet Size

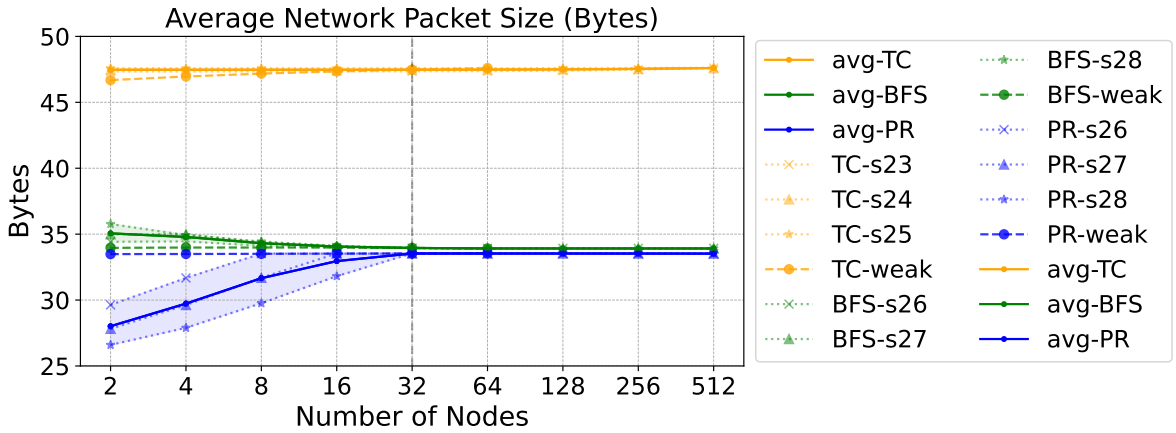

Figure 4.14: Average network packet size on 2-512 nodes under strong and weak Scaling.

Figure 4.14 presents the average packet size for three applications across different machine sizes. The average packet size for Triangle Counting (TC) remains consistently close to 48 bytes across all configurations. In contrast, the average packet sizes for PageRank (PR) and

Breadth-First Search (BFS) fluctuate between 2 and 16 nodes, primarily due to cache misses caused by limited cache capacity. Beyond 32 nodes, the cache becomes sufficiently large to mitigate conflict misses, resulting in stabilized average packet sizes. To further illustrate these trends, Figures 4.15 and 4.16 show the distribution of network packet sizes and packet type ratios at 4 and 32 nodes, respectively.

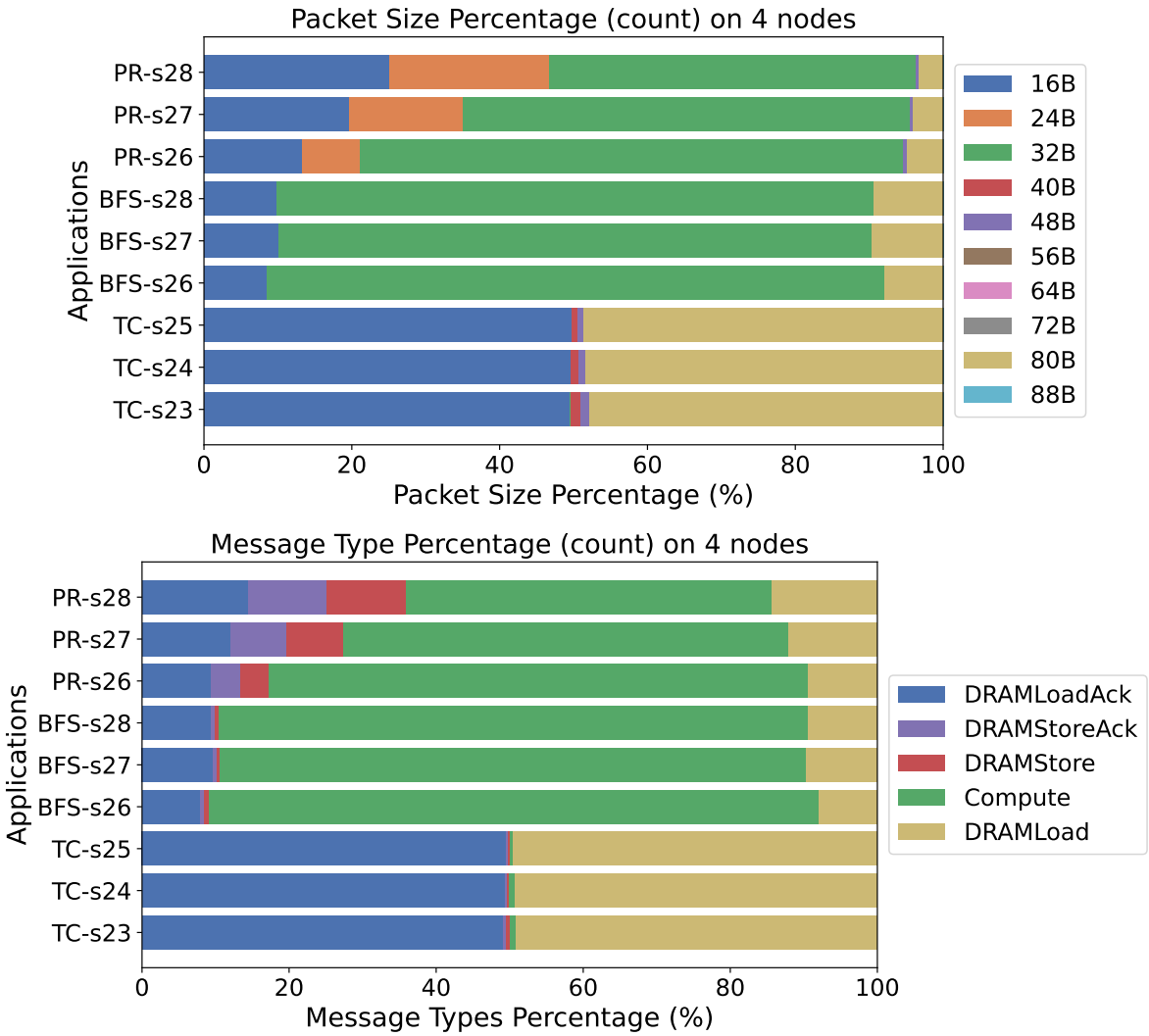

Figure 4.15: Distribution of network packet sizes (upper) and packet type ratios (lower) at 4 nodes.

BFS and PR both utilize the scratchpad as a shared software cache across all nodes to cache vertex data. The key difference is that BFS caches the data read from memory,

while PR caches the updated data that needs to be written back to memory. When a cache miss occurs, BFS triggers a DRAM read operation (corresponding to DRAMLoad and DRAMLoadAck packets), resulting in an increased DRAM read ratio. This trend is observable when comparing BFS-s26 and BFS-s28 on 4 nodes, as shown in Figure 4.15. In contrast, PR triggers a DRAM write operation to store the updated data, generating a 24-byte DRAMStore packet and a 16-byte DRAMStoreAck packet, both smaller than 33 bytes. Consequently, under limited cache capacity, the average packet size in PR becomes smaller, especially for larger graphs or smaller machine sizes since they can trigger the cache miss easily.

Figure 4.14 illustrates the network packet size and type distribution across 32 nodes. Beyond 32 nodes, both the packet size and type distributions become stable. As shown in Figure 4.14, the packet sizes across all three applications are primarily dominated by three categories: 16B packets (corresponding to DRAM acknowledgment packets), 32B packets (the smallest lane-to-lane compute communication packets), and 80B packets (the largest DRAM packets).

4.5 Network Injection Bandwidth Per Node

Network injection bandwidth measures the rate at which compute nodes inject data into the network, typically expressed in GB/s or TB/s. It is a key indicator of network load capacity and communication efficiency.

Section 4.5.1 presents the trends of average network injection bandwidth for the three graph applications under both strong and weak scaling regimes. Additionally, Section 4.5.2 analyzes the instantaneous injection bandwidth behavior within each application, providing insights into temporal variations.

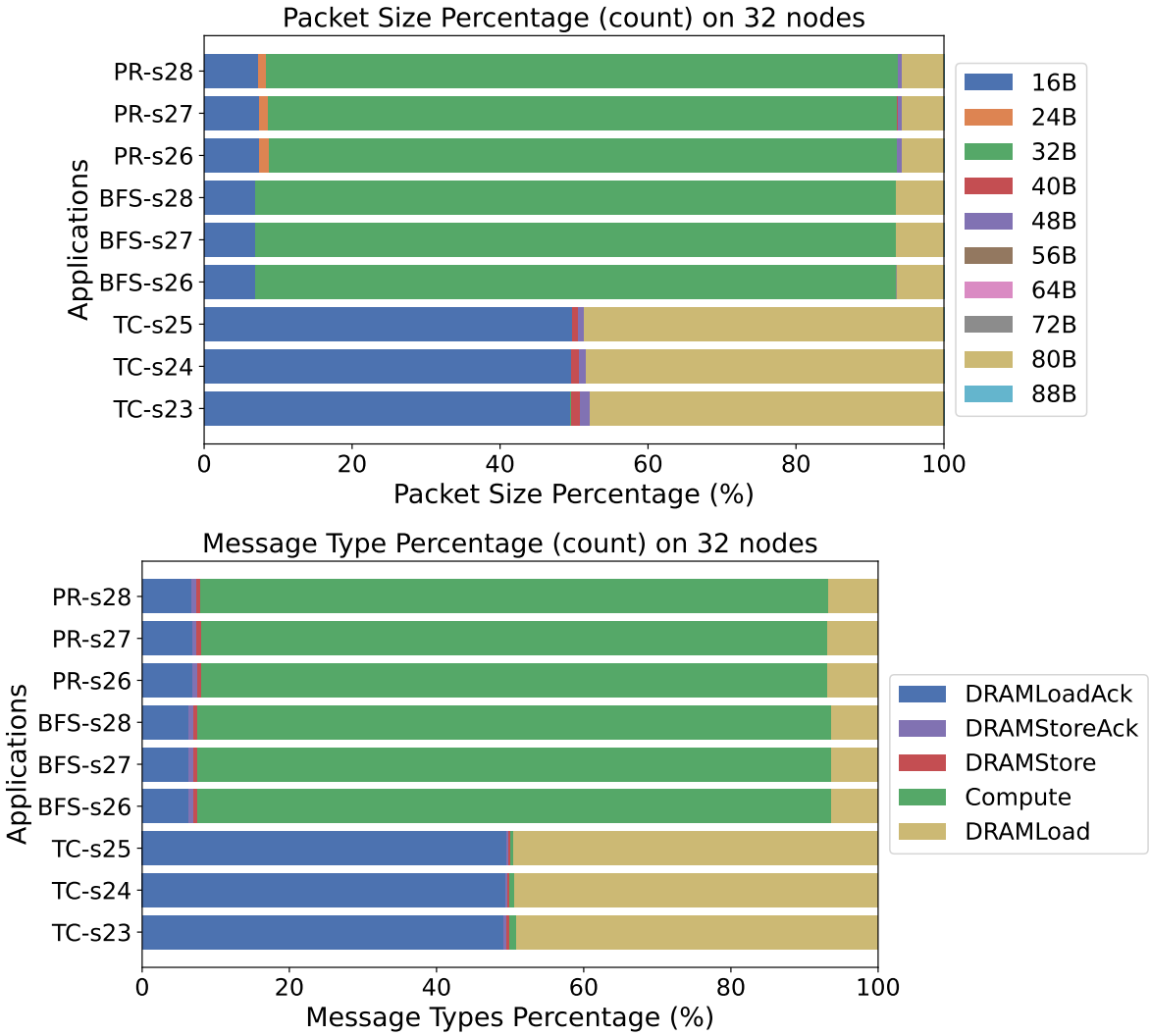

Figure 4.16: Distribution of network packet sizes (upper) and packet type ratios (lower) at 32 nodes.

4.5.1 Average Injection Bandwidth

Figure 4.17 shows the comparison between the average simulated (real) network bandwidth and the ideal network bandwidth under strong scaling. The ideal network bandwidth represents the maximum achievable bandwidth if the application executes at peak computational efficiency without any latency or synchronization overhead (i.e., 100% utilization). The gap between the real and ideal bandwidth reflects inefficiencies and overheads that could potentially be optimized in the program.

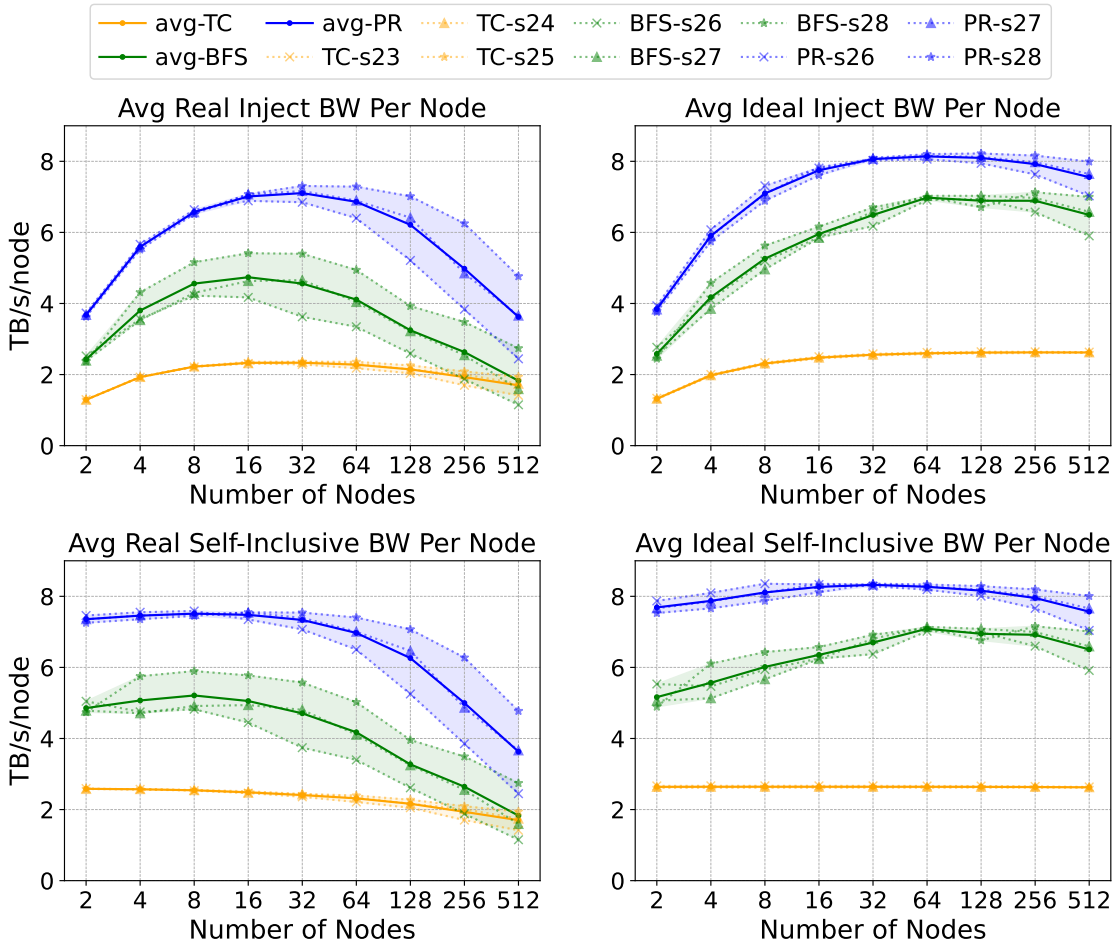

Figure 4.17: Real (left) and Ideal (right) Average Network Injection Bandwidth Per Node under Strong Scaling.

The average real and ideal bandwidths are computed as follows:

$$\text{Avg Real BW} = \frac{\text{Total Traffic}}{\text{Execution Time} * n} \tag{4.7}$$

$$\text{Avg Ideal BW} = \frac{\text{Total Traffic}}{\text{Avg Inst Time} * n} = \frac{\text{Total Traffic}}{\text{Total Inst Time}} \tag{4.8}$$

As discussed in Section 4.3, UpDown applications generate nearly uniform point-to-point communication. Therefore, the network bandwidth measured in simulation reflects only the cross-node traffic and does not capture the total communication activity, which also includes

local (intra-node) traffic. In a system with n nodes, approximately $\frac{1}{n}$ of the traffic is local (sent to the same node), while $\frac{n-1}{n}$ is distributed to remote nodes. To more accurately analyze the network injection bandwidth trend across different machine sizes, we define the *self-inclusive network bandwidth*, which accounts for both local and remote traffic:

$$\text{Self-Inclusive BW} = \frac{n}{n-1} \times \text{Network BW} \tag{4.9}$$

The self-inclusive network bandwidth provides a more accurate estimate of the total communication volume generated by UpDown applications, especially for small-scale systems. It also offers a clearer view of bandwidth trends and scaling behavior across increasing node counts.

For a more direct comparison of real and ideal bandwidths, Figure 4.18 (strong scaling) presents the average bandwidth across the three input graphs for each machine size and application. Table 4.5 summarizes the average network injection bandwidth for both strong scaling and weak scaling experiments.

For strong scaling (Figure 4.17), the real network bandwidth peaks at 16 nodes, after which the average traffic injection rate per node begins to decline. The increase from 2 to 16 nodes is attributed to a reduction in the proportion of traffic sent to the local node as machine size grows. The decline beyond 16 nodes is primarily due to reduced compute utilization and degraded scaling efficiency, as illustrated in Figure 4.3 for larger machine configurations. Under ideal conditions, all three applications—across different input graph sizes—maintain similar ideal bandwidths at larger scales (32–512 nodes), with only minor reductions compared to their real bandwidths.

In contrast, under weak scaling, the self-inclusive bandwidth remains relatively stable as the number of nodes increases. As a result, both real and ideal network bandwidths converge toward an upper bound given by $(n-1)/n \times$ SelfInclusive BW, indicating consistent and scalable network behavior across machine sizes.

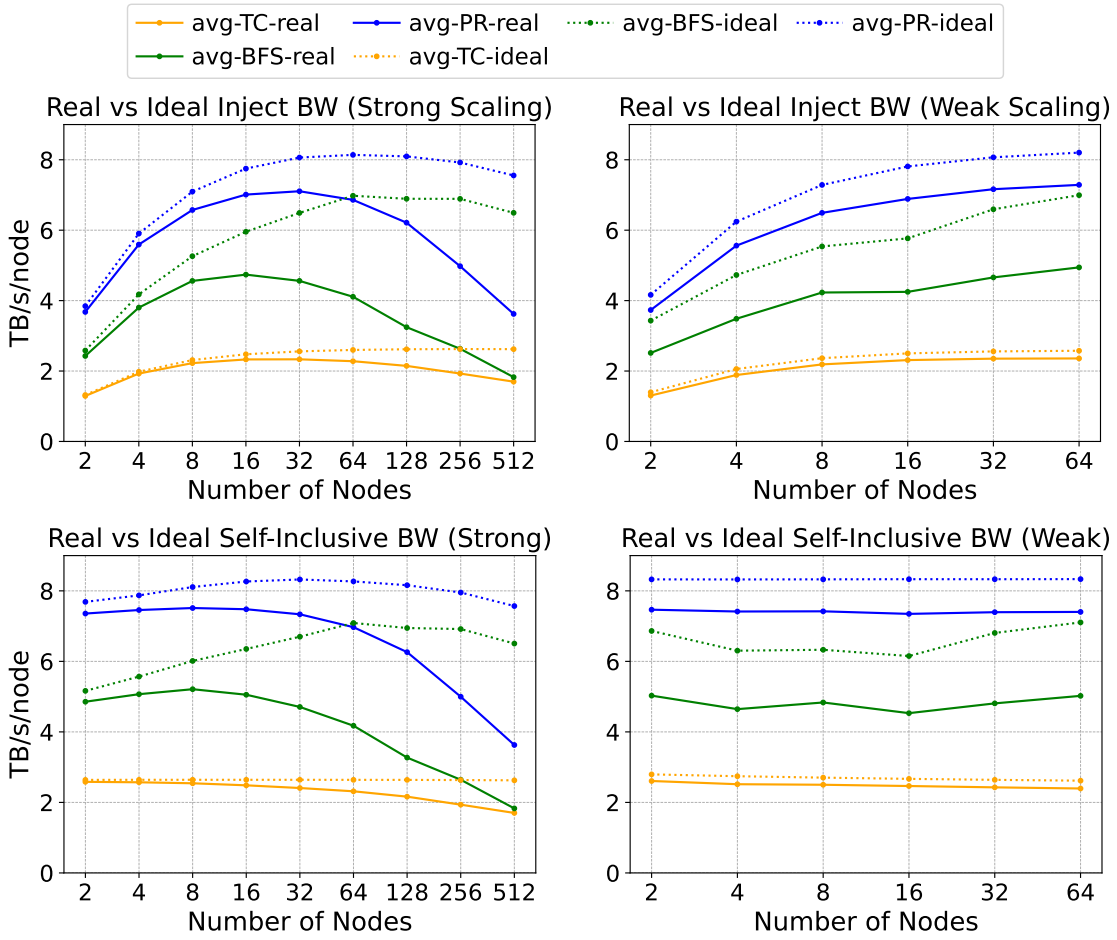

Figure 4.18: Network Injection Bandwidth Per Node on Strong Scaling (left) and Weak Scaling (right).

Table 4.5: Average Network Injection Bandwidth Per Node

Applications	**TC-Real**	**TC-Ideal**	**BFS-real**	**BFS-Ideal**	**PR-real**	**PR-Ideal**
Strong Scaling (16 nodes)	2.3 TB/s	2.5 TB/s	4.2-5.4 TB/s	5.9 TB/s	7.1 TB/s	7.8 TB/s
Weak Scaling (64 nodes)	2.4 TB/s	2.6 TB/s	4.9 TB/s	7.0 TB/s	7.3 TB/s	8.2 TB/s

4.5.2 Instantaneous Injection Bandwidth

In the simulation, time is divided into slots of 100 cycles (equivalent to 50 ns), and the network traffic within each slot is used to compute the instantaneous injection bandwidth. Figure 4.19 and present histograms of the instantaneous network injection bandwidth per node for 16-node and 64-node configurations.

Comparing the two cases (16 nodes and 64 nodes), the overall trends are similar. Both Triangle Counting (TC) and PageRank (PR) exhibit relatively stable injection behavior throughout most of the execution, as indicated by the sharp and concentrated peaks in Figure 4.19. Furthermore, the small gap between the average and P90 bandwidth values confirms the consistency of their network usage. In contrast, Breadth-First Search (BFS) shows greater variability in instantaneous bandwidth. The histogram exhibits a broader spread across statistical measures (average, P90, P95, P99), indicating less stable and more bursty network injection behavior during execution.

Figure 4.20 shows the instantaneous injection bandwidth over time for 16-node and 64-node configurations, further illustrating the observations from the bandwidth histograms. In these figures, we increase the size of the time slots used to compute instantaneous bandwidth. Instead of a fixed 100-cycle window, each data point now represents the average network bandwidth over 1/1000 of the total execution time. As a result, the sharp spikes observed in the histograms (Figure 4.19) are smoothed out. Note that the execution time on 16 nodes is 3.3–3.6× longer than on 64 nodes, due to reduced compute parallelism and resource availability.

As shown in Figure 4.20, both Triangle Counting (TC) and PageRank (PR) maintain relatively stable injection bandwidth throughout most of execution. TC demonstrates an almost ideal timeline, with only a single sharp bandwidth peak at the beginning. This peak results from a broadcast from node 0 to all other nodes; hence, the spike is larger in the 64-node case compared to the 16-node case. When the maximum injection bandwidth is limited, this initial burst can be amortized over the following execution time, minimizing its effect on overall performance. In contrast, PR exhibits an additional sharp bandwidth spike near the end of execution, corresponding to a cache write-back event. Unlike the initial broadcast, this late-stage spike is difficult to amortize due to limited remaining execution time, which may cause an increase in total execution time under bandwidth-constrained

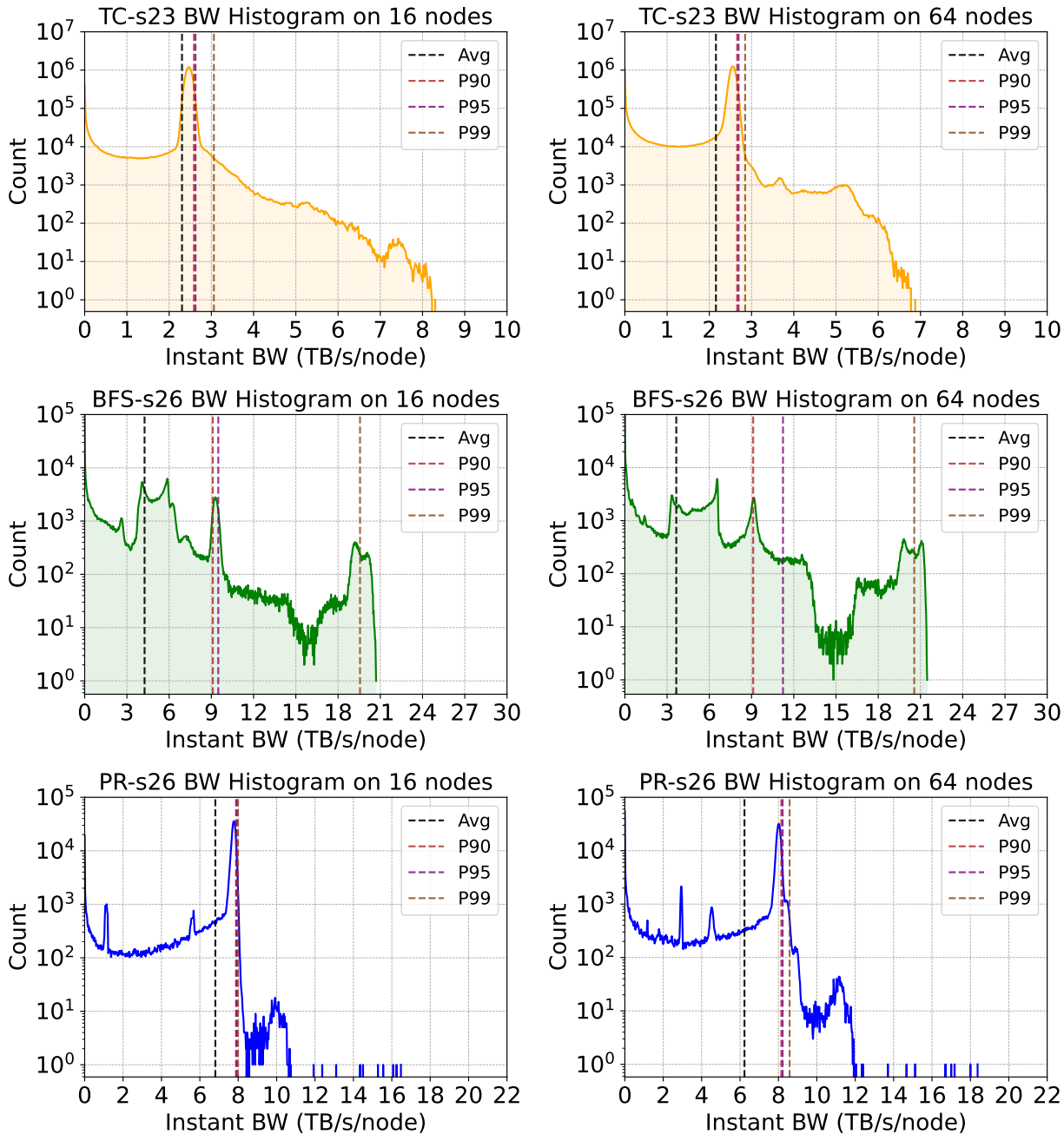

Figure 4.19: Instantaneous Bandwidth Histogram on 16 and 64 nodes.

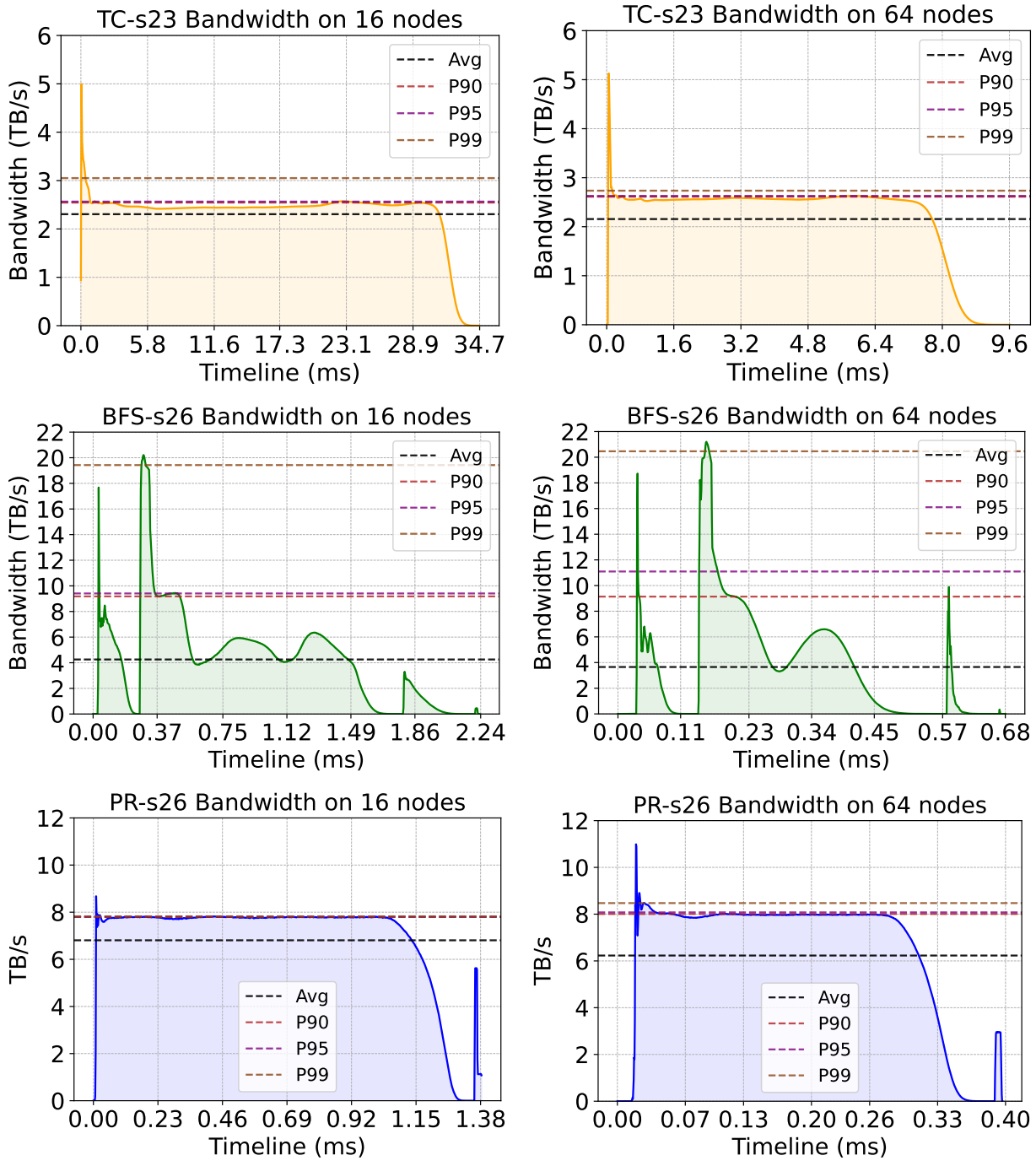

Figure 4.20: Instantaneous Bandwidth Timeline on 16 and 64 nodes.

conditions. However, the total network traffic associated with this tail event is relatively small, so the impact on performance is limited.

BFS, by comparison, displays a highly fluctuating bandwidth profile over time. This variability arises because TC and PR follow a single `do-all` iteration model (i.e., `for v in Graph`), whereas BFS consists of multiple `do-all` iterations (i.e., `for v in frontier`), each corresponding to a distinct level of the traversal frontier. As a result, the communication pattern of BFS is inherently more irregular, leading to greater bandwidth variation throughout execution.

4.6 Network Packet Injection Rate Per Node

Network packet injection rate per node refers to the rate at which packets are sent or injected into the network from each individual machine. Both the packet injection rate and packet sizes influence the routing decision frequency, as they determine how often the network must make routing decisions to efficiently deliver data across the system.

Before diving into further detail, it is important to note a key observation: the distribution and trend of the packet injection rate closely mirror those of the injection bandwidth.

4.6.1 Average Packet Injection Rate

Figure 4.21 shows the average simulated real and ideal packet injection rates per node. As discussed in Section 4.5.1, the ideal injection rate represents the maximum achievable packet rate assuming peak computational efficiency, without any latency or synchronization overhead (i.e., 100% utilization). The self-inclusive packet injection rate accounts for packets sent both to the local node and to remote nodes. This metric provides a more accurate understanding of packet behavior as the machine size increases, especially in small-size systems where local communication constitutes a non-negligible portion of overall traffic.

For a more direct comparison of real and ideal packet injection rates, Figure 4.22 (strong

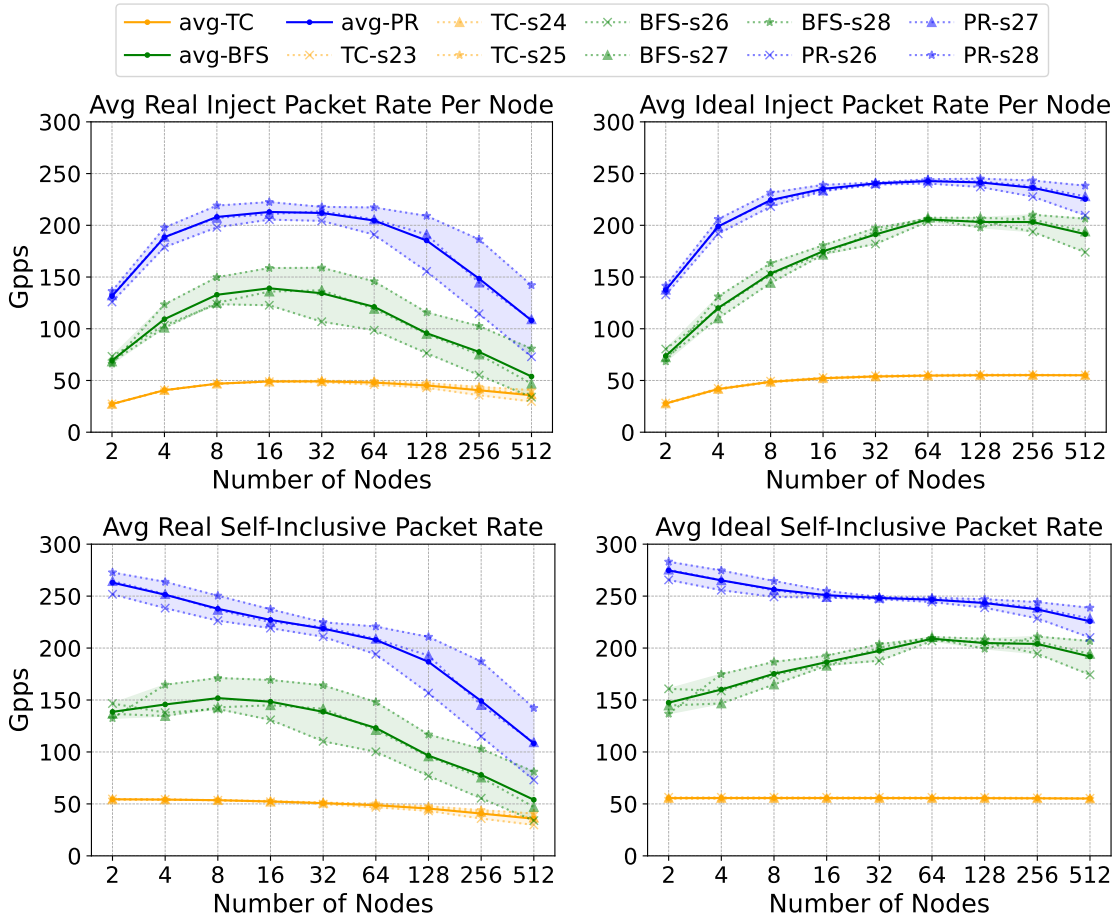

Figure 4.21: Real (left) and Ideal (right) Average Network Packet Injection Rate Per Node under Strong Scaling.

scaling) presents the average packet rate across the three input graphs for each machine size and application. The observed trends are similar to those seen in network injection bandwidth, as discussed in Section 4.5.1.

Under strong scaling, as the number of nodes increases, the gap between the self-inclusive real and ideal packet rates widens. This is due to the decreasing workload per node, which reduces the system's ability to hide communication and synchronization overheads.

In contrast, for weak scaling, both the self-inclusive real and ideal packet injection rates remain nearly constant across different machine sizes. Consequently, both real and ideal rates converge toward an upper limit as the number of nodes increases.

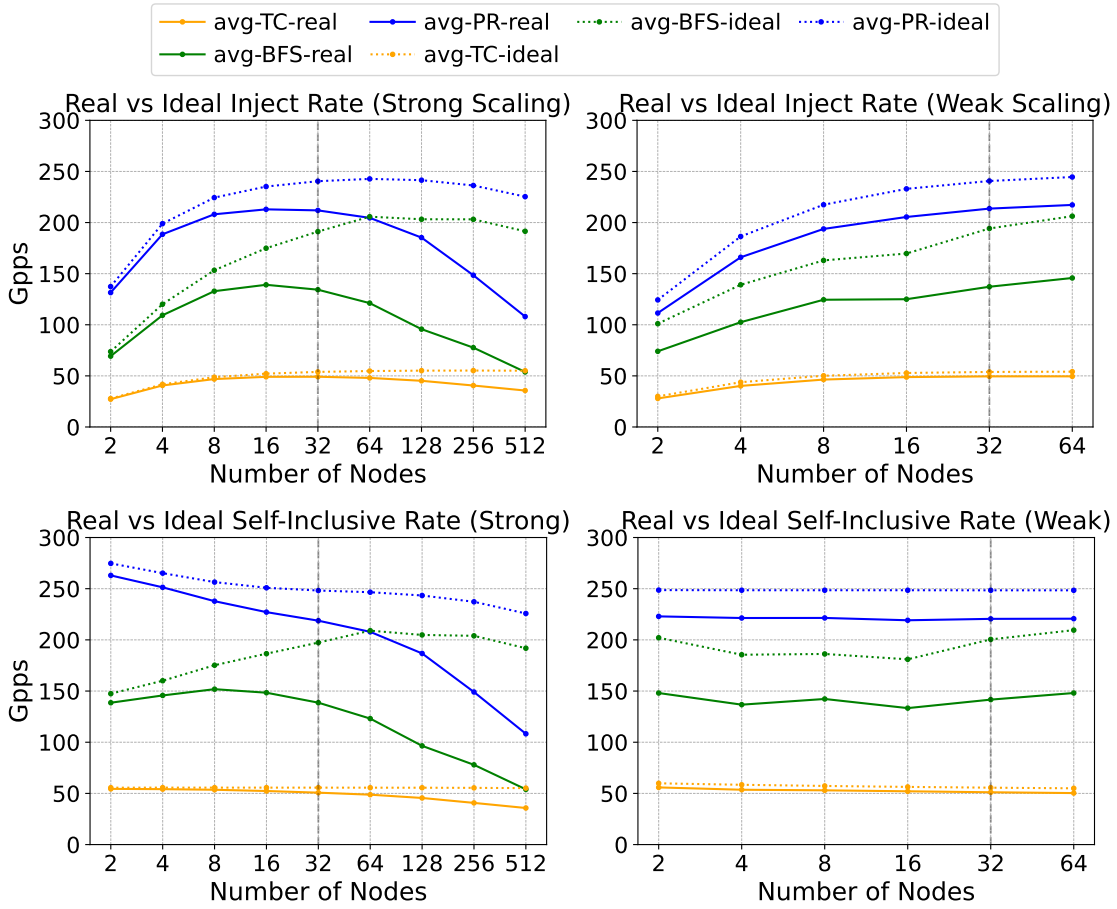

Figure 4.22: Network Packet Injection Rate Per Node on Strong Scaling (left) and Weak Scaling (right).

Table 4.6 summarizes the highest observed packet injection rates under both strong scaling (at 16 nodes) and weak scaling (at 64 nodes). Compared to the frame forwarding rates discussed in Section 2.3, which typically support only hundreds of Mpps, UpDown requires 100–1000× higher packet injection and forwarding rates. However, when examined at the granularity of individual cores, as shown in Table 4.7, each UpDown core sends only average 1–6 network packets per 100 instructions. This observation suggests that while system-level network pressure is high, the per-core communication intensity remains modest, enabling efficient overlap between computation and communication.

Table 4.6: Average Network Packet Injection Rate Per Node

Applications	**TC-Real**	**TC-Ideal**	**BFS-real**	**BFS-Ideal**	**PR-real**	**PR-Ideal**
Strong Scaling (16 nodes)	49.5 Gpps	52.8 Gpps	123-159 Gpps	180 Gpps	205-222 Gpps	239 Gpps
Weak Scaling (64 nodes)	49.5 Gpps	54.1 Gpps	146 Gpps	206 Gpps	217 Gpps	245 Gpps

Table 4.7: Average Network Packet Injection Rate Per Core (2GHz)

Applications	**TC-Real**	**TC-Ideal**	**BFS-real**	**BFS-Ideal**	**PR-real**	**PR-Ideal**
Strong Scaling (16 nodes)	24.75 Mpps	26.4 Mpps	62-80 Mpps	90 Mpps	103-111 Mpps	120 Mpps
Weak Scaling (64 nodes)	24.75 Mpps	27.05 Mpps	73 Mpps	103 Mpps	108.5 Mpps	123 Mpps

4.6.2 Instantaneous Packet Injection Rate

In the simulation, we divide time into slots of 100 cycles (equivalent to 50 ns) and compute the average packet injection rate within each slot to represent the instantaneous packet injection rate. As discussed in Section 4.5.2, the trend of the instantaneous packet injection rate closely mirrors that of the instantaneous network injection bandwidth.

Following the analysis in Section 4.5.2, we select two representative strong-scaling points —16 nodes and 64 nodes—to further examine the detailed distribution of instantaneous packet injection rates.

Figure 4.23 shows the histogram of instantaneous network packet injection rates per node for 16-node and 64-node configurations. Comparing the two cases, the overall trends are similar. Both Triangle Counting (TC) and PageRank (PR) exhibit stable injection behavior for most of the execution time, as indicated by the prominent peaks in Figure 4.23. The small gap between the average and the 90th percentile (P90) further confirms the consistency of their instantaneous injection rates. In contrast, the injection rates observed in Breadth-First Search (BFS) are less stable. Although two small peaks are visible in the histogram, there is a wider spread of instantaneous packet rates, indicating higher variability during execution.

Figure 4.24 further illustrates the earlier observations by showing the average instantaneous packet injection rate across different nodes over time. In Figure 4.24, we increase the size of the time slots used to compute the instantaneous packet rate. Instead of using a fixed

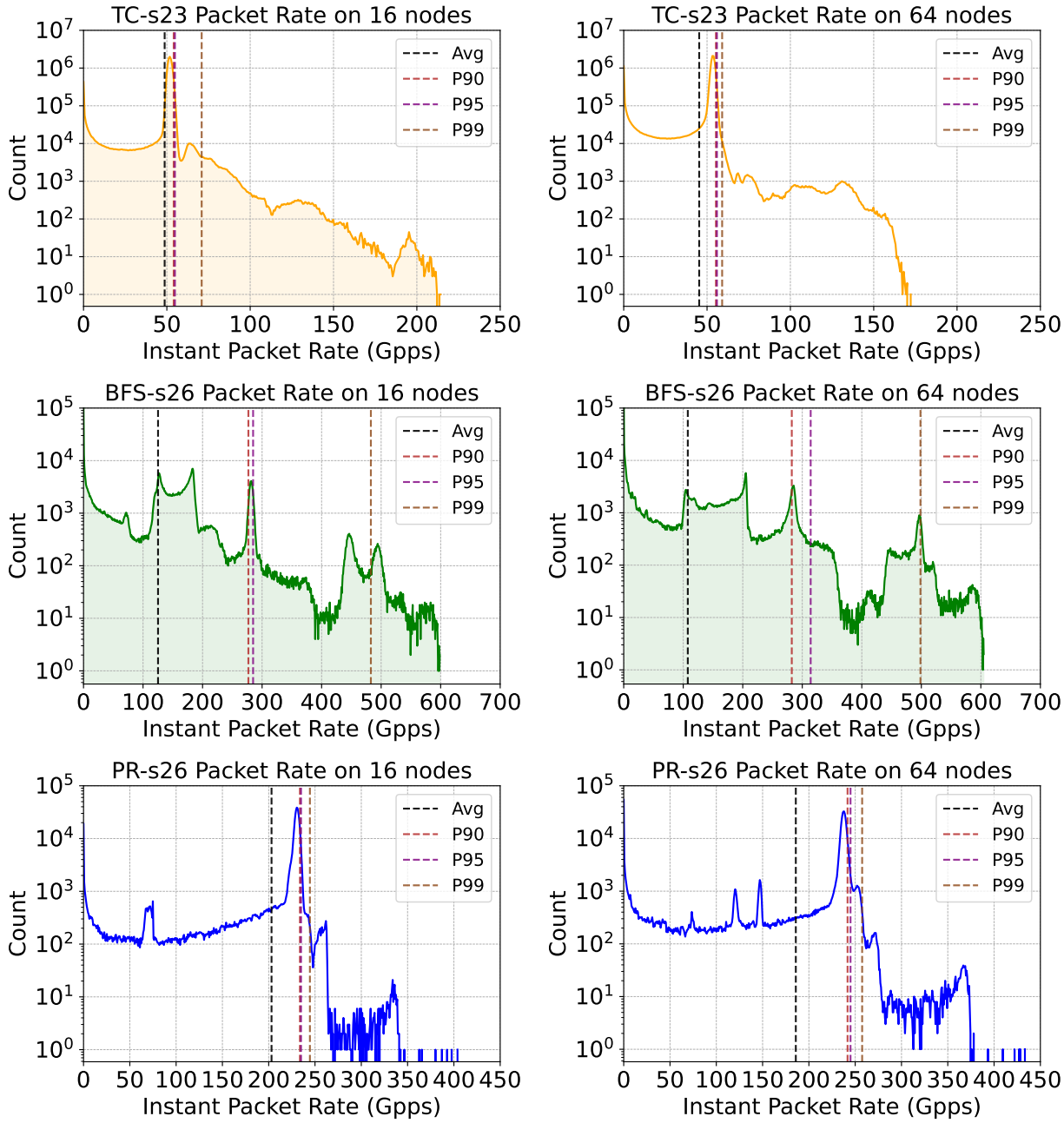

Figure 4.23: Instantaneous Packet Rate Per Node Histogram on 16 and 64 nodes.

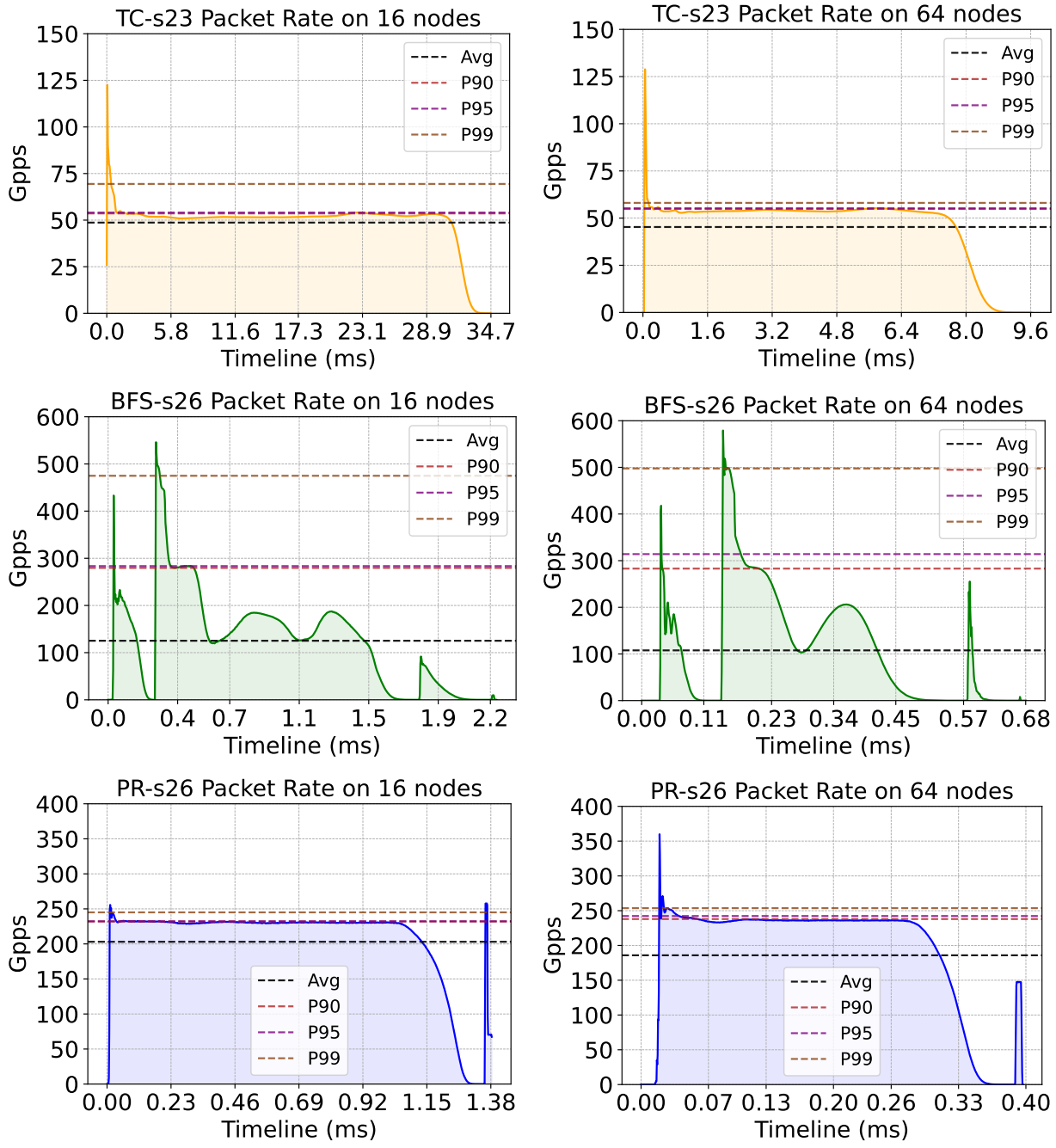

Figure 4.24: Instantaneous Packet Rate Per Node Timeline on 16 and 64 nodes.

100-cycle window, each data point now represents the average rate over 1/1000 of the total execution time. As a result, some of the sharp instantaneous rate spikes observed in the histogram (Figure 4.23) are smoothed out.

Both Triangle Counting (TC) and PageRank (PR) maintain relatively stable injection rates throughout most of the execution (Figure 4.24). In contrast, Breadth-First Search (BFS) exhibits a highly fluctuating instantaneous packet rate profile. This difference arises because TC and PR follow a single do-all iteration model, while BFS consists of multiple do-all iterations, each corresponding to different levels of the traversal frontier. These iterations lead to noticeable fluctuations in packet injection activity over time.

4.7 Other Characteristics

Similar to HPE Slingshot and NVIDIA NVLink, the UpDown network constructs packets directly on top of the physical layer to minimize protocol overhead and enhance data transmission efficiency. These packets typically incorporate mechanisms such as checksums for error detection and flow control. However, in this study, we focus solely on the network injection behavior and do not model network topology, protocol, or switch-level details. Consequently, we assume a reliable network where fault tolerance, packet loss recovery, and congestion control are handled internally by the network system.

Furthermore, based on the UpDown architectural design, network packets are independent and can be delivered out of order without impacting program correctness.

4.8 Conclusion

Three irregular and scalable graph applications are selected to evaluate the UpDown system network. These applications generate uniform point-to-point network traffic, with a normalized traffic bisection ratio close to 0.5, indicating a strong requirement for a high-bisection

bandwidth network. Packet sizes vary between 16 bytes and 88 bytes, with 16 B, 32 B, and 80 B packets being the most frequently used. These applications generate between 0.5 and 2 bytes of network traffic per executed instruction, and achieve injection traffic rates per node ranging from 2 TB/s to 8 TB/s per node. The average packet injection rate ranges from 50 Gpps to 250 Gpps, which is higher than the capabilities of current network switches that typically support 500–800 Mpps per port.

CHAPTER 5
UNDERSTANDING UPDOWN ARCHITECTURE AND APPLICATION PERFORMANCE SENSITIVITY TO NETWORK PERFORMANCE

Chapter 4 examines the network characteristics generated by applications, including network communication connectivity, injection rate, and packet size under both strong and weak scaling scenarios.

This chapter focuses on analyzing the impact of network latency and network injection bandwidth limitations per node on the performance of UpDown applications. Since weak scaling provides more stable performance and maintains high utilization, we conduct weak scaling experiments across 1 to 16 nodes in this section.

5.1 Methodology

Fastsim2 (Section 3.5) is a cycle-driven UpDown simulator. To model network latency and throughput limitations, a queue is implemented on each node to temporarily store outgoing network messages. In each cycle, only messages that have been queued for at least the specified network latency (in cycles) are eligible to be sent out. Furthermore, the total outgoing packet traffic is constrained by the network injection bandwidth limitation. This mechanism ensures that latency and bandwidth limitations are enforced at the packet send-out stage.

5.2 Network Latency Sensitivity

UpDown incorporates two key hardware features to tolerate network latency: lightweight threading and low-latency, non-blocking event-driven scheduling, as illustrated in Figure 5.1.

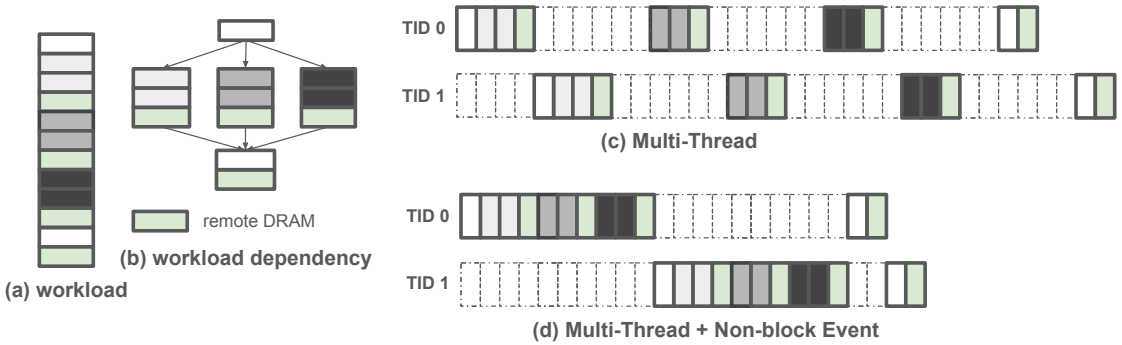

Figure 5.1: Network Latency Influence on UpDown. (Remote DRAM latency = 6)

With a small register state, UpDown threads can be created and destroyed in a single cycle, enabling high thread parallelism. In addition to multi-threading, UpDown supports non-blocking, event-driven execution. Events can be triggered by either software or hardware sources (e.g., DRAM accesses), allowing computation to proceed while waiting on data. As shown in Figure 5.1, UpDown can expose fine-grained parallelism even within a single thread and to hide network latency more effectively.

As described in Section 3.4, Triangle Counting (TC), Breadth-First Search (BFS), and PageRank (PR) are implemented using the Map-Shuffle-Reduce framework. Table 5.1 summarizes the latency tolerance features employed by each application to mitigate network latency.

To minimize instruction overhead during set intersection, TC processes neighbor lists sequentially. As a result, there is no internal parallelism within a single thread, and latency hiding relies solely on multi-thread parallelism. In contrast, both PR and BFS exploit intra-thread parallelism during the `map` phase, as each thread accesses multiple neighbors in parallel. This structure allows event-driven scheduling to overlap memory operations and computation within a thread. For the `reduce` phase, PR performs scalar accumulation, which does not benefit from intra-thread parallelism. However, in BFS, the `reduce` step involves inserting vertices into the next frontier. This operation can be executed using event-driven mechanisms in a non-blocking manner, allowing single-threaded processing to still leverage

intra-thread parallelism for latency hiding.

Table 5.1: Latency Tolerance Features Usage on TC, PR and BFS.

Parallelism Feature	**TC-map**	**TC-reduce**	**PR-map**	**PR-reduce**	**BFS-map**	**BFS-reduce**
Multi-thread	Y	Y	Y	Y	Y	N
Intra-thread Parallelism	N	N	Y	N	Y	Y

In the following sections, we first analyze the degree of multi-threaded parallelism in each application and evaluate how effectively it can hide network latency. We then present the actual network latency that each UpDown application can tolerate without significant performance degradation.

Table 5.2 presents the level of multi-thread parallelism under a simulated 1000-cycle (500 ns) network latency on 16 nodes with 64, 128, and 255 threads. Other weak scaling configurations exhibit similar behavior.

In the optimal scenario, network packets are generated evenly throughout execution. On average, each thread executes approximately 73, 17, and 23 instructions—depending on the application—before issuing a network communication and switching to another thread. The "Max Thread" value indicates the average maximum number of active threads per UpDown lane during execution. The maximum compute parallelism achievable through multi-threading is calculated as InstPerPacket $\times$ MaxThread.

Table 5.2: Multi-thread Parallelism on TC, PR and BFS.

Applications	**TC-128t**	**TC-255t**	**PR-64t**	**PR-128t**	**BFS-128t**	**BFS-255t**
Inst Per Packet	74.1	72.7	16.5	16.5	22.7	24.6
Max threads	123	242	67	110	124	239
Tolerant cycles	9,114	17,593	1,106	1,815	2,815	5,879

Figure 5.2 shows the slowdown of each application under varying network latency conditions and baseline is default 1000 cycles. The maximum network latency each application can tolerate with less than 20% performance degradation is summarized in Table 5.3.

To further analyze the sensitivity to network latency, Figure 5.3 presents the relative

Table 5.3: Maximum Network Latency Cycles Applications < 20% Performance Degradation.

Applications	**TC-128t**	**TC-255t**	**PR-64t**	**PR-128t**	**BFS-128t**	**BFS-255t**
Multi-thread	9,114	17,593	1,106	1,815	2,815	5,879
Experiment Result	8,000	16,000	4,000	8,000	8,000	8,000

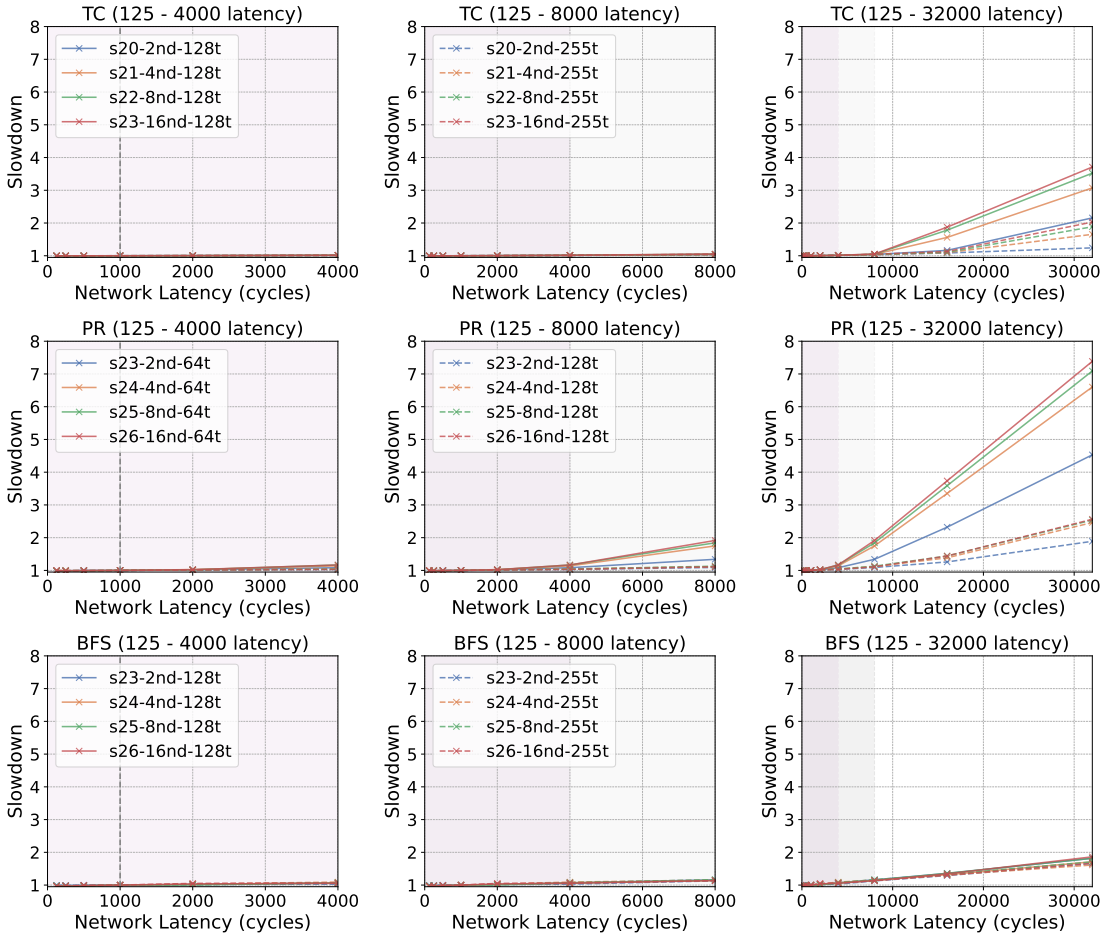

Figure 5.2: Slowdown across different network latency (baseline 1000-cycle network latency), first row is TC, second row is PR, third row is BFS under network latency between 125-32,000 cycles.

execution time between $2k$-cycle latency and k-cycle latency, illustrating the increasing execution time (diminishing performance) as network latency increases.

In the best case, UpDown has sufficient workload to fully hide network latency, resulting in minimal execution time increase—yielding a speedup close to 1. In the worst case, execution is dominated by network latency, and doubling the latency results in a nearly 2× increase in execution time, producing a speedup close to 2. Values between 1 and 2 indicate

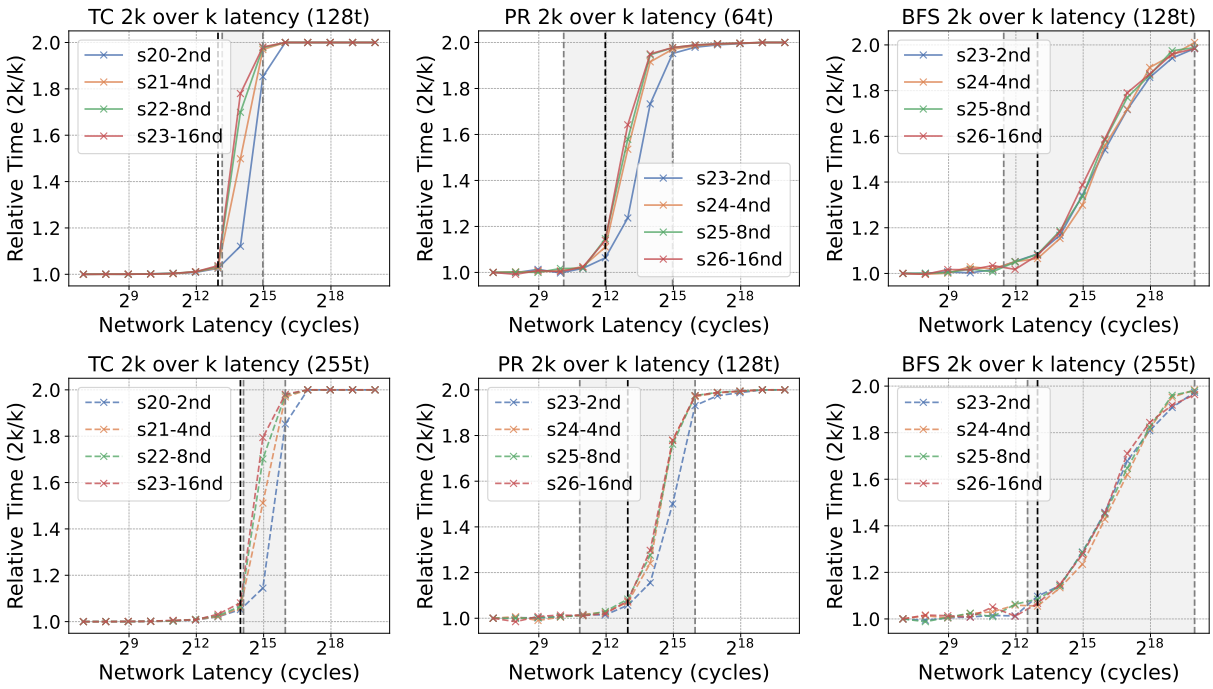

Figure 5.3: Relative Execution Time on 2k latency over k latency. First column is TC, second column is PR, third column is BFS. The beginning of the gray region corresponds to the effective multi-thread parallelism, while the end marks the highest latency value at which some latency hiding is still effective (i.e., speedup approaches 2). The shaded gray region represents the range of internal-parallelism latency tolerance. The dotted black line indicates the maximum tolerant network latency in Table 5.3.

partial latency hiding, where available parallelism is not sufficient to completely mask the increased latency.

In Figure 5.3, the beginning of the shaded gray region corresponds to the effective multi-thread parallelism—i.e., the minimum network latency that can be fully hidden by available threads. The end of the gray region marks the highest latency value at which latency hiding remains partially effective, as indicated by a speedup approaching 2×. Thus, the gray region represents the range of network latency that can be tolerated using internal parallelism (both inter- and intra-thread). The dotted black line indicates the maximum network latency that each application can tolerate without incurring more than 20% performance degradation, as reported in Table 5.3. The detailed analysis for each application is provided below:

- **Triangle Counting (TC)**: TC primarily relies on multi-threaded compute parallelism

to hide network latency. As shown in Table 5.2, TC can tolerate network latency up to approximately 9,000 cycles with 128 threads and up to 18,000 cycles with 255 threads. Beyond these thresholds, execution time increases significantly due to insufficient remaining parallelism to hide further latency.

- **PageRank (PR)**: PR can hide network latency up to 4,000 cycles with 64 threads and up to 8,000 cycles with 128 threads. In addition to multi-threaded parallelism, PR leverages internal parallelism within the `map` phase. This internal parallelism provides roughly 4× the latency-hiding capability compared to multi-threading alone.

- **Breadth-First Search (BFS)**: BFS exhibits a different behavior. The network latency tolerance trends for 128 and 255 threads are similar, as shown in Figure 5.3. This is because the `reduce` phase fully utilizes internal compute parallelism within a single thread, making the effective compute parallelism at 128 and 255 threads nearly equivalent. Additionally, the rate of execution time increase under growing latency is significantly slower for BFS compared to TC and PR, further demonstrating the benefit of intra-thread parallelism in network hiding latency.

5.3 Network Injection Bandwidth Sensitivity

In this section, we limit the network injection bandwidth per node from 0.5 TB/s to 9 TB/s across 1–16 nodes under weak scaling conditions. To maximize compute parallelism, we select the configurations with the highest effective thread-level parallelism: TC with 255 threads, BFS with 255 threads, and PR with 128 threads. We constrain the amount of network traffic each node can send every 100 cycles in Fastsim2 based on the configured injection bandwidth and rerun the simulation. The corresponding performance results are shown in Figure 5.4.

For TC, performance degradation begins when the injection bandwidth per node falls

below 3 TB/s. For BFS, the minimum injection bandwidth required to maintain performance is approximately 6 TB/s. For PR, the minimum required injection bandwidth to sustain performance is around 8 TB/s.

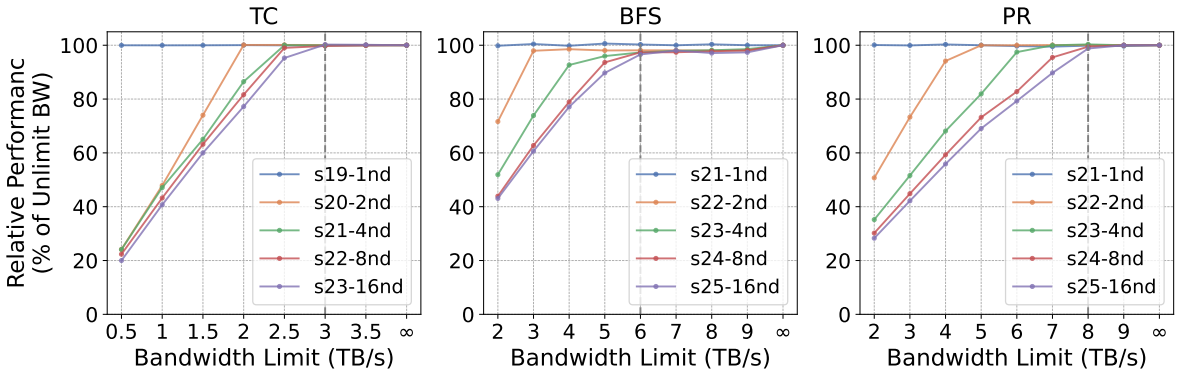

Figure 5.4: Relative Performance versus network injection bandwidth limitation.

5.3.1 Influence of Sharp Peak Instantaneous Bandwidth

In this section, we study the performance impact of sharp peak instantaneous bandwidth.

As shown in Section 4.5.2, UpDown applications occasionally generate sharp peaks in instantaneous network injection bandwidth in Fastsim2, reaching up to 20 TB/s per node. However, in real-world systems, network bandwidth is limited. To evaluate the effect of such peaks, we limit the maximum injection bandwidth per node to values close to the P95 or P99 of the observed instantaneous bandwidth and rerun the experiment. Figure 5.5 shows the instantaneous bandwidth over time across 16 nodes under these limitations.

The first column presents the instantaneous network injection bandwidth over time without any bandwidth limitations. The second column shows the bandwidth over time under P95 or P99 limitations. The third column provides the estimated bandwidth over time based on the unlimited case with estimated model applied (details in Section 5.3.2).

As discussed in Section 4.5.2, limiting the peak injection bandwidth does not significantly affect overall performance. The excess network traffic from peak events is spread

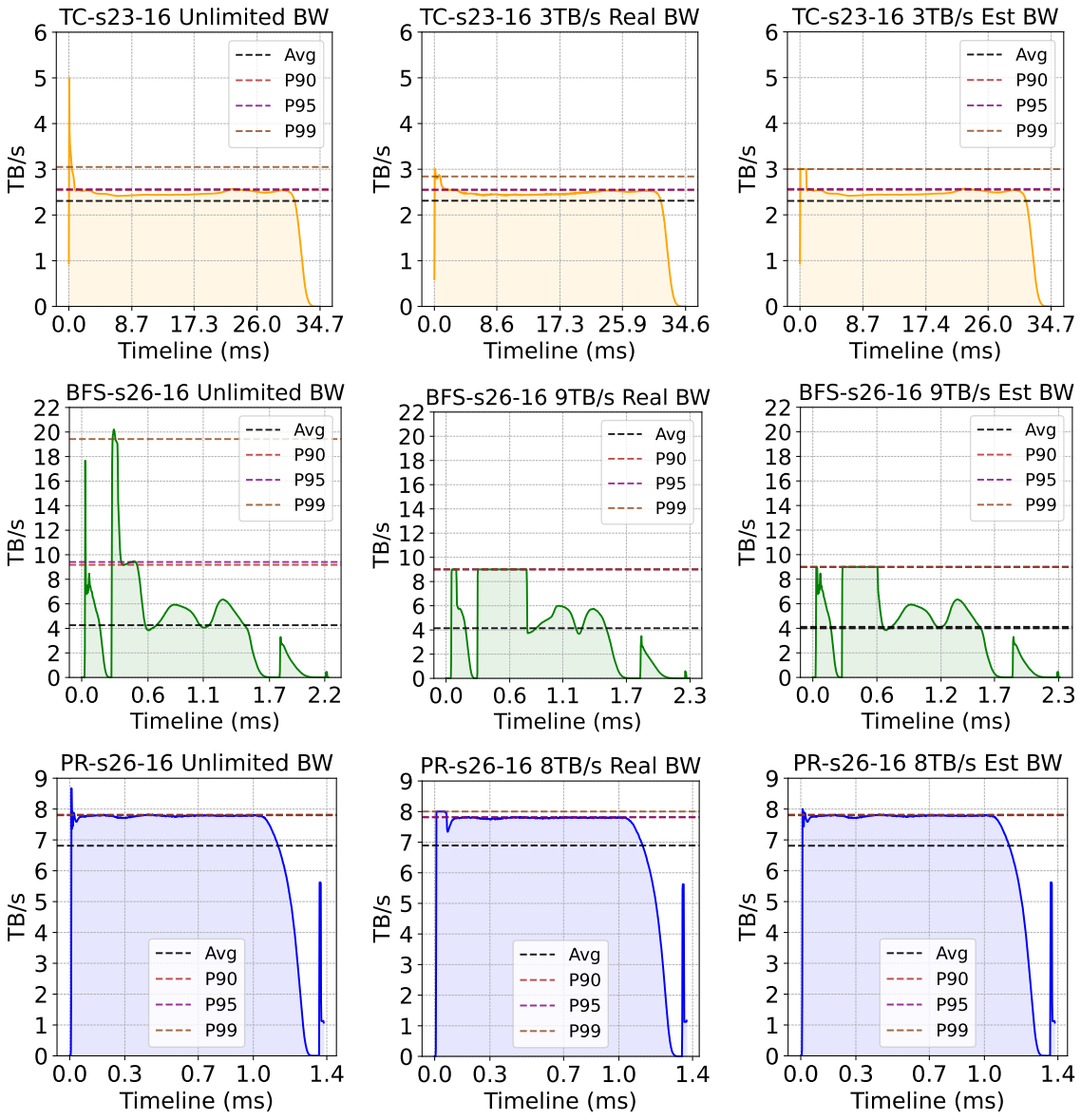

Figure 5.5: Instantaneous bandwidth over time for each application: unlimited injection bandwidth (first column), P95 or P99 injection bandwidth limitation (second column), and estimated bandwidth based on unlimited execution (third column).

over subsequent time periods. This amortization does not impact the ongoing computation speed and the ongoing network injection bandwidth, demonstrating that UpDown's architecture provides sufficient compute parallelism to hide delays caused by peak traffic injection limitations.

5.3.2 Injection Bandwidth Estimation over Time

Algorithm 1 Generate New Bandwidth Timeline with Upper Limit

Input: `bw_timeline`, `upper_bw`

Output: `new_timeline`

```
1  Initialize new_timeline as empty list
   idx ← -1
   tmp_traffic ← 0
   while idx < length of bw_timeline −1 do
2      if tmp_traffic < upper_bw then
3          idx ← idx +1
           tmp_traffic ← tmp_traffic + bw_timeline[idx]
           if tmp_traffic > upper_bw then
4              Append upper_bw to new_timeline
               tmp_traffic ← tmp_traffic − upper_bw
5          else
6              Append tmp_traffic to new_timeline
               tmp_traffic ← 0
7      else
8          Append upper_bw to new_timeline
           tmp_traffic ← tmp_traffic − upper_bw
```

In Figures 5.5, we observe that instantaneous bandwidth exceeding the limitation is amortized over time, while bandwidth below the limitation remains mostly unchanged. Based on this observation, we build a model to estimate the instantaneous bandwidth over time (`new_timeline`) based on the original bandwidth timeline without bandwidth limits (`bw_timeline`). This model estimates the execution time of the UpDown system under the assumption that the injection bandwidth is the limiting factor, given sufficient network traffic. If the estimated execution time closely matches the actual simulation time, it indicates that the application and system configuration are able to fully utilize the available network bandwidth. In this case, the execution is bottlenecked by the injection bandwidth limit rather than computation or other system components. This model will be used for performance projection in Section 6.2.2.

The input bandwidth timeline is divided into multiple time slots. During the simulation, the model first handles the current estimated network traffic (`tmp_traffic`). If current traffic (`tmp_traffic`) is larger than bandwidth limitation(`upper_bw`), the excess bandwidth is amortized across subsequent slots (Algorithm 1, line8). Otherwise, the model accumulates traffic from the next entry in `bw_timeline` to current traffic (`tmp_traffic`) (line4). If the updated `tmp_traffic` then exceeds `upper_bw`, the traffic is capped at `upper_bw` (line 5); otherwise, the full value of `tmp_traffic` is injected into the network (line 7). Thus, the output timeline caps the peak bandwidth while preserving the low-bandwidth regions.

The estimated instantaneous bandwidth model predicts the achievable bandwidth timeline and application execution time under various injection bandwidth limitations, assuming an ideal scenario in which sufficient compute parallelism exists to hide the resulting communication delays. This model is used to project execution time under different network injection bandwidth constraints, as discussed in Section 6.2.2. The third column in Figure 5.5 presents the estimated instantaneous network bandwidth over time. The estimates closely match the simulated results produced by Fastsim2, validating the model's accuracy.

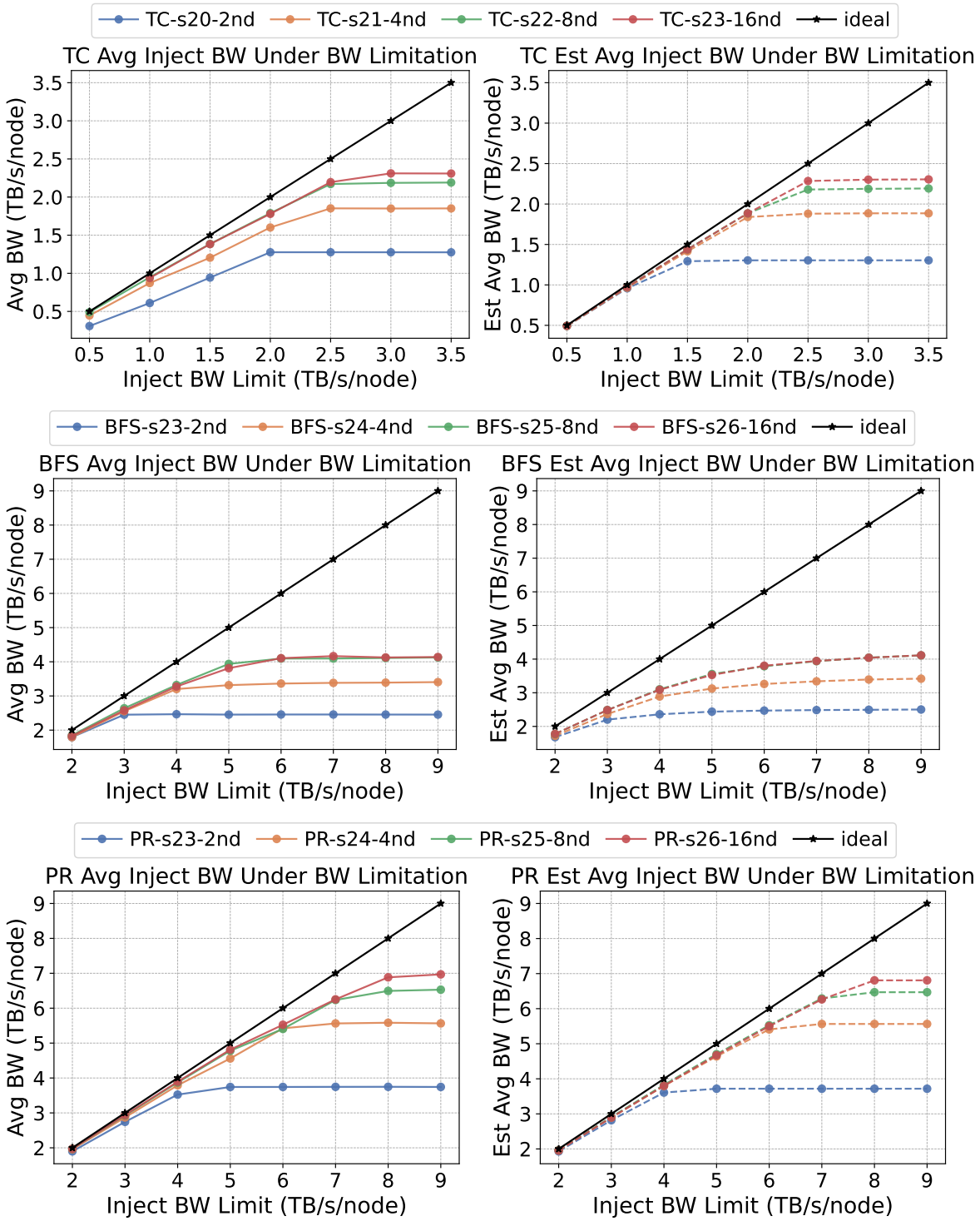

Figure 5.6: The average network injection bandwidth per node under different injection bandwidth limitation (left: real; right: estimated).

Figure 5.6 shows the average network injection bandwidth per node across different bandwidth limitations. The left column shows real performance results measured on Fastsim2, while the right column presents the estimated results. This demonstrates that UpDown can tolerate low injection bandwidth limitation and still deliver optimal performance.

5.4 Conclusion

UpDown can tolerate significantly longer network latencies by leveraging both multi-threaded and intra-thread parallelism. All three UpDown applications are able to sustain performance with network latencies ranging from 8,000 to 16,000 cycles (4–8 µs) without significant degradation.

In addition to latency tolerance, UpDown also effectively handles fluctuations in injection bandwidth and can fully utilize the available network bandwidth under low bandwidth constraints. To better analyze temporal variations in bandwidth usage, we developed an instantaneous bandwidth model that simulates injection bandwidth over time and helps characterize bandwidth sensitivity across applications.

CHAPTER 6

PROJECTION TO FULL-SCALE UPDOWN SYSTEM

Due to memory limitations, UpDown simulator supports simulating TC, PR, and BFS on up to 512 UpDown nodes. However, based on the system design, a full-scale UpDown system consists of 16,384 nodes, which is 32 times larger than the largest simulated machine size. This chapter projects the network injection rate and application performance on full-scale UpDown systems under different network configurations.

6.1 Projection Methodology

6.1.1 Input Graph Size

As discussed in Chapter 4, strong scaling reveals growing inefficiencies as the number of nodes increases—specifically, reduced per-node workload leads to greater sensitivity to latency, synchronization overhead, and more pronounced imbalance. In contrast, our goal in projection is to estimate the best-case network performance for each application. To this end, we adopt a weak scaling strategy for BFS and PageRank, which maintains a constant workload per node and avoids the degradation observed in strong scaling. The projection is based on the weak scaling data collected from 1 to 64 nodes, as presented in Section 4.2.2. The input graph sizes used for projection are detailed in Table 6.1.

Table 6.1: Projected input graph

Nodes	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**	**8192**	**16384**
TC	s19	s20	s21	s22	s23	s24	s25	s26	s27	s28	s29	s30	s31	s32	s33
PR	s22	s23	s24	s25	s26	s27	s28	s29	s30	s31	s32	s33	s34	s35	s36
BFS	s22	s23	s24	s25	s26	s27	s28	s29	s30	s31	s32	s33	s34	s35	s36

6.1.2 Network Traffic

This section projects the total network traffic for 2 to 16,384 nodes using the input graphs listed in Table 6.1. To validate the projection methodology, we compare the projected values against simulated results on 2 to 64 nodes in Section 6.2.

As shown in Section 4.3, UpDown applications generate nearly uniform point-to-point communication. Assuming a system with n nodes, approximately $\frac{1}{n}$ of the network traffic is local (sent to the same node), while the remaining $\frac{n-1}{n}$ is distributed to other nodes. Therefore, the self-inclusive network traffic can be derived from the measured cross-node network traffic using the relationship defined in Formula 6.2:

$$\text{Network Traffic} = \frac{n-1}{n} \times \text{Self-Inclusive Network Traffic} \tag{6.1}$$

$$\text{Self-Inclusive Network Traffic} = \frac{n}{n-1} \times \text{Network Traffic} \tag{6.2}$$

Figure 6.1 presents the self-inclusive network traffic for Triangle Counting (TC), PageRank (PR), and Breadth-First Search (BFS) from 2 to 64 nodes, based on weak scaling simulations conducted with Fastsim2. For all three applications, the self-inclusive network traffic—plotted on a $\log_2$ scale—increases approximately linearly with the $\log_2$ of the node count over this range. Accordingly, we model the projected network traffic using a linear function in log-log space:

$$log_2(\text{Self-Inclusive Network Traffic}) = k * log_2(n) + b_0 \tag{6.3}$$

$$\text{Self-Inclusive Network Traffic} = 2^{k*log_2(n)+b_0} = n^k * 2^{b_0} = n^k * b \tag{6.4}$$

$$\text{Network Traffic} = \frac{n-1}{n} * \text{Self-Inclusive Network Traffic} \tag{6.5}$$

$$= n^{k-1} * (n-1) * b \tag{6.6}$$

The coefficients k and b are computed using simulation results at 32 and 64 nodes.

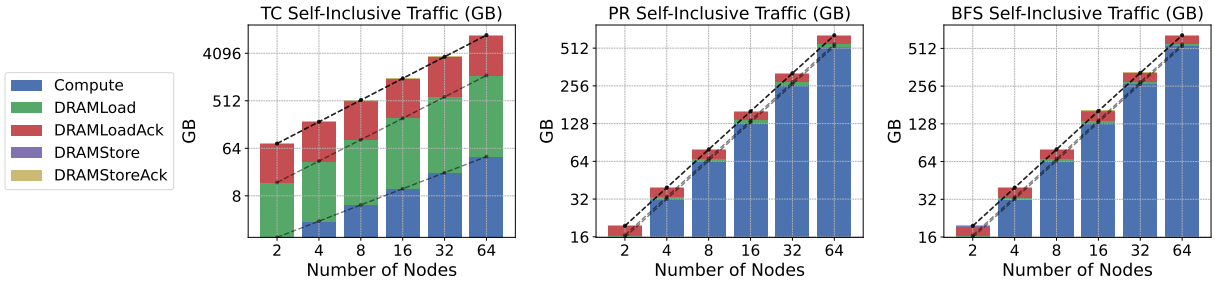

Figure 6.1: Total Network Traffic (GB).

6.1.3 Execution Time

This section projects the execution time of applications on 2 to 16,384 nodes using the input datasets defined earlier (Table 6.1).

As discussed in Chapter 5, execution time is influenced by both the application's computational behavior and the underlying network configuration. In the case of UpDown applications, sensitivity analysis reveals that execution time is largely unaffected by network latency unless it exceeds 8000 cycles (4us). Therefore, in this projection, we focus solely on the impact of network injection bandwidth as the limiting network factor.

Two execution time estimates are computed: one based on program computation time, and the other constrained by network injection bandwidth. The projected execution time is defined as the maximum of these two values, reflecting the effective bottleneck in each case.

$$\text{Execution Time} = max(\text{Program Execution Time}, \text{Network Injection Time}) \tag{6.7}$$

Program Execution Time

Figure 6.2 presents the program execution time for Triangle Counting (TC), Breadth-First Search (BFS), and PageRank (PR) across machine sizes ranging from 2 to 64 nodes. In

Triangle Counting (TC), the execution time grows approximately linearly with the $\log_2$ of the number of nodes. The projected execution time for TC, shown as a dotted line, is extrapolated from simulation results on 32 and 64 nodes. In contrast, BFS and PR have workloads that scale linearly with the number of vertices and edges, adhering more closely to ideal weak scaling. As a result, their program execution times remain nearly constant as the system scales. The dotted lines in Figure 6.2 show the projected program execution times, computed based on 64 nodes, with an observed projection error of less than 10%.

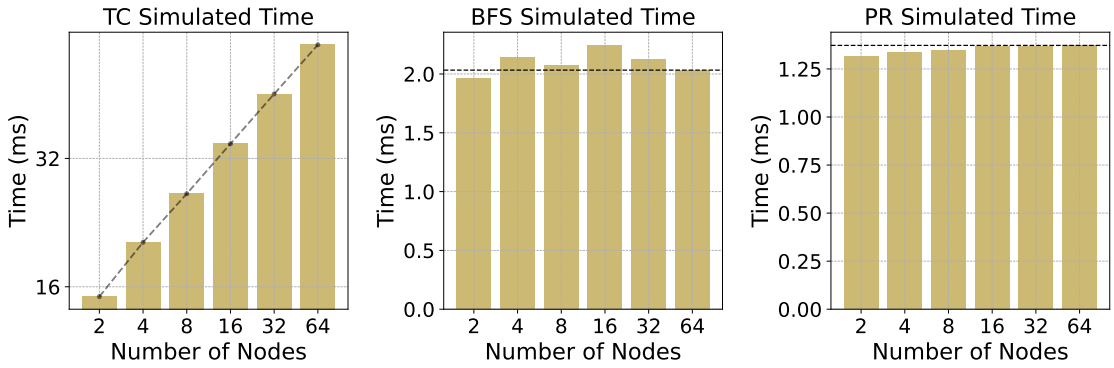

Figure 6.2: Program Execution Time.

$$log_2(\text{Program Execution Time}) = a log_2(n) + c_0 \tag{6.8}$$

$$\text{Program Execution Time} = n^a * c \tag{6.9}$$

Network Injection Time

In Section 5.3.2, we introduced a model to estimate instantaneous network bandwidth under various injection bandwidth limitations, using the unconstrained instantaneous bandwidth timeline as a baseline. Therefore, if we can accurately project the unconstrained injection bandwidth timeline, the model can be used to estimate execution time under different bandwidth limitations.

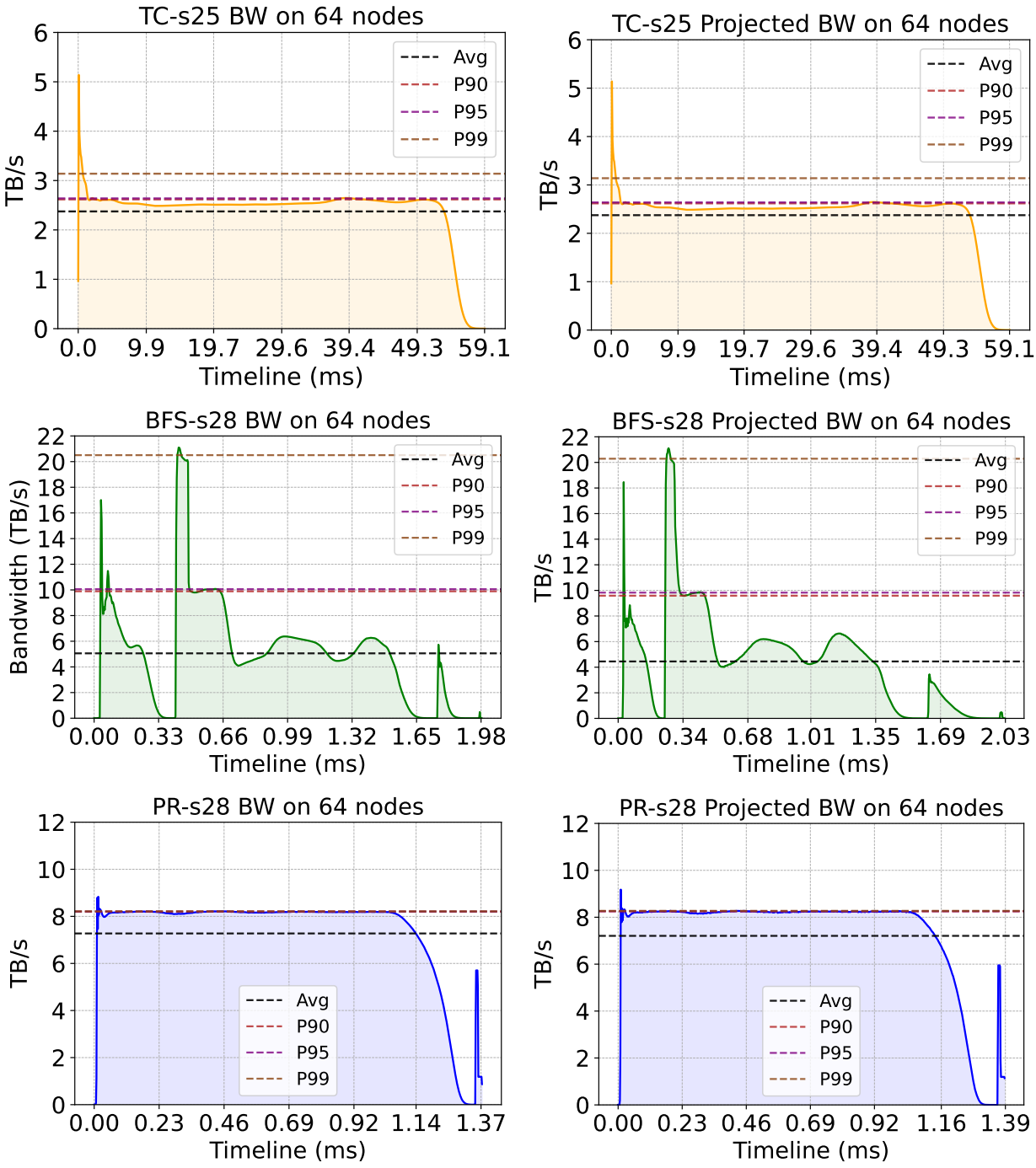

Figure 6.3: Instantaneous Real (left) and Projected (right) Bandwidth Timeline on 64 node.

Figure 4.20 presents the instantaneous injection bandwidth over time for 16-node and 64-node configurations. The overall trends are similar across both scales, with the primary differences appearing along the horizontal axis (execution time) and vertical axis (bandwidth magnitude). To handle the time axis, we adopt the projected program execution time described in Section 6.1.3. For the vertical axis, we scale the bandwidth values using the ratio of the projected average injection bandwidth per node across different system sizes. Figure 6.3 compares the actual instantaneous injection bandwidth on 64 nodes to the projected bandwidth derived from 16-node data, demonstrating the accuracy of our projection methodology. The close match between the two curves confirms that the projection preserves both temporal structure and magnitude trends.

6.2 Result and Analysis

In this section, we present the projected performance results for configurations ranging from 2 to 16,384 nodes, alongside real measurement data from 2 to 64 nodes, to validate the accuracy of our projection methodology. This projection is conducted under a weak scaling regime to ensure that each node maintains sufficient workload as the system scales. We use network data collected from 32-node and 64-node configurations to project performance on other system sizes. The input graphs used for this projection are summarized in Table 6.1.

We begin by analyzing performance without any injection bandwidth limitation, representing the best-case scenario the UpDown system can achieve under a fixed network latency of 550 ns. We then compare the performance of each application under different per-node injection bandwidth constraints to evaluate the impact of network bandwidth limitations.

6.2.1 Without Injection Bandwidth Limitation

Figure 6.4, Figure 6.5 and Figure 6.6 present both the real simulation results for 16–64 nodes and the projected results for 16–16,384 nodes, based on data collected from the 32

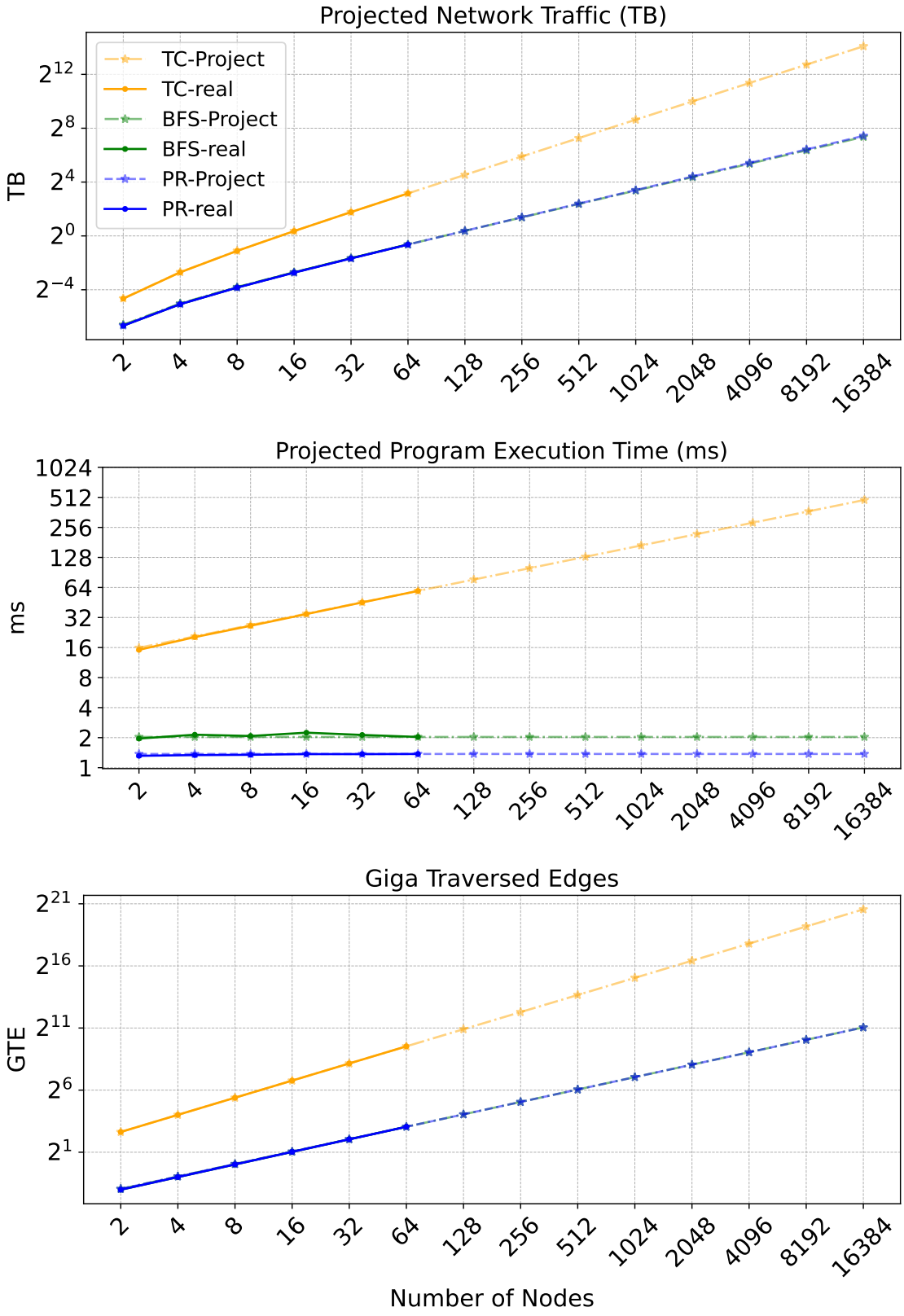

Figure 6.4: Projected and Real Network Traffic, Execution Time, and Traversed Edges on 2-16384 nodes.

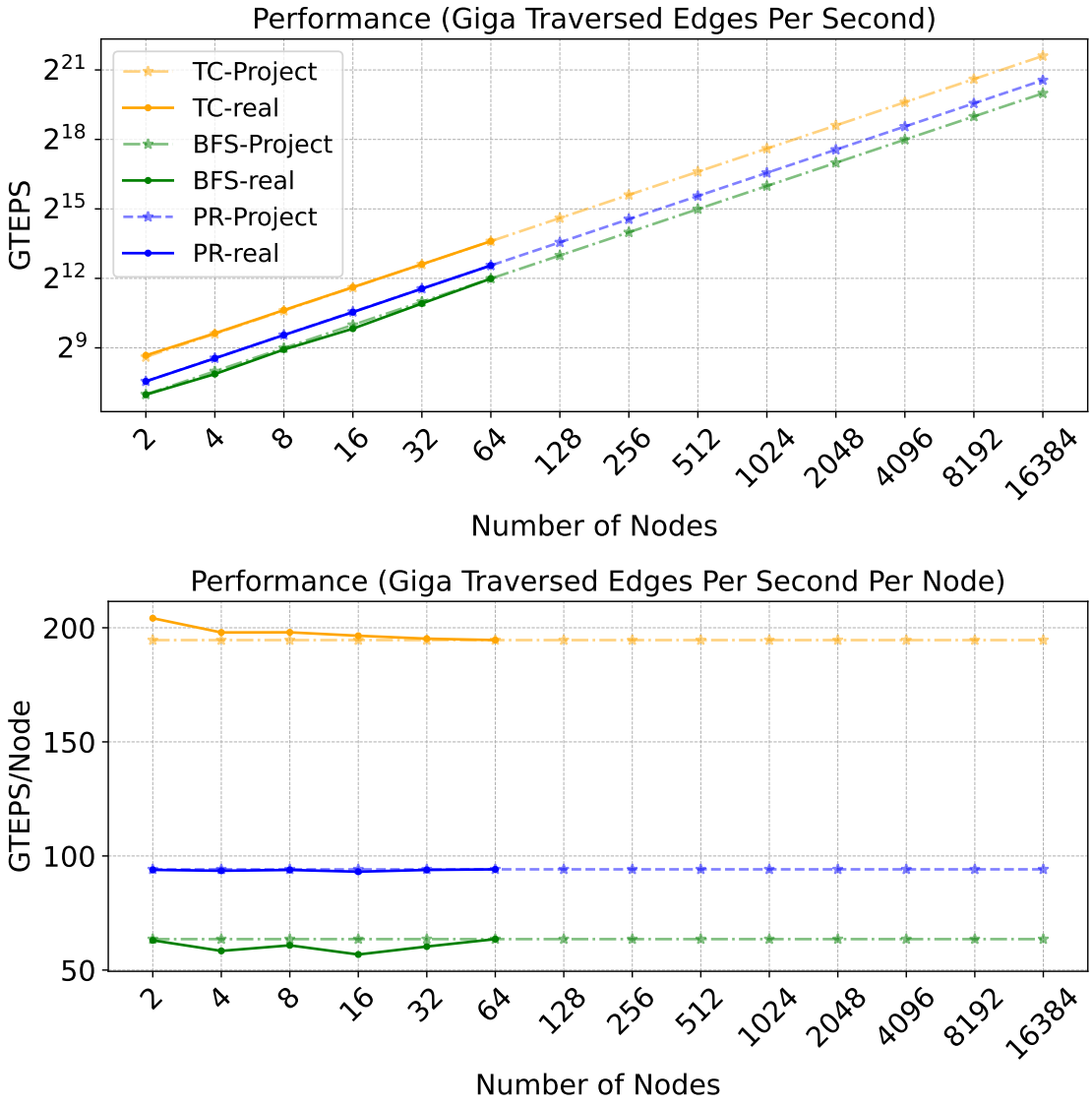

Figure 6.5: Projected and Real Performance on 2-16384 nodes.

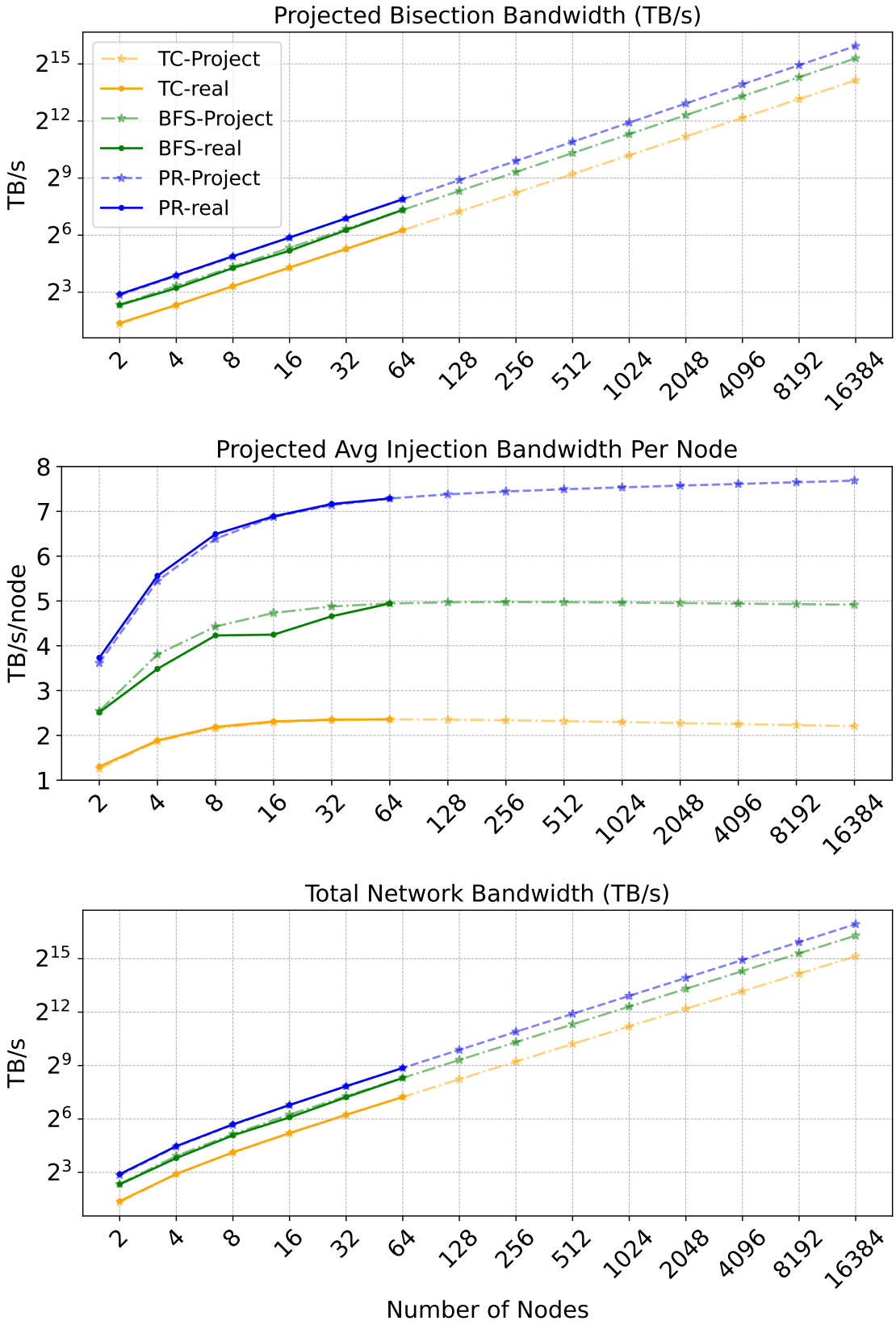

Figure 6.6: Projected and Real Network Injection Bandwidth on 2-16384 nodes.

and 64-node configuration. The projected performance for 16,384 nodes is summarized in Table 6.2.

As shown in the figures, the projected results closely match the real performance data. Both network traffic and execution time scale well with system size, and performance exhibits linear speedup. The average injection bandwidth per node stabilizes at approximately 2.5 TB/s for Triangle Counting (TC), 5 TB/s for Breadth-First Search (BFS), and 8 TB/s for PageRank (PR).

Table 6.2: Projected Data on Full-scale UpDown System

Apps	**Network Traffic**	**Execution Time**	**Total BW**	**BW Per Node**	**Bisection BW**	**GTEPS**
TC	17.4 PB	482.1 ms	26.2 PB/s	2.2 TB/s	18.1 PB/s	3,181,388
BFS	173 TB	2.0 ms	80.5 PB/s	4.9 TB/s	40.3 PB/s	1,041,354
PR	164 TB	1.4 ms	125.9 PB/s	7.7 TB/s	63.0 PB/s	1,542,398

6.2.2 With Injection Bandwidth Limitation

Figure 6.7 presents the projected performance of the UpDown system across 2 to 16,384 nodes under varying network injection bandwidth limitations. The GTEPS performance is calculated by projecting the number of traversed edges and dividing by the projected execution time, which is determined as the maximum of the program execution time and the network injection time, as described in Section 6.1.3. Due to computational constraints, real performance is simulated only for 2–16 nodes. As shown in Figure 6.7, the projected results align closely with the simulation data in this range and demonstrate near-linear speedup as the system scales.

To better understand the impact of injection bandwidth limitations on the full-scale UpDown system (16,384 nodes), Table 6.3 and Figure 6.8 report the projected exact performance number (GTEPS) and the relative performance compared to the ideal (unconstrained) bandwidth case, under injection bandwidth limits ranging from 1 TB/s to 9 TB/s per node.

Under a 1 TB/s constraint, UpDown applications achieve only 25% of the performance

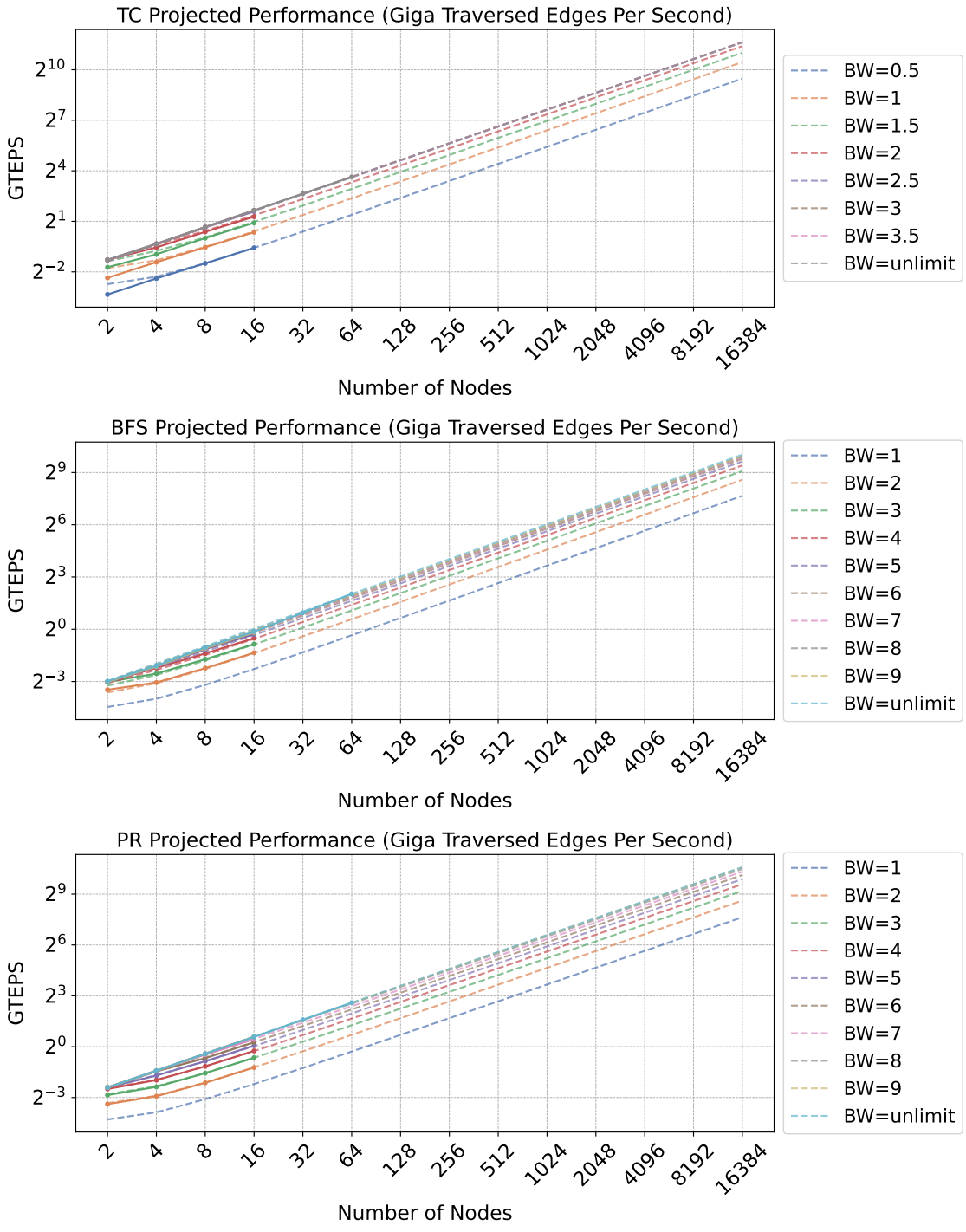

Figure 6.7: Projected and Real Performance (GTEPS) under different network injection bandwidth per node constrain on 2-16384 nodes.

Table 6.3: Full-scale UpDown system performance (GTEPS) under different network injection bandwidth per node constrain (project method is shown in Section 6.1.3)

GTEPS	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Unlimit**
TC	1,406k	2,713k	3,177k	3,181k	3,181k	3,181k	3,181k	3,181k	3,181k	3,181k
TC/Unlimit	44.1%	85.1%	99.6%	99.8%	99.8%	99.8%	99.8%	99.8%	99.8%	100%
BFS	203k	384k	542k	680k	797k	881k	935k	964k	987k	1,041k
BFS/Unlimit	19.5%	36.8%	52.1%	65.3%	76.5%	84.6%	89.8%	92.6%	94.7%	100%
PR	199k	394k	584k	768k	947k	1,117k	1,280k	1,432k	1,524k	1,542k
PR/Unlimit	12.9%	25.5%	37.9%	49.8%	61.4%	72.5%	83.0%	92.8%	98.8%	100%

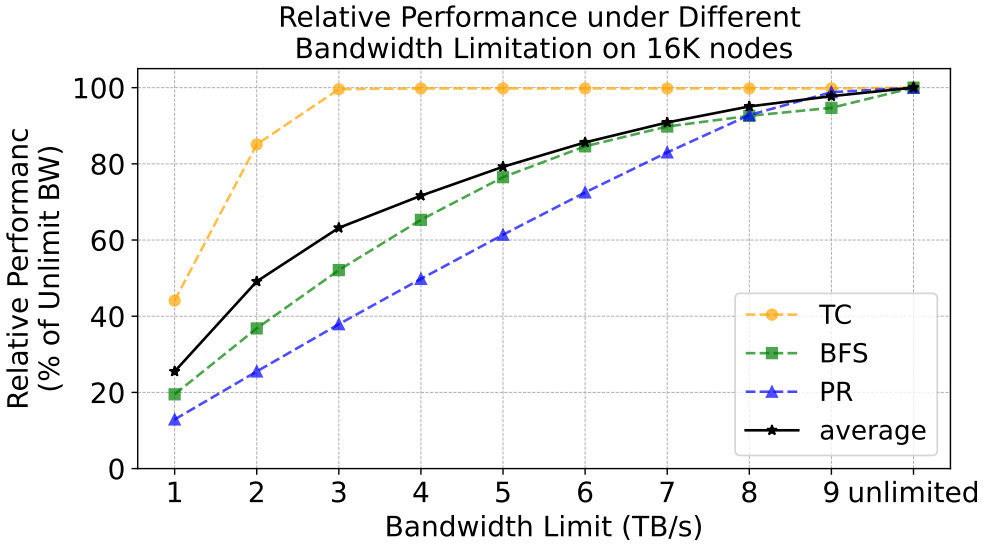

Figure 6.8: Relative Performance under Different Bandwidth Limitation on 16K nodes (baseline is unlimited network injection bandwidth limitation).

compared to the unlimited bandwidth scenario. At 2 TB/s, performance improves to an average of 50%. With 4 TB/s, performance reaches approximately 71%, and at 8 TB/s per node, all three applications achieve over 90% of their ideal performance.

Therefore, to avoid significant bandwidth bottlenecks, the per-node injection bandwidth should ideally exceed 8 TB/s. If future hardware such as Broadcom's 51.2 Tbps Co-Packaged Optics (CPO), expected by 2028 [31], is deployed, each node could support up to 6.4 TB/s. This would allow all three applications to reach at least 80% of their ideal performance.

CHAPTER 7

SUMMARY AND FUTURE WORK

7.1 Summary

In this paper, we analyze the network behavior generated by UpDown applications, including traffic patterns, packet size distributions, per-node injection bandwidth, and packet forwarding rates across up to 512 nodes. Our results show that UpDown applications achieve linear speedup with high utilization, generating uniform traffic with small packet sizes ranging from 16 B to 88 B. Among these, 16 B, 32 B, and 80 B are the most frequently used sizes. The injection bandwidth per node ranges from 2 TB/s to 7 TB/s, and the packet forwarding rate ranges from 50 Gpps to 250 Gpps—significantly exceeding those of other large-scale systems.

We also conduct a sensitivity study on network latency and injection bandwidth. Results show that UpDown can tolerate network latencies of 8,000 to 16,000 cycles (4–8 µs) by leveraging both multi-threading and intra-thread parallelism. In terms of injection bandwidth, UpDown effectively tolerates burstiness and fully utilizes available bandwidth.

Finally, we project network behavior for a full-scale 16K-node UpDown system and evaluate performance under various bandwidth constraints. Results indicate that linear scaling is achievable even under limited per-node injection bandwidth.

Table 7.1: Estimated application performance under varying network injection requirements

Injection BW	**Forwarding Rate**	**0.5 B/Inst App**	**1 B/Inst App**	**2 B/Inst App**
1 TB/s	25 Gpps	45%	20%	12.5%
2 TB/s	50 Gpps	85%	40%	25%
4 TB/s	100 Gpps	100%	70%	50%
6 TB/s	150 Gpps	100%	90%	72.5%
8 TB/s	200 Gpps	100%	97%	93%
9 TB/s	225 Gpps	100%	99%	99%

Table 7.1 summarizes the estimated normalized application performance on 16k UpDown node under varying injection bandwidths and corresponding packet forwarding rates, catego-

rized by communication intensity (0.5 B/instruction, 1 B/instruction, and 2 B/instruction) as observed in the simulations. Results suggest that 6.4 TB/s (51.2 Tbps) per-node injection bandwidth is a practical target for achieving near-peak application performance.

7.2 Future Work

During this study, several interesting directions emerged for future exploration:

- **Latency Tolerance in Other Systems:** Our analysis shows that UpDown can tolerate long network latency using intra-thread parallelism. It would be valuable to study whether and how other large-scale systems achieve similar latency tolerance.

- **Packet Prioritization under Bandwidth Limitation:** In our bandwidth limitation study, prioritizing DRAM acknowledgment packets improved PageRank performance by up to 40%. Exploring broader prioritization strategies could further improve overall system performance.

- **Routing for Workload-Aware Scheduling:** Traditional routing decisions focus on minimizing latency, maximizing bandwidth, and avoiding congestion. However, in HPC systems, the ultimate objective is to maximize overall application utilization and minimum execution time. In cases where destination nodes already have sufficient local workload, communication overheads—such as latency and bandwidth—can be effectively hidden. Therefore, routing decisions should consider the workload status of destination compute nodes, even at the expense of traditional network-optimal metrics. Workload-aware routing can improve global efficiency by prioritizing underutilized nodes. Alternatively, program-level send policies such as *send-to-shortest-queue* may be employed, directing network packets to the compute node with the least pending work during runtime.

REFERENCES

[1] A. Amer, H. Lu, P. Balaji, and S. Matsuoka, "Characterizing mpi and hybrid mpi+ threads applications at scale: Case study with bfs," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2015, pp. 1075–1083.

[2] Y. Pan, R. Pearce, and J. D. Owens, "Scalable breadth-first search on a gpu cluster," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018, pp. 1090–1101.

[3] T. P. Morgan. (2019) How cray makes ethernet suited for hpc and ai with slingshot. [Online]. Available: https://www.nextplatform.com/2019/08/16/how-cray-makes-ethernet-suited-for-hpc-and-ai-with-slingshot/

[4] Ethernet ii frame packet. [Online]. Available: https://support.huawei.com/enterprise/en/doc/EDOC1100174721/ea0a043c/ethernet-ii-frame

[5] D. Foley and J. Danskin, "Ultra-performance pascal gpu and nvlink interconnect," *IEEE Micro*, vol. 37, no. 2, pp. 7–17, 2017.

[6] (2024) Top500 highlights - november 2024. [Online]. Available: https://top500.org/lists/top500/2024/11/highs/

[7] L. Dhulipala, J. Łącki, J. Lee, and V. Mirrokni, "Terahac: Hierarchical agglomerative clustering of trillion-edge graphs," *Proceedings of the ACM on Management of Data*, vol. 1, no. 3, pp. 1–27, 2023.

[8] L. Dhulipala, G. E. Blelloch, and J. Shun, "Theoretically efficient parallel graph algorithms can be fast and scalable," *ACM Transactions on Parallel Computing (TOPC)*, vol. 8, no. 1, pp. 1–70, 2021.

[9] M. Nakao, K. Ueno, K. Fujisawa, Y. Kodama, and M. Sato, "Performance of the supercomputer fugaku for breadth-first search in graph500 benchmark," in *High Performance Computing: 36th International Conference, ISC High Performance 2021, Virtual Event, June 24–July 2, 2021, Proceedings 36*. Springer, 2021, pp. 372–390.

[10] Using el capitan systems: Hardware overview. [Online]. Available: https://hpc.llnl.gov/documentation/user-guides/using-el-capitan-systems/hardware-overview

[11] T. P. Morgan. (2024) "el capitan" supercomputer blazes the trail for converged cpu-gpu compute. [Online]. Available: https://www.nextplatform.com/2024/11/18/el-capitan-supercomputer-blazes-the-trail-for-converged-cpu-gpu-compute/

[12] S. Abraham, "Frontier architecture overview," 2024. [Online]. Available: https://www.olcf.ornl.gov/wp-content/uploads/Frontier-Architecture-Overview_Abraham.pdf

[13] S. Muralidharan, "Aurora exascale architecture," 2023, aTPESC-2023. [Online]. Available: https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2023/08/ATPESC-2023-Track-1-Talk-3-Servesh-Mulalidharan-Aurora.pdf

[14] Fugaku specifications. [Online]. Available: https://www.fujitsu.com/global/about/innovation/fugaku/specifications/

[15] About fugaku. [Online]. Available: https://www.r-ccs.riken.jp/en/fugaku/about/

[16] Y. Ajima, T. Kawashima, T. Okamoto, N. Shida, K. Hirai, T. Shimizu, S. Hiramoto, Y. Ikeda, T. Yoshikawa, K. Uchida *et al.*, "The tofu interconnect d," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2018, pp. 646–654.

[17] Tofu interconnect d of supercomputer fugaku and future prospects. [Online]. Available: https://nowlab.cse.ohio-state.edu/static/media/workshops/presentations/exacomm22/04_exacomm2022_ajima.pdf

[18] X. Gan, Y. Zhang, R. Wang, T. Li, T. Xiao, R. Zeng, J. Liu, and K. Lu, "Tianhegraph: Customizing graph search for graph500 on tianhe supercomputer," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 941–951, 2021.

[19] Nvidia gh200 grace hopper superchip architecture. [Online]. Available: https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper?ncid=no-ncid

[20] R. K. N. S. V. M. G. B. Michael Andersch, Greg Palmer and S. Ramaswamy. (2022) Nvidia hopper architecture in-depth. [Online]. Available: https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

[21] Nvidia h100 tensor core gpu architecture. [Online]. Available: https://resources.nvidia.com/en-us-tensor-core

[22] Nvidia h200 nvl. [Online]. Available: https://www.techpowerup.com/gpu-specs/h200-nvl.c4254

[23] A. Tirumala and R. Wong, "Nvidia blackwell platform: Advancing generative ai and accelerated computing," in *2024 IEEE Hot Chips 36 Symposium (HCS)*. IEEE Computer Society, 2024, pp. 1–33.

[24] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[25] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtarik, "Scaling distributed machine learning with In-Network aggregation," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 785–808. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/sapio

[26] B. Li, Y. Guo, Y. Wang, A. Jaleel, J. Yang, and X. Tang, "Idyll: Enhancing page translation in multi-gpus via light weight pte invalidations," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 1163–1177.

[27] A. Li, S. L. Song, J. Chen, J. Li, X. Liu, N. R. Tallent, and K. J. Barker, "Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 1, pp. 94–110, 2019.

[28] S. Zhang, "Scaling performance in multi-chip gpu systems: challenges and opportunities," Ph.D. dissertation, Ghent University, 2023.

[29] J. Lee, P.-C. Chiang, P.-J. Peng, L.-Y. Chen, and C.-C. Weng, "Design of 56 gb/s nrz and pam4 serdes transceivers in cmos technologies," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 9, pp. 2061–2073, 2015.

[30] Optical i/o: Ayar labs launches 8 tbps ucie optical chiplet for ai scale-up architectures. [Online]. Available: https://insidehpc.com/2025/04/optical-i-o-ayar-labs-launches-8-tbps-ucie-optical-chiplet-for-ai-scale-up-architectures/

[31] M. Mehta, "An ai compute asic with optical attach to enable next generation scale-up architectures," in *2024 IEEE Hot Chips 36 Symposium (HCS)*. IEEE, 2024, pp. 1–30.

[32] Ultimate guide to jumbo frames: Configuration and best mtu size. [Online]. Available: https://stonefly.com/resources/jumbo-frames-configuration-and-best-mtu-size/

[33] Configure the lustre network interfaces. [Online]. Available: https://support.hpe.com/hpesc/public/docDisplay?docId=a00113896en_us&page=Configure_the_Lustre_Network_Interfaces.html

[34] (2024) Hpe slingshot-a50002546enw quickspecs. [Online]. Available: https://www.hpe.com/psnow/doc/a50002546enw?jumpid=in_pdfviewer-psnow

[35] D. Roweth, G. Faanes, J. Treger, and M. Terpstra, "Hpe slingshot launched into network space."

[36]

[37] W. NVIDIA, "Nvidia nvswitch: The world's highest-bandwidth on-node switch," 2018.

[38] November 2024 bfs. [Online]. Available: https://graph500.org/?page_id=1332

[39] J. Sun, Z. Gao, D. Grant, K. Nawaz, P. Wang, C.-M. Yang, P. Boudreaux, S. Kowalski, and S. Huff, "Energy dataset of frontier supercomputer for waste heat recovery," *Scientific Data*, vol. 11, no. 1, p. 1077, 2024.

[40] Nvidia blackwell architecture technical brief. [Online]. Available: https://resources.nvidia.com/en-us-blackwell-architecture?ncid=no-ncid

[41] Nvidia b200 sxm 192 gb. [Online]. Available: https://www.techpowerup.com/gpu-specs/b200-sxm-192-gb.c4210

[42] A. Chien, A. Rajasukumar, M. Nourian, Y. Wang, T. Su, C. Zou, and Y. Fang, "Updown accelerator instruction set architecture (isa) v2.4," University of Chicago, Technical Report TR-2024-03, July 2024. [Online]. Available: https://newtraell.cs.uchicago.edu/research/publications/techreports/TR-2024-03

[43] A. Rajasukumar, J. Su, Yuqing, Wang, T. Su, M. Nourian, J. M. M. Diaz, T. Zhang, J. Ding, W. Wang, Z. Zhang, M. Jeje, H. Hoffmann, Y. Li, and A. A. Chien, "Updown: Programmable fine-grained events for scalable performance on irregular applications," 2024. [Online]. Available: https://arxiv.org/abs/2407.20773

[44] A. R. Yuqing Wang, T. Su, M. Nourian, A. P. Jose M Monsalve Diaz, J. Ding, C. Colley, W. Wang, Y. Li, D. F. Gleich, H. Hoffmann, and A. A. Chien, "Efficiently exploiting irregular parallelism using keys at scale," in *Proceedings of Conference Workshop on Languages and Compilers for Parallel Computing*, 2023.

[45] A. Rajasukumar, T. Zhang, R. Xu, and A. A. Chien, "Updown: A novel architecture for unlimited memory parallelism," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 61–77. [Online]. Available: https://doi.org/10.1145/3695794.3695801

[46] Y. Wang, S. Perarnau, and A. A. Chien, "Updown: Combining scalable address translation with locality control," in *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2024, pp. 1014–1024.

[47] Intelligence Advanced Research Projects Activity (IARPA). (2022) Agile: Advanced graphic intelligence logical computing environment. [Online]. Available: https://www.iarpa.gov/research-programs/agile

[48] D. Chakrabarti, Y. Zhan, and C. Faloutsos, *R-MAT: A Recursive Model for Graph Mining*, pp. 442–446. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611972740.43

[49] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 1, pp. 2–es, 2007, section 4.2.1.

[50] Y. Pan, Y. Wang, Y. Wu, C. Yang, and J. D. Owens, "Multi-gpu graph analytics," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017, pp. 479–490.

[51] E. Ginsberg, V. Alston, A. A. Wild III, A. Sheth, W. Liu, and R. Periakaruppan, "Generating traffic for testing a system under test," Apr. 7 2009, uS Patent 7,516,216.

[52] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, September 2009.