Quantized In-band Network Telemetry for Low Bandwidth Overhead Monitoring

Chanbin Bae[1], Kyeongtak Lee[2], Heewon Kim[1], Seongyeon Yoon[1], Junkyu Hong[1], Sangheon Pack[1], and Dongjin Lee[3]

[1]School of Electrical Engineering, Korea University, Seoul, Korea.
Email: {bin6050, harry0475, ysy0326, clickhjk123, shpack}@korea.ac.kr
[2]System Design Group, Samsung Electronics, Gyeonggi-do, Korea.
Email: kyeot00.lee@samsung.com
[3]Core Network Dev., ICT Infra Tech, SK Telecom, Gyeonggi-do, Korea.
Email: dongjin@sk.com

***Abstract*—Given the importance of robustness and resilience in emerging cloud-native networks, effective monitoring for fault detection is paramount and In-band network telemetry (INT) is a key candidate that enables real-time and fine-grained network monitoring with a programmable data plane. However, INT increases bandwidth overhead because network information is inserted directly into the packet header. In this paper, we propose a quantized INT (QINT) to effectively reduce overhead by considering the distribution of raw data. In QINT, the programmable switch encodes a raw telemetry item into a quantized bit stream using the Huffman coding scheme. To do this, QINT monitors the distribution of network telemetry items and encodes high-frequency data that are generated most of the time in a few bits. We implemented QINT on a programmable switch and our experimental results demonstrate that QINT can reduce the relative bandwidth usage by up to 60.6% compared to traditional INT, respectively.**

***Index Terms*—In-band Network Telemetry, Programmable Data Plane, Network monitoring, P4.**

I. Introduction

As technologies move toward emerging networks (e.g., 6G), mobile network operators (MNO) are adapting new network functions (NFs) and creating enhanced features based on AI/cloud-native architecture (e.g., 6G mobile core network). However, failure in such environments, especially in mission-critical services such as autonomous driving and industrial networks, may lead to serious accidents. In this context, it is expected that emerging networks will be implemented as robust and resilient to provide better quality of service (QoS) and quality of experience (QoE) to customers [16], [17]. To achieve this goal, real-time and fine-grained monitoring is a crucial technology for fault prediction, detection, and root cause analysis.

Programming protocol-independent packet processor (P4) improves the flexibility of the PDP by enabling the desired configuration of operations on networking devices such as switches, routers, and network interface cards [1]. Due to this flexibility of the programmable data plane (PDP), INT [2] facilitates detailed and real-time monitoring of internal network metrics such as link utilization, hop latency, and queue occupancy within the data plane by embedding telemetry information in packet headers. This telemetry item is then transmitted to a collector, enabling more precise network management.

To effectively monitor telemetry items, INT must ensure both high accuracy and minimal bandwidth consumption [6]. Accuracy means that INT should provide detailed information on each packet and accurate network telemetry details. This is crucial as inaccuracies in INT can lead to significant errors in network management, such as falsely identifying anomalous traffic. On the other hand, minimal bandwidth overhead indicates that INT should reduce its impact on standard data traffic during monitoring. Otherwise, excessive monitoring traffic could disrupt normal data flows, potentially leading to congestion due to telemetry items included in packets.

Despite the high accuracy that traditional INT offers, it fails to deliver low bandwidth overhead. This issue arises because the conventional INT collection process necessitates each PDP device along the route to include all telemetry items within the packet header, which could result in considerable bandwidth overhead and subsequent performance degradation [3], [6]. For instance, when a packet passes through 5 hops in the fat-tree topology and monitors three items (e.g., switch ID, hop latency, and queue occupancy), it inserts 12 bytes of telemetry items at each hop. Consequently, each packet carries 68 bytes of telemetry items (e.g., 8 bytes for INT header and 60 bytes for telemetry items of 5 hops), representing 4.53% of the maximum transmission unit (MTU) size (e.g., Ethernet has a 1500 bytes MTU). This results in a 20% reduction in goodput and a 25% increase in flow completion time [3].

To address this drawback, sampling-based INT [3], [4] and encoding-based INT [5], [6] have been introduced. Sampling-based INT schemes can reduce bandwidth overhead by inserting telemetry items into only several packets in a selective or probabilistic manner. For example, PINT [3] inserts telemetry items in a probabilistic manner, and DeltaINT [4] inserts telemetry items only when the difference between the current value and the last stored value is greater than the threshold.

978-3-903176-66-9

These schemes are effective in reducing the bandwidth overhead; however, they cannot provide high accuracy, since they do not support per-packet and per-node telemetry. On the other hand, encoding-based INT schemes can reduce bandwidth overhead by collecting telemetry items with fewer bits that can be decoded into the original monitoring information. LightGuardian [5] inserts a skecthlet in the packet header probabilistically with the flow-level telemetry item, and the end host reconstructs the complete sketch from the inserted sketchlets. However, it does not operate in a per-packet manner. Although these non-per-packet approaches can efficiently reduce bandwidth overhead, they cannot achieve full visibility (e.g., path trace) [6] that is critical in network diagnosis with INT. OffsetINT [6] provides a per-packet/node monitoring system. It inserts offset bits which are fewer than the original bits if the difference between reference value and current monitoring value is smaller than the threshold. However, these schemes collect telemetry items with static bit length without considering the distribution of the telemetry item. To the best of our knowledge, there is no existing work to reduce bandwidth overhead with the flexible quantized bit length of telemetry items while considering their data distribution.

Data compression is a widely used method to mitigate bandwidth overhead by minimizing data size. This technique allows data to be transmitted using fewer bits via various encoding schemes [14], [15]. An efficient way to shrink data size is by compressing the data based on the frequency of each value. Thus, understanding the data distribution is crucial for successful data compression. Concerning network states, data variations are typically minor, except during microburst traffic [4]. This suggests that most data are densely packed, with only a minor portion being dispersed. In such scenarios, encoding the highly frequent data into fewer bits offers more significant bandwidth reduction benefits than doing the same for less frequent data. Consequently, focusing on the data frequency to compress the bulk of the data into fewer bits is an effective strategy to reduce bandwidth consumption.

In this paper, we propose a novel quantized INT scheme (QINT) that significantly reduces bandwidth usage by taking into account the data distribution of telemetry items. Initially, we examine the frequency of the data for each item using per-packet telemetry information with the QINT configuration. Subsequently, we adjust the size of the encoding sections with flexible quantized levels determined by Huffman coding [7]. QINT minimizes bandwidth overhead by encoding high-frequency data sections with fewer quantized bits. To collect telemetry items with flexible sizes, we define a customized packet header and implement QINT with P4. Experimental results indicate that QINT can reduce relative bandwidth usage by up to 60.6% compared to traditional INT, while maintaining adequate accuracy with both per-packet and per-node monitoring. Our source code is accessible through the GitHub repository at https://github.com/Chanbin-Bae/QINT.

The remainder of this paper is organized as follows. We explain the design of QINT in Section II. Then, we present the results of the performance evaluation in Section III. Finally, we conclude the paper in Section IV.

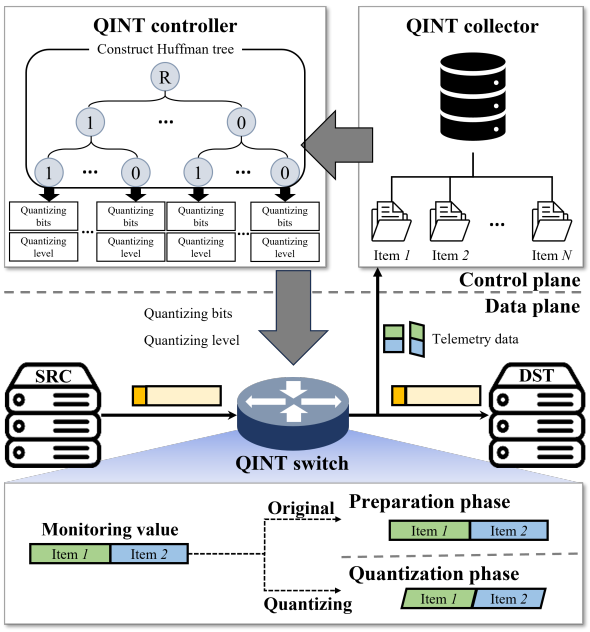


Fig. 1: System model of QINT.

II. Design of QINT

Figure 1 illustrates the QINT system model, which consists of the control plane (i.e., QINT collector and QINT controller) and the data plane (i.e., QINT switches). The control plane collects per-packet trace with the original bits from the QINT switches for a certain time interval (see the rectangular items in Figure 1) and determines the QINT configuration. With this trace and configuration, the control plane analyzes the distribution of each telemetry item and determines the quantization rules by constructing the Huffman tree. Along with the quantization rules, the QINT switches insert their telemetry items with the quantized bit stream (see the parallelogram items in Figure 1). After that, the control plane receives the quantized bit streams from the QINT switches and decodes them into the corresponding telemetry items. Detailed operations of the QINT control and data planes are described in Sections II-A and II-B, respectively.

A. QINT control plane

Within the QINT control plane, the QINT collector archives and supplies both the raw telemetry items and the quantized telemetry items from the QINT switches. The raw telemetry items are employed to formulate quantization rules during the preparation phase, while the quantized telemetry items are utilized to decode back into the equivalent telemetry items in the quantization phase.

In the preparation phase, the QINT control plane first determines the data section size, which is the configuration of QINT, to formulate quantization rules with the raw telemetry items. This data section size is used to establish the range unit for quantizing telemetry items. Based on the data section

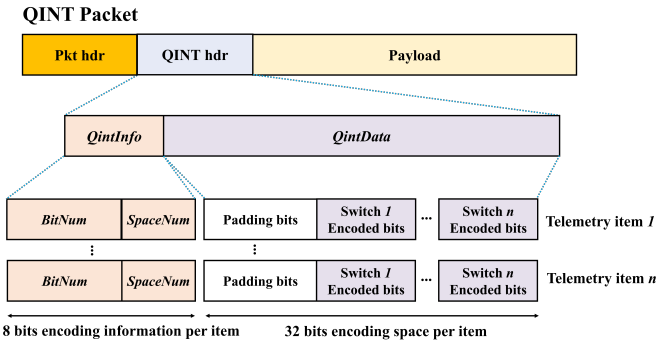


Fig. 2: QINT packet header format.

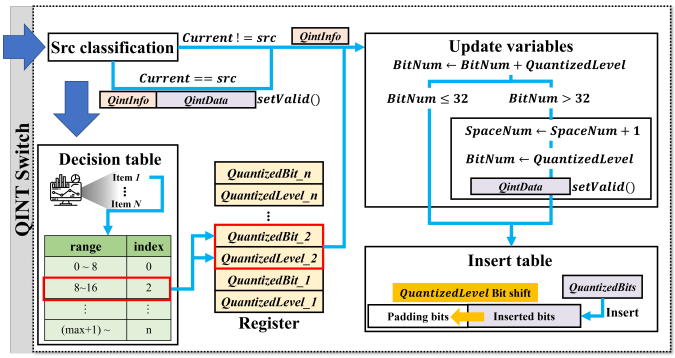


Fig. 3: QINT switch architecture.

size, the QINT controller divides the raw telemetry items into ranges and then runs the Huffman algorithm to decide the quantization granularity for each range. To run the Huffman algorithm to build a Huffman tree, the QINT controller defines nodes that are configured with symbols and weights. The symbol of a node is configured as the data section, and the weights are the telemetry item counts within that section. Then, the QINT controller extracts two nodes with the smallest weights and creates a new node with these extracted nodes as its child nodes (e.g., the left child node as a small weighted node and the right child node as a larger weighted node). The weight of the created node is configured as the sum of the weights of its child nodes. This process of creating a new node is repeated until only one parent node remains. Once a single parent node is left, the QINT controller assigns 1 to the left child node and 0 to the right child node, as shown in Figure 1. Consequently, the QINT controller determines the quantization rules (i.e., the quantized bit stream and the bit count of the quantized bit stream) along with the assigned bits in the Huffman tree to reduce bandwidth overhead. Through the Huffman coding scheme, QINT takes advantage of two aspects: effective data compression and decoding. Most telemetry item values tend to be highly concentrated [4], and the Huffman coding scheme allocates fewer bits to higher frequency occurrences, enabling monitoring of information with minimal bit usage in most cases. Additionally, the QINT controller, being knowledgeable about Huffman coding rules, can easily decode serialized bits through the characteristics of the Huffman coding. The last role of the QINT controller is to decode the quantized telemetry items received in the quantization phase. With the quantization rules, the QINT controller decodes the quantized bit stream into the middle value of each data section.

B. QINT data plane

The QINT switch allows collecting INT data in quantized bit streams with installed quantization rules. Due to the Huffman coding scheme, quantized bit streams have variable sizes (e.g., 1 bit∼22 bits) depending on the value and frequency of telemetry items. However, the P4 parser allows header parsing at the byte level so that quantized bit streams cannot be parsed flexibly. To address this issue, we define two additional headers *QintInfo* and *QintData*, as illustrated in Figure 2, to insert various sizes of quantized bit streams. The *QintData* header is a pre-allocated space (i.e., 32 bits) used for stacking the telemetry item by concatenating with the data from previous hops. If the pre-allocated space is exceeded with the insertion of the telemetry item of the current node, an additional *QintData* header is added. To insert telemetry items properly, the QINT switch need to know additional information, such as how many bits are stacked in *QintData* and the number of the *QintData* headers. To provide this information, we define *BitNum* and *SpaceNum* fields in *QintInfo*. *BitNum* indicates that how many bits are stacked in the current *QintData*, and *SpaceNum* represents the number of stacked *QintData* headers.

Figure 3 shows the architecture of the QINT switch. Based on this architecture, the QINT switch collects quantized telemetry items with *QintInfo* and *QintData*. The operation of the QINT switch is as follows. When a packet enters the switch, the QINT switch classifies whether the current node is a source node or a transit node. If the current node is the source node, it sets *QintInfo* and *QintData* to be valid. Otherwise, the QINT switch utilizes *QintInfo* to parse the *QintData* headers. Specifically, the QINT switch leverages the *SpaceNum* filed in the *QintInfo* header to parse the stacked *QintData* headers. Next, the QINT switch then employs the decision table to enforce the quantization rules. Through a range match operation, the decision table evaluates the telemetry item of the current node. Following this, it assigns an index to the register that corresponds to the appropriate range.

The QINT switch then reads *QuantizedBits* and *QuantizedLevel* from the register with the provided index. *QuantizedBits* indicates the quantized bit stream and *QuantizedLevel* indicates the bit length of *QuantizedBits*. The QINT switch uses *QuantizedLevel* to update *BitNum* to check if *QintData* exceeds the pre-allocated 32-bit space when *QuantizedBits* is inserted. If the updated *BitNum* is greater than the pre-allocated 32-bit space, *SpaceNum* is incremented by 1, and *BitNum* is set to *QuantizedLevel*. If *SpaceNum* is updated, the QINT switch adds the *QintData* header. After that, the QINT switch applies the insert table to insert *QuantizedBits* in the *QintData* header. It uses the updated *SpaceNum* as the matching key in the insert table to insert *QuantizedBits*

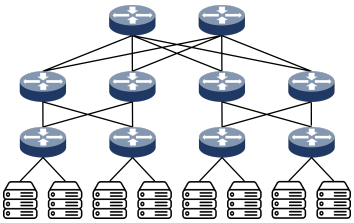


Fig. 4: Fat-tree topology.

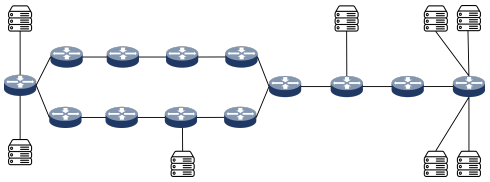


Fig. 5: Internet2 topology.

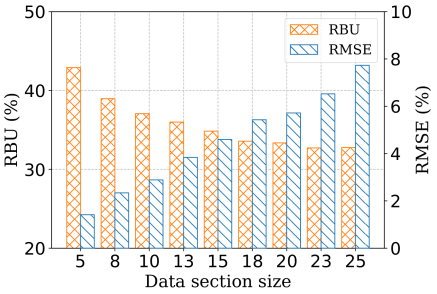


Fig. 6: Performance distribution.

into the appropriate *QintData* space. To insert *QuantizedBits* into *QintData*, the QINT switch applies a bit shift operation to *QintData* by *QuantizedBits*, then inserts *QuantizedBits*. Finally, the packet is forwarded to the next hop, and the same operation is repeated.

III. Performance Evaluation

We implemented and evaluated QINT on BMv2 software switches [12] and mininet [11]. As shown in Figure 4 and 5, we consider two topologies: 1) fat-tree and 2) Internet2 [8]. The fat-tree topology consists of 8 hosts and 10 programmable switches whereas the Internet2 topology has 8 hosts and 13 programmable switches. Each host generates its traffic based on the Web search [10] and Hadoop workload [9].

Baseline. We compare QINT with traditional INT [2], DeltaINT [4], and OffsetINT [6] with a representative telemetry item, hop latency. We configure the latency threshold as 10μ in DeltaINT, which allows us to collect hop latency with 32 bits if the difference is greater than the threshold. In the case of OffsetINT, we set the threshold to $2^{16}\mu$ as introduced in [6], which allows us to collect hop latency with 16 bits.

Metric. We employ the following metrics: 1) relative bandwidth usage (RBU), 2) root mean square error (RMSE), and 3) relative error (RE). RBU measures the total bandwidth consumed by the INT scheme compared to traditional INT and is calculated by dividing the total bits inserted with the INT scheme by the total bits inserted using traditional INT. These two metrics reflect the bandwidth overhead associated with the INT scheme. As an accuracy metric, RMSE can be expressed as

$$RMSE(\%) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2} \quad (1)$$

where n denotes the total number of data points. Also, $\hat{y}_i$ and y_i are the measured and real values, respectively. Meanwhile, RE is a relative error of the 50th and 99th percentile latencies.

Parameter setting. Let S be the data section size. As S increases, QINT achieves reduced bandwidth overhead, but this comes at the cost of lower accuracy. This is because a larger S allows for more values to be encoded into smaller bit streams. Conversely, when QINT decodes the quantized bit stream, it returns an average value for the section, and a larger S leads to decreased accuracy. Thus, there is a trade-off between bandwidth efficiency and accuracy based on the chosen QINT configuration.

Figure 6 illustrates how the QINT configuration balances between RBU and RMSE, demonstrating that RBU stabilizes as the size of the data section increases. Consequently, we choose a data section size of 8, where we observe the sharpest decline in relative bandwidth usage. It is important to note that the selection of the data section size might vary depending on the prioritization of RBU over RMSE, or vice versa.

A. Bandwidth overhead reduction

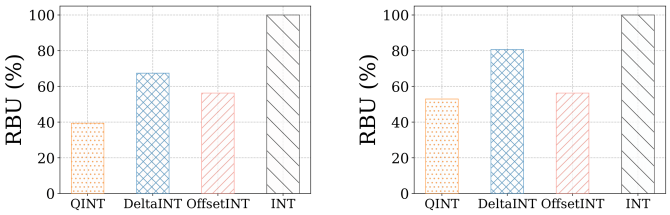
Figure 7(a) and (b) show the RBU in the internet2 topology with Web search workload and the fat-tree topology with Hadoop workload, respectively.

As illustrated in Figure 7(a), QINT employs the fewest bandwidth usage for data collection compared to other schemes. In particular, QINT, DeltaINT, and OffsetINT use 39.3%, 67.4%, and 56.2% of the bandwidth required in traditional INT, respectively. This efficiency is due to QINT's ability to insert telemetry items using significantly fewer bits most of the time (e.g., 2 bits per node). Furthermore, QINT avoids exceeding the 32-bit space allocated, since it uses fewer bits per node. For instance, in QINT, the most frequently occurring data can be quantized into 2 bits, which is much lower than the original 32 bits. Conversely, DeltaINT and OffsetINT collect telemetry items with static bits without considering the frequency of data values. Specifically, DeltaINT collects latency data in the original 32 bits for each node when the difference exceeds a predefined threshold, while OffsetINT collects latency information using 16 bits per packet and per node.

Furthermore, Figure 7(b) shows trends similar to Figure 7(a), demonstrating that QINT surpasses other schemes in performance with a Hadoop workload in a fat-tree topology. In particular, QINT consumes 52.93% of the RBU used by traditional INT, whereas DeltaINT and OffsetINT consume 80.66% and 56.25% of traditional INT, respectively.

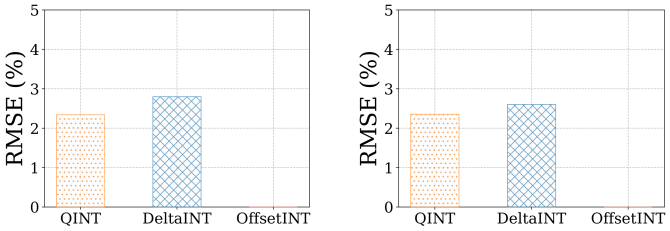
B. Accuracy

We evaluate the accuracy (in terms of RMSE and RE) of OffsetINT, DeltaINT, and QINT. Figure 8(a) and (b) depict the RMSE performance in the Internet2 and fat-tree topologies,



(a) Web search workload (b) Hadoop workload

Fig. 7: Bandwidth usage in hop latency measurement.



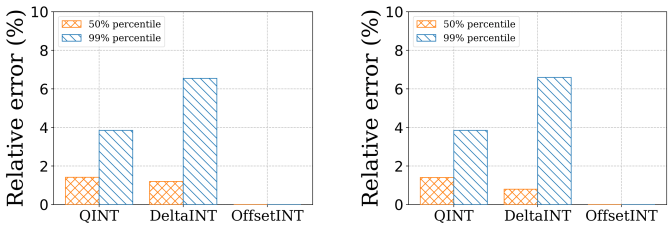
(a) Web search workload (b) Hadoop workload

Fig. 8: RMSE in hop latency measurement.

respectively. It is observed that QINT has a lower RMSE compared to DeltaINT in both the Internet2 and fat-tree topologies. Specifically, QINT achieves an RMSE of 2.34% and 2.35%, whereas DeltaINT has an RMSE of 2.79% and 2.60% in the Internet2 and fat-tree topologies, respectively. This indicates that QINT provides up to 16.38% higher accuracy than DeltaINT. Furthermore, 50% and 99% percentile latencies are measured in Figure 9(a) and (b), respectively. The results show that QINT achieves slightly lower accuracy in 50% percentile latency but significantly higher accuracy in 99% percentile latency compared to DeltaINT. This suggests that QINT can achieve more stable measurements with a lower bandwidth overhead than DeltaINT. These findings are due to DeltaINT collecting telemetry items only when the difference exceeds a certain threshold, while QINT collects all telemetry items. Even though OffsetINT provides the highest accuracy among the compared schemes due to its full recovery capability, QINT offers sufficiently high accuracy with per-packet per-node monitoring and lower bandwidth overhead than OffsetINT.

IV. Conclusion

In this paper, we proposed QINT, a quantized in-band network telemetry scheme per packet and per node that can provide low bandwidth overhead and accuracy simultaneously. QINT can reduce bandwidth overhead by flexible quantizing telemetry items into fewer bits than the original bits when considering the data frequency. Evaluation results demonstrate that QINT reduces the bandwidth overhead in terms of RBU by up to 60.6% compared to the existing INT scheme while maintaining sufficient accuracy. In our future work, we plan to conduct extended evaluations with large-scale commercial



(a) Web search workload (b) Hadoop workload

Fig. 9: RE in hop latency measurement.

MNO/networks to verify the feasibility of QINT in practical environments.

Acknowledgement

This work was supported in part by SK Telecom, in part by National Research Foundation (NRF) of Korea Grant funded by the Korean Government (MSIT) (No. RS-2024-00341965), and in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the MSIT (No. RS-2024-00398948).

References

[1] P. Bosshart *et al.*, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87-95, July 2014.
[2] T. P. A. Group, "In-band Network Telemetry (INT) Dataplane Specification, v2.1," Jun 2020. [Online]. Available: https://p4.org/p4-spec/docs/INT_v2_1.pdf
[3] R. B. Basat *et al.*, "PINT: Probabilistic In-band Network Telemetry," in *Proc. ACM SIGCOMM 2020*, July 2020.
[4] S. Sheng *et al.*, "DeltaINT: Toward General In-band Network Telemetry with Extremely Low Bandwidth Overhead," in *Proc. IEEE ICNP 2021*, November 2021.
[5] Y. Zhao *et al.*, "LightGuardian: A Full-Visibility, Lightweight, In-band Telemetry System Using Sketchlets," in *Proc. USENIX NSDI 2021*, April 2021.
[6] M. Qian *et al.*, "OffsetINT: Achieving High Accuracy and Low Bandwidth for In-band Network Telemetry," *IEEE Transactions on Services Computing*, vol. 17, no. 3, pp.1072-1083, May-June 2024.
[7] A. Moffat, "Huffman Coding," *ACM Computing Surveys*, vol. 52, no. 4, pp. 1-35, August 2019.
[8] S. Knight *et al.*, "The Internet Topology Zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765-1775, October 2011.
[9] A. Roy *et al.*, "Inside the Social Network's (Datacenter) Network," in *Proc. ACM SIGCOMM 2015*, August 2015.
[10] M. Alizadeh *et al.*, "Data Center TCP (DCTCP)," in *Proc. ACM SIGCOMM 2010*, August 2010.
[11] Mininet, Accessed: May 31, 2024. [Online]. Available: https://github.com/mininet/mininet.
[12] BMv2, Accessed: May 31, 2024. [Online]. Available: https://github.com/p4lang/behavioral-model.
[13] PINT github, Accessed: May 31, 2024. [Online]. Available: https://github.com/ProbabilisticINT/HPCC-PINT/tree/master/traffic_gen.
[14] S. Vaucher *et al.*, "ZipLine: in-netowrk compression at line speed," in *Proc. CoNEXT 2020*, November 2020.
[15] H. Lin *et al.*, "P4CTM: Compressed Traffic Pattern Matching Based on Programmable Data Plane," in *Proc. IEEE ISCC 2023*, August 2023.
[16] SK Telecom, "SK Telecom 6G White Paper version 1.0," August 2023.
[17] "SA1#106 (2024-05-27 – Jeju(KR)), Agenda #8," 3rd Generation Partnership Project (3GPP), https://portal.3gpp.org/ngppapp/TdocList.aspx?meetingId=60406