2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) | 979-8-3503-6460-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/IPDPSW63119.2024.00161

DUST: Resource-Aware Telemetry Offloading with A Distributed Hardware-Agnostic Approach

Mehrnaz Sharifian*‡, Diman Zad Tootaghaj⋆, Chen-Nee Chuah*, Puneet Sharma⋆

⋆Hewlett Packard Labs, *UC Davis, ‡Hewlett Packard Enterprise (Aruba Networks)

***Abstract*—In-device network monitoring has emerged as a promising alternative to centralized telemetry for gaining insights into the status and behavior of network devices. Despite its advantages of providing in-depth device telemetry and predicting failures in advance, it can impose substantial computational and storage burdens, potentially hindering networking devices' core switching and bridging functions. In light of this challenge, DUST system is introduced to dynamically distribute and offload in-device monitoring tasks by harnessing the available computational resources across network nodes. It is designed to be hardware-agnostic, making it deployable on switches, servers, DPUs, SmartNICs, and other relevant devices. Our initial experiments on a real data center testbed indicate that DUST can reduce CPU utilization by up to 50% and memory usage by up to 15% in the context of in-device monitoring workloads.**

We present a comprehensive system architecture that encompasses various nodes and discuss the flow of packets and message communications. To tackle one of the primary challenges posed by DUST—namely, the optimal relocation of computations while considering network performance constraints and controllable routing decisions—we mathematically formulate the problem as an Integer Linear Program (ILP), along with a heuristic algorithm to reduce the computational complexity. We thoroughly examine the effectiveness and scalability aspects of our algorithms by considering various network sizes and use cases.

***Index Terms*—network telemetry, monitoring offload, data processing units, optimization, scalability**

I. Introduction

Network monitoring is undeniably a crucial facet of network deployment. However, recently, its significance has intensified due to the considerable surge in network complexity, the intricacies of management, heightened security considerations, and the need for effective anomaly detection. Furthermore, monitoring the scaled-up traffic poses a myriad of challenges, including relentless traffic growth, device diversity, and load imbalances [1]. Traditionally, network monitoring was predominantly carried out in a centralized fashion, utilizing methods such as direct streaming telemetry, which involves collecting real-time monitoring data from network devices and transmitting it to a remote centralized location [2]. However, conventional network monitoring methods are limited by switch performance. Measurement process by itself adversely affects the device status and network state, resulting in inaccurate monitoring data. Thereafter, the emergence of software-defined measurement methods (SDN [3] and PDP [4]) has significantly increased the programability of network devices' control plane and data plane [5]. While software-defined measurements reshaped the traditional measurement approaches, its administrator's ability to manage the network devices themselves yet relies on centralized and legacy mechanisms such as SNMP, sFlow, Netflow, etc., [6] and limited information accessible via standardized MIB. However, as networks grow in size and complexity, these constraints bring up more controversial issues. Recently, in response to the challenges posed by large-scale networks with diverse device capacities, distributed monitoring approaches incorporating machine and deep learning analytics have emerged as promising solutions within the realm of networking [7]. However, the unavoidable challenges persist in the form of high computational demands and time consumption associated with centralized management locations, where data aggregation and machine learning operations take place.

In contrast to centralized monitoring, in-device network telemetry leverages the 'network device' compute to empower analytics in a distributed architecture. It allows for detailed observation and data collection precisely where user traffic traverses. It eliminates the extensive data traversal throughout the network and supports advanced applications such as multi-path reconstruction, dead-hop detection, and localization of latency bottlenecks acquired in case an anomaly is detected. Consequently, it facilitates the collection of usage-based, scaled, and fine-grained monitoring data, enabling real-time data collection, processing, and action [8], [9]. While obtaining detailed insights through network monitoring is valuable, in-device monitoring comes with computational and storage demands that can hinder the core switching and bridging functions of network devices. By consuming device capacity, existing telemetry faces the dilemma between resource efficiency (i.e., low CPU, memory, and bandwidth overhead) and full accuracy (i.e., error-free and holistic measurement) [10].

Conversely, even though certain nodes running in-device network monitoring face resource constraints, there are ample computing resources available in high-performance and industry-leading devices throughout the network. Discarding these resources would not be cost-effective for users. Additionally, collaborations between network switches and smart fabrics, such as Data Processing Units (DPUs), have led to the emergence of a new category of network switch solutions known as Distributed Services Switches (DSS) [11]. These DSS solutions offload essential functions from network switches to DPUs, including Deep Packet Inspection (DPI), stateful firewall, and load balancing. By leveraging this innovative technology, we believe that in-device telemetry is a

979-8-3503-6460-6/24/$31.00 ©2024 IEEE
DOI 10.1109/IPDPSW63119.2024.00161

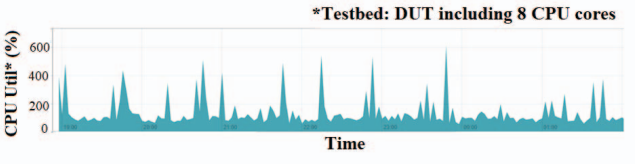


Fig. 1: CPU utilization of monitoring module.

remarkable offloading function to be adopted in future network devices and DSS switches.

To illustrate the extent of this excessive resource usage, we evaluated a CPU consumption of a modular in-device analytic engine on a database-driven network operating system installed on an enterprise switch [12]. We deployed several user-defined modular in-device monitoring agents, such as those monitoring network protocol health, Rx/TX packet rates, and device CPU and memory utilization. As depicted in Figure 1, our experiments revealed that the CPU utilization of the monitoring module reached around 100% average, spiking to as high as 600% (in 8 CPU cores DUT) when subjected to 20% line-rate VxLAN overlay traffic in a data-center topology.

In this paper, we introduce *DUST*, an innovative, dynamic, and distributed usage-based service telemetry solution that leverages available computing resources across the network, including (among others) DPUs, servers, and switches. DUST can be deployed in a hardware-agnostic system, enabling network-wide monitoring with in-depth device analysis. It significantly reduces device resource utilization through intelligent offloading and dynamic distribution of telemetry services, making in-device monitoring deployable across all network nodes. The ***key*** contributions of *DUST* are as follows:

- We propose a comprehensive system architecture that distributes the network monitoring load from nodes with high computational loads to nodes with lower loads, minimizing data movement costs. Our approach is hardware-agnostic and can be deployed on SmartNIC DPUs, network switches, and servers.
- Our offloading experiments on a real data center testbed demonstrate up to 50% savings in computing at scale in the context of in-device monitoring workloads.
- We also mathematically formulate the optimal workload distribution problem as an Integer Linear Program (ILP), targeting minimal response time and prioritizing data locality for compute vs. connectivity trade-offs.
- We evaluate the performance of the optimization algorithm on networks of varying sizes and compare it to a heuristic model aimed at simplifying large-scale networks in terms on parameters such as computational time, defined Heuristic Failure Rate (HFR), and the number of hops required to reach the destination.

DUST is a dynamic traffic-aware solution that periodically monitors the in-device computational load of all nodes and makes distributed monitoring decisions accordingly. Distributed monitoring is facilitated through a cloud-based 'DUST-Manager' module that provides network-wide visibility and a central point for configuring distributed in-device telemetry. Our solution provides controllable routes across all capable nodes and paths with efficient resource utilization, which can be reallocated for network monitoring. We design a flexible offloading system, allowing one or multiple nodes to fully or partially offload the excessive load to one or more destination nodes simultaneously.

Our approach relies on utilizing the abundant but often underutilized computing resources present in top-performing devices across the network. The emergence of DPUs and diverse computational capabilities in servers and network switches in data centers and High-Performance Computing (HPC) support this notion [13]–[15]. However, there is a concern about fully utilizing the expensive resources available in HPCs. A study of NERSC's Perlmutter, a state-of-the-art open-science HPC system [16], revealed that a quarter of CPU node-hours achieved high utilization, while GPU-accelerated nodes were utilized for only 0-5% of the node-hours [17]. These issues are typical in HPC systems where resources are primarily assigned on a per-node basis, hindering co-located workload interference.

II. Related Works

Previous research in the field of in-device and distributed network telemetry has explored a variety of approaches to extend and apply these technologies. Several studies have focused on distributed network monitoring, addressing the challenges of monitoring large-scale networks across wide geographical areas or systems inherently distributed in nature [5], [6], [8], [10], [18]–[25]. In [18], the authors developed a robust algorithm known as RoDic to tackle the flow-size computation problem while distributing network telemetry in a stateful manner. However, their approach does not adequately address the issue of selecting the optimal target node across the network with available spare resources. Additionally, this approach assumes that flows traverse between source and destination nodes regardless of network routing considerations. Li et al. [19] also used a distributed monitoring approach in their work, but they faced difficulties in computing the optimal solution for reasonably sized networks. They eventually resorted to comparing three heuristic algorithms. Their problem formulation did not consider dynamic resource utilization, capabilities of nodes, controllable route decisions for optimal placement, or flexible offloading approaches. In UDAAN [6], the authors implemented an in-device network monitoring system that includes user-defined analytic applications. Despite addressing various device-level use cases and proposing a framework for telemetry abstractions, their architecture did not thoroughly evaluate the resource-intensive downside of UDAAN-Local or propose a solution to address it. Authors of [10] designed OmniMon, a re-architecturing of network telemetry solution to address the trade-offs between resource efficiency and full accuracy. However, the proposed scheduler-based approach does not address the scalability concern of network telemetry, and still needs more resources to achieve their predefined objective of 'full accuracy'. Moreover, the split-and-merge solution lacks any optimal node selection for

distributed analytics. Methods such as optimal proactive monitor placement proposed in [20] contain specific assumptions about IoT devices, such as lossy links being monitored only by their extremities, resource constrained nodes with limited capabilities assigned for monitoring, and eventually utilization of a centralized monitoring mechanism. While many of these works assume that some destination distribution nodes exist, we contend that a more thorough investigation is imperative to ascertain the most optimized selection of target node for offloading. This aspect plays a crucial role in improving the distributed analytic implementations in network switches akin to how it is often deployed in IoT devices [21], [22].

Other papers, such as [5], [8], [23]–[25], have delved into in-band real-time distributed network telemetry management with focus on mitigating packet loss and reordering issues [23], and approximation techniques, as seen in PINT for reducing in-band telemetry overhead [24], or HPCC for studying congestion control aspects [25]. While In-band network telemetry has been extensively researched in academia, its implementation in the industry faces constraints and performance overhead, such as growing packet sizes [8].

Our contribution is centered around the concept of offloading in-network usage-based analytics for hardware-agnostic and heterogeneous nodes. To fully utilize the available compute resources of the network, we have proposed a corresponding routing control solution. It considers the dynamic state of the network and optimizes the response time (cost) accordingly. Previous methods for distributed in-device telemetry have primarily focused on IoT devices and have utilized outdated sampling-based telemetry. Additionally, they deployed permanent third-party aggregators (poller) for distribution, resulting in recurring costs to the network irrespective of node capacity, underutilized resources, network status, and routes.

III. Proposed Solution

Our *DUST* architecture, depicted in Figure 2, is versatile and can be deployed across various network topologies. However, our initial efforts have been focused on data center networks due to their time-sensitive applications and the critical role of swift troubleshooting facilitated by network telemetry services. Data centers have become a primary arena for the deployment of *DUST*, driven by the rapid adoption of DPUs [26], and as they can offload network functions from servers [27]. Major cloud providers such as Google Cloud Platform (GCP), Azure, and AWS have already integrated DPUs into their data center infrastructures to offload select network and RDMA functions, thereby freeing up valuable resources for application workloads [28]–[30]. *DUST* capitalizes on the availability of additional compute resources within the network, which can be harnessed by analytic engines. This allows for in-device monitoring or remote monitoring from neighboring devices via protocols like REST [31], gRPC [32].

A. DUST System Architecture

As demonstrated in Figure 2, the 'Service Controller' and Optimizer' play crucial roles in the selection of nodes for

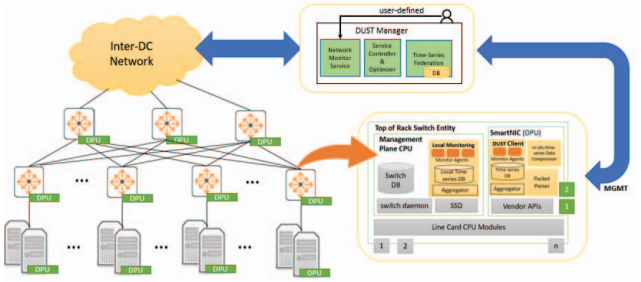


Fig. 2: DUST system architecture.

hosting the in-device DUST node cluster. Simultaneously, the 'Time-Series Federation' component performs the essential task of aggregating data throughout the underlying network. Our 'Network Monitor Service' (NMS) can initiate network monitoring either based on user input or through automated triggers. NMS collects a comprehensive set of metrics for the service and then transmits the pertinent information to the DUST client, effectively creating a 'Monitor Agent' for each required metric. These Monitor Agents continuously monitor updates within specific database (DB) tables on network devices and subsequently update the associated time series data. The 'Time Series Database' (TSDB) efficiently stores the metrics and rules established by these Monitor Agents. Our system includes in-situ data compression and packet parsing capabilities in SmartNICs, which aid in reducing data transfers and improving end-to-end performance.

B. DUST System Nodes and Workflows

The main components of DUST are DUST-Client and DUST-Manager, as shown in Figure 3. In the following, we will explain in more detail each component of DUST.

DUST-Manager: A decision node defines the most optimized destination monitoring node by evaluating network resource utilization, monitoring capabilities, and the number of monitoring agents. It includes *Network Monitoring Data Base (NMDB)* and *Optimization Engine* to place the monitoring processes and workloads. NMDB is a database used for keeping the current network status and utilization (e.g., network typologies, link utilization) and nodes' monitoring and offloading capabilities (e.g., resource utilization, number of user-defined monitoring requests, offloading capabilities and variables, etc.). Optimization engine deploys the network and monitoring states provided by and stored in NMDB to allocate the optimized node for running monitoring agents.

DUST-Client: It can be deployed on switches, servers, or any available compute resources such as DPUs across the network. Based on resource utilization and configurations, nodes' roles are defined as *Busy nodes, Offload-candidate nodes, None-offloading nodes*, and *Offload-destination nodes.*

To determine the roles of clients within the network, each monitoring node initially sends an *Offload-capable* message to DUST-Manager. In this message, a value of '1' signifies the node's willingness to participate in offloading, while '0' indicates its status as a *None-offloading node*. Subsequently,

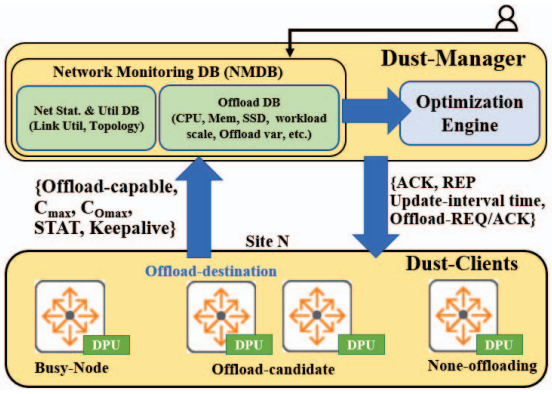


Fig. 3: DUST system nodes and packet flows.

DUST-Manager responds with an *ACK* message to each client engaged in the offloading process. Following this acknowledgment, these nodes are assigned distinct roles based on their dynamic capabilities. *Busy node* is defined as a client node whose resource utilization exceeds its user-defined threshold, denoted as C_{max}. Each client node initially conveys its C_{max} value to DUST-Manager, representing the maximum resource capacity that the node can sustain without requiring offloading. Conversely, DUST-Manager identifies *Offload-candidate nodes* as client nodes with resource utilization below their designated threshold, referred to as CO_{max}. These thresholds determine the maximum acceptable resource utilization for a node to be considered as a candidate for offloading. In simpler terms, a node is designated as an Offload-candidate by the Manager if its current resource utilization is lower than the previously defined CO_{max} value. Lastly, *Offload-destination nodes* are nodes designated to host the monitoring workload originally assigned to one or more Busy nodes.

For a dynamic and lossless offloading process in our system, client nodes send periodic *STAT* messages to the Manager node. These provide real-time updates on the client's usage of device services and monitoring, regardless of their current status. The frequency of these updates is determined by the *Update-Interval Time*, setting by the Manager's *ACK* message. This enables a Busy node to reclaim its local resources when they become available, while an Offload-destination node can redirect the workload to another node if it becomes busy.

Our solution utilizes user-defined interval times, typically in minutes, which align with the recommended collective interval times of enterprise networks and telemetry tools like Cisco, VMware, and Palo Alto Networks [33]–[35]). This approach allows for a better average view of the network's state while the optimization engine evaluates the offloading process requirements and determines the offload-destination nodes and controllable routes towards them. Lastly, the transition state is initiated upon acknowledgment of destination nodes.

DUST Monitoring Placement Workflow: When a Busy node is detected, DUST-Manager initiates the monitoring offloading distribution process by deploying the optimization engine (discussed in the following section). The optimization engine determines the Offload-destination nodes, which are then notified by the Manager node through an *Offload-Request* message, followed by an *Offload-ACK*. Monitoring data, denoted as D_i and originating from a Busy node, is subsequently redirected toward the acknowledged Offload-destination node.

C. Post-offloading Process in DUST

Following a successful offloading operation, we define post-offloading processes designed to uphold system performance while simultaneously monitoring the health status of the offloaded workloads.

QoS Guarantees: Monitoring data offloaded to a remote node is assigned the lowest priority value, ensuring a guaranteed quality of service for the remote node. This prioritization allows for the monitoring data to be safely discarded in the event of network congestion or overload. Consequently, remote nodes participating in the offloading process are not expected to experience any traffic loss.

Offload-destination Node Status: Offload-destination node needs to send *Keepalive* message to DUST-Manager and verify its offloading operational state for remote monitoring of a Busy node. As a result, the malfunctioning destination-node is diagnosed and substituted with a replica node. Manager notifies this node by sending it a *REP* message.

IV. Network Monitoring Placement

In this section, we tackle the challenge of selecting the optimal destination node for offloading monitoring tasks from heavily burdened nodes. To address this, we formulate the Minimum Cost Optimization Problem as an Integer Linear Programming (ILP). Given the high computational complexity of our problem, we also propose a heuristic algorithm to efficiently solve it in large-scale networks. As one of the key contributions, our algorithms determine a controllable route solution by evaluating all potential paths between each Busy node and the designated Offload-candidate nodes. We leverage data sourced from NMDB to facilitate DUST-Manager in identifying Busy nodes and potential Offload-candidate nodes. Subsequently, our optimization engine selects the destination node while prioritizing the shortest response time route to it.

A. Assumptions and Objectives

Before delving into our solution and algorithm, we define its assumptions and optimization objectives are as follows.

Assumptions: Data center, WAN, and backbone networks are the most stable networks by deploying high performance computing servers along with efficient switches and high availability nodes. However, these networks are vigorously vulnerable to extensive transient server workloads and data planes across the network. Thus, network monitoring is required to detect and localize the overloaded failures caused by either primary device functionality or its monitoring duties. We assume all nodes participating in the offloading process carrying monitoring tasks, and willingness to get offloaded in case of overloading. In our assumption for DUST as a usage-based device telemetry, data collection, monitoring, and

any further alerts or actions are assumed to be real-time and continuous tasks. We assume a stable network with overloaded nodes (Busy nodes) that are candidates for offloading to reachable remote nodes (Destination nodes). Likewise, stable (none-overloaded) nodes exist initially in the network with the probability of getting overloaded as a result of launching scaled user-defined device telemetry. Thereupon, all network nodes can be monitored with corresponding reachable nodes while priority is within closer nodes to achieve our objective of minimum response time. For better use of the network and optimization resources, in the large-scale network, max-hop is assumed and evaluated in accordance with the network use case and applications' required response time. Given the diverse functionality and platform capability of network devices, we assume custom-defined values for CO_{max} and C_{max}, reflecting node deployment and persona. Subsequently, the utilized resource of the Offload-candidate node is retained below CO_{max} to maintain its primary tasks in addition to the offloaded monitoring workloads of a Busy node, before getting overloaded itself. Equivalently, offloading of a Busy node is assumed whenever its resource utilization is at or higher than C_{max} for most usage of switch capacity. We assume that distributing the monitoring task is managed centrally with DUST-Manager while monitoring itself, data collection, processing, and aggregation, are achieved in a distributed device-level approach. By offloading monitoring task capabilities of a Busy node to remote nodes, zero computational load associated with transferring monitoring workloads over relay nodes is assumed to retain the dynamic state of relay nodes unchanged throughout a given offloading process. As a resource awareness system, device persona and status are assumed to be provided to DUST-Manager. We assume a homogeneity between source-overloaded nodes and selected nodes for offloading. Thus, offloading monitoring agents from the source node raises equivalent resource utilization in the destination node. It is noteworthy that the assumption of homogeneity does not negate the presence of heterogeneous node capabilities of DUST. The aforementioned assumption was made to simplify our optimization algorithm and based on the unavailability of 'scaled' data center nodes with offloading capability. In industry implementations, it can be adjusted with a coefficient factor relating two endpoint platform capacities.

Objectives: In distributing the network monitoring tasks, we tackle the challenge of selecting the optimal destination nodes and path towards it considering the minimum response time objective. Overloaded nodes' monitoring tasks are offloaded to a remote node with minimal hops distance priority whenever minimum response time is achieved. Similarly, minimal response time path and destination node are selected whenever a solution with the equivalent number of hops exists. Thus, bandwidth usage and data movement across relay nodes are minimized. Consequently, monitoring the workload placement of one Busy node by selecting one or multiple optimized remote nodes is applicable. The same objective is considered when multiple nodes are associated with offloading one excessively overloaded Busy node, attributing the flexible offloading objective of our proposed solution. The objective is to detect the potentially overloaded nodes (Busy node) while the node is not overloaded but efficiently utilized, and hereafter dynamically offload and redistribute all extra workload to a destination node while not overloading it.

B. Optimization Model Definition

Given an undirected graph $G = (V, E)$, where V is the set of nodes and E is the edges connecting the nodes, the goal is to select a subset of nodes from the set of Offload-candidate nodes, V_o, to efficiently offload monitoring tasks of Busy nodes, V_b, with minimum cost. We assume zero computational loads associated with transferring monitoring workloads over relay nodes ($v \notin V_b \cup V_o$) to a destination node. We define x_{ij} to be the continuous decision variable of offloading x_{ij} amount of the monitoring tasks capacities from the Busy node $i \in V_b$ to the Offload-candidate node $j \in V_o$. Let C_j be the percentage of the node j's utilized capacity. The network monitoring process compares C_j with respect to C_{max} and CO_{max} to identify node j as a Busy or Offload-candidate node. We define C_{max} and CO_{max} as maximum node capacity to be considered as Busy node and Offload-candidate node, respectively. We consider pre-defined values for C_{max} and CO_{max} to use max resources of Busy nodes, while not overloading the monitoring destination nodes computation after offloading. Consequently, a node is defined as a Busy node by DUST-Manager if its utilized capacity is greater than the predefined value C_{max}. Similarly, a node is defined as an Offload-candidate node if its utilized capacity is lower than the predefined value CO_{max}. In our formulation, V_o denotes the set of Offload-candidate nodes $\forall o \in V$ if $C_o \leq CO_{max}$ while V_b denotes the set of Busy nodes $\forall b \in V$ if $C_b \geq C_{max}$.

Furthermore, we use the notation D_i to represent the volume of usage-based in-device monitoring data to be offloaded and transferred to a remote destination node, measured in megabits (Mb). We denote $Lu_{i,j}$ as the utilized bandwidth, expressed in megabits per second (Mbps), between each pair of nodes i and j. This parameter reflects the total data plane traffic flowing across the network through the edge ij. It is determined by multiplying the physical link bandwidth and the dynamic utilization rate resulting from the data in transit.

Additionally, we define $Tr_{i,j}$ as the response time, measured in seconds, between nodes i and j. This metric quantifies the speed at which monitoring data traverses the network. Thus, we calculate $Tr_{i,j}$ as the sum of the response times for all available routes to the destination. Specifically, it is computed as $\frac{D_i(\text{Mb})}{Lu_{i,j}(\text{Mbps})}$ across all potential paths.

In our consideration of controllable routes, we account for all feasible paths between a Busy node $b \in V_b$ and an Offload-candidate node $o \in V_o$, denoted as $p = \{r_1, r_2, ..., r_n\}$. Here, each r_i comprises the edges forming the ith path, represented as $r_i = \{e_k - e_l - e_m\}$, while another path might be denoted as $r_j = \{e_n - e_o\}$, and so on.

An illustrative example: To further clarify the discussion, consider the network topology in Figure 4 with 7 nodes

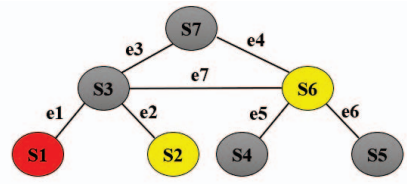


Fig. 4: Network with one busy node (red) and two offload-candidate nodes (yellow)

TABLE I: SUMMARIZATION OF NOTATIONS.

Notation	Explanation
$G = (V, E)$	an undirected graph where V represents the set of nodes and E is the set of links in graph/network topology G
x_{ij}	Continuous optimization decision variable
C_{max} (%)	Busy node's threshold capacity
CO_{max} (%)	Offload-candidate node's threshold capacity
C_j (%)	Utilized capacity of node; $\forall j \in V$
$D_i(Mb)$	Monitoring data of node i; $\forall i \in V$
$Lu_{i,j}(Mps)$	Link utilization bandwidth between node i to j
p	Set of all reachable paths between each pair nodes i and j $\forall i \in V_b$ and $\forall j \in V_o$
$Tr_{i,j}$	Response time (in seconds) between each node i and j
$T_{rmin,i,j}$	Minimum response time (in seconds) between each node i and j among all paths p
X_{min}	Nodes' minimum usage capacity
Cs	Total resources to be offloaded from Busy nodes
Cd	Total available resources of Offload-candidate nodes
β	Optimization objective

and 7 edges. In this scenario, we have a single Busy node (S1) and two Offload-Candidate nodes (S2 and S6). Our objective is to determine the optimal destination and route for offloading the excess workload from node S1 to either S2, S6, or both. Our solution offers flexible offloading, allowing a Busy node to be offloaded either partially or entirely to one or more Offload-destination nodes. Similarly, it supports the simultaneous offloading of multiple Busy nodes to a single Offload-destination node. For illustration, let's define four potential routes: $r_1 = \{e_1 - e_2\}$, $r_2 = \{e_1 - e_3 - e_4\}$, $r_3 = \{e_1 - e_3 - e_4 - e_7 - e_2\}$, and $r_4 = \{e_1 - e_7\}$. We can now form a set p containing these routes, denoted as $p = \{r_1, r_2, r_3, r_4\}$, to explore the optimal offloading.

Further, variable Cs_i is defined to be the amount of extra load for each Busy node i that needs to be offloaded to the Offload-candidate nodes. Thus, the total load to be offloaded in the system is equal to $Cs = \sum_{i \in V_b} Cs_i$. Similarly, Cd_j is defined as the available capacity of an Offload-candidate node j. Thence, every offload candidate node j can contribute to the offloading process with its Cd_j spare capacity, and total available resources in the system are defined as $Cd = \sum_{j \in V_o} Cd_j$. Table I summarizes the notations of our formulation.

C. Optimization Formulation

In our model, we define the response time for data movement from each Busy node i to an Offload-candidate node j as $Tr_{i,j}$, as expressed in Equation 1. The response time is computed based on the amount of monitoring workload D_i being transferred across all edges on the selected path r_k.

$$Tr_{i,j}(r_k) = \sum_{\forall e \in r_k} \left(\frac{D_i}{Lu_e}(sec)\right) \quad \forall i \in V_b, \forall j \in V_o, \quad \forall r_k \in p \tag{1}$$

where p is the set of all paths between i and j, $p = \{r_1, r_2, ..., r_n\}$; and r_k consist of the edges on that paths.

Further, we introduce $T_{rmin,i,j}$ to identify the minimum response time between each pair nodes {i,j} among all possible paths p for $\forall i \in V_b, j \in V_o$, respectively (Equation 2).

$$T_{rmin,i,j} = Min_{r_k \in p}(Tr_{i,j}(r_k)) \quad \forall i \in V_b, \forall j \in V_o \tag{2}$$

The constrained network offload optimization problem (min-cost) aims at minimizing the monitoring data transfer costs, which can be formulated as follows:

$$\text{Minimize} \quad \beta = \sum_{i \in V_b} \sum_{j \in V_o} x_{ij} * T_{rmin,i,j}$$

$$\text{subject to} \sum_{i \in V_b} x_{ij} \leq Cd_j \qquad \forall j \in V_o \tag{3a}$$

$$\sum_{j \in V_o} x_{ij} = Cs_i \qquad \forall i \in V_b \tag{3b}$$

$$Cs_i = C_i - C_{max} \qquad \forall i \in V_b \tag{3c}$$

$$Cd_j = CO_{max} - C_j \quad \forall j \in V_o \tag{3d}$$

$$C_i \in [x_{min}, 100] \qquad \forall i \in V \tag{3e}$$

Constraint 3a shows that the volume of monitoring data eligible for offloading from all Busy nodes to an Offload-candidate node j must not exceed its available spare capacity denoted as C_{d_j}. As a result, our distributed solution does not overload a destination node while offloading one/multiple Busy nodes. Constraint 3b shows that the total amount of load offloaded from a Busy node i should be equal to its total load that needs to be offloaded. In our formulation, for each Busy node i, we define Cs_i as the difference between C_i and a predefined value C_{max} (as shown in 3c). Similarly, for each Offload-candidate node j, we define Cd_j as the difference between a predefined value CO_{max} and C_j (as indicated in 3d). This signifies that each Offload-candidate node j can contribute to offloading based on its available spare capacity C_d. The amount of node capacity for the ones partitioning in offloading is assumed between a minimum value of x_{min} and a maximum value of 100, constraint 3e. Depending on the node function, platform capability, and its average minimum resource utilization in the network, x_{min} is adaptable.

D. Heuristic Algorithms

To solve the minimum cost offload problem in large-scale networks and in a time-efficient manner, we also design a heuristic algorithm shown in Algorithm 1. Heuristic algorithm starts with selecting all available Busy nodes, V_{hb}, and their corresponding utilized capacity, C_k for $\forall k \in V_{hb}$. Here, we use the same metrics to define Busy nodes, $C_k \geq C_{max}$, and workloads to be offloaded, $Cs_i = C_k - C_{max}$. However, in the heuristic algorithm, Offload-candidate nodes, V_{hoj}, are distinctly being selected for every Busy node, V_{hbi}, and within its directly connected node with available compute

Algorithm 1: Heuristic Algorithm of min-cost problem

for *every node* $k \in V$ *of graph* $G = (V, E)$ *and node capacity* C_k **do**
  Determine set of Busy nodes, $V_{hb} = \{V_{hb0}, V_{hb1}, ..\}$ if $C_k \geq C_{max}$
  for *every Busy node* $i \in V_{hb}$ **do**
    Determine set of Offload-candidate nodes, $V_{ho} = \{V_{ho0}, V_{ho1}, ..\}$, including every node j if $C_j \leq CO_{max}$ and within shortest path of **max-hop=1** to a given Busy node V_{hbi}
    Calculate $Cs_i = C_i - C_{max}$
    for *every Offload-candidate node* j *in set* V_{ho} *, calculate* $Cd_j = CO_{max} - C_j$ **do**
      Define continuous optimization decision variable X_{ij}
      for *every given Busy node* i *and within its Offload-candidate nodes set* V_{ho} **do**
        Minimize β (Equation 3) for defined heuristic set of nodes and X_{ij}

resources, if exist. In other words, every V_{hbj} is merely offloaded to one-hop accessible Offload-candidate nodes of that Busy node. While the optimization algorithm can offload monitoring workload to n-hop distance for every Busy node, in the heuristic approach, there is a probability of (partial or full) failure in offloading with minimum cost next-hop problem. To evaluate the offloading performance by deploying the heuristic algorithm, we defined Heuristic Failure Rate (HFR) as the ratio of resources that failed to be offloaded in heuristic selection to the total of resources required to be offloaded at any given network state. As indicated in Equation 4, for every Busy node i, Cse_i is defined as the amount of resources incapable of getting offloaded to its one-hop distance Offload-candidate node.

$$HFR(\%) = \frac{\sum_{i \in V_{hb}} Cse_i}{\sum_{i \in V_{hb}} Cs_i} * 100 \tag{4}$$

Complexity Analysis: With p defined as feasible routes between each desired pair of nodes {i,j}, and the objective of our optimization algorithm to minimize β (Equation 3), let $F(|p|)$ be the time complexity of moving data between nodes. In a k-port fat-tree topology, maximum $|p|$ between each node pair can be in the order of k^2 [36]. Assuming a situation with all $k(k+1)$ nodes actively being either Busy nodes or Offload-candidate nodes, $F(|p|)$, time complexity of optimization computation can be in the order of $k^4(k+1)^2 \approx k^6$. In order to reduce the optimization complexity for large sized networks, heuristic algorithm is proposed and evaluated in response to various network sizes. By limiting the number of Offload-candidate nodes for each Busy node to solely one-hop paths offloading destinations, $F(|p|)$ for our heuristic algorithm reduced to $k^2(k+1) \approx k^3$. Our results show that it drastically decreases the computation time of algorithms and consequently the offloading process.

V. Performance Evaluations

We deployed two distinct testbeds to assess our comprehensive system-level solution. The first testbed, explained in (V-A), consisted of a real prototype with a VxLAN data center topology and commercial enterprise switches with support for in-device telemetry. This testbed aimed to replicate a production environment, taking into account all the subtleties and practical and operational constraints of nodes with heterogeneous capabilities in real-world industry-scale hardware. Additionally, we implemented a simulator using the Gurobi optimization toolkit to address the scalability of our optimization algorithms. This simulator, (V-B), can evaluate the proposed approach for large-scale networks.

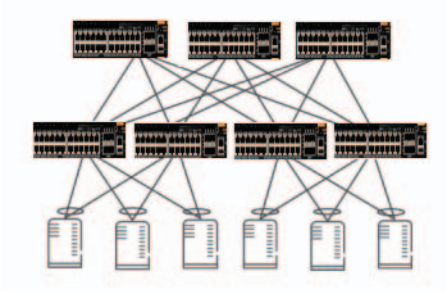


Fig. 5: Test bed topology

A. DUST Offloading Analysis and Evaluation

To evaluate the performance advantages of delegating monitoring computations, we implemented analytic engines in a data center architecture, illustrated in Figure 5. This topology employs modular in-device analytic engines (Python-based) installed on the network OS of a commercial enterprise switch, HPE Aruba 8325 with 8 CPU cores, 16 GB RAM, and 64GB SSD disk specifications [12]. Moreover, 10 user-defined monitoring agents are installed on the device for monitoring critical features of the deployed testbed[1]. We then analyzed device resource utilization in scenarios where user-defined monitoring agents were offloaded remotely using DUST and conducted a comparative analysis between local monitoring and offloaded mechanisms.

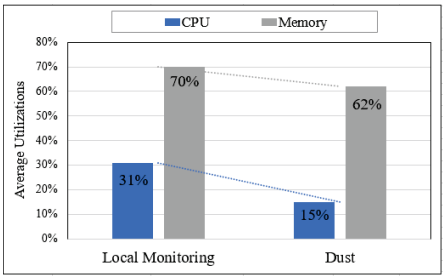
As illustrated in Figure 6a and Figure 6b, our results demonstrates a substantial average CPU and memory reductions of 52% (from 31% to 15%) and 12% (from 70% to 62%), respectively. In this set of experiments, retaining around 1.2 GiB memory usage indicates that monitoring workloads are perfect offloading candidates for network switches.

B. DUST Scalability Analysis and Evaluation

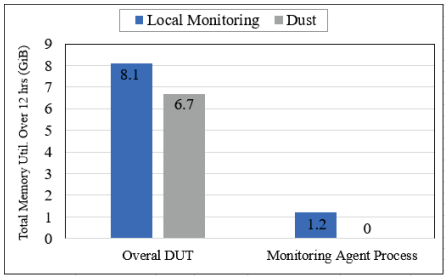
We evaluate the scalability of DUST proposed optimization algorithm in two defined network sizes, namely small-scale and large-scale networks. We implemented our network monitoring offload algorithms in Python and by Gurobi optimization toolkit [37], on a 6-core, 2.60 GHz, 16 G RAM cluster.

A small-scale network is characterized as a network, data center pod, or network zone comprising 20 or fewer network

[1]Agents named as routing protocols, software and network health, software functions and system resource utilization e.g., CPU/Memory, Rx/Tx packet rates on interfaces, link states, system temperature and hardware health, fault finder (such agents are prevalent and described in other works e.g. [6]).



(a) Avg. resource utilization.



(b) Total resource utilization.

Fig. 6: Memory and CPU resource utilization comparison in DUST and local monitoring.

nodes. In our experiments, it's equivalently represented as a three-level 4-k port fat-tree topology with 20 nodes and 32 edges. Conversely, a large-scale network is defined as a network or data center architecture with more than 20 network nodes. We classify fat-tree topologies with 8-k, 16-k, and 64-k ports as large-scale size networks with 80, 320, and 5120 nodes along with 256, 2048, and 131072 edges, respectively.

• **Small-Scale Network Evaluation:** We selected fat-tree 4-k ports for performance evaluations of small-scale networks in addition to analysis of some optimization algorithm' settings. To increase the probability of a 'feasible' optimization solution, we define the Δ parameter as follows:

$$\Delta_{io} = \frac{\Delta_o}{\Delta_b} = \frac{CO_{max} - x_{min}}{100 - C_{max}} \quad , \quad \Delta_{io} \geq K_{io} \qquad (5)$$

This parameter assists the user in selecting optimal user-defined values for C_{max} and CO_{max}, and improves the efficiency of offloading and optimization processes by reducing the rate of impossible optimization. As shown in Figure 7, our experiments in the 4-k port topology, conducted over 1000 iterations, reveal an Infeasible Optimization (io) Rate ranging from 0.2% to 69%, with Δ_{io} falling within the range of 3.5 to 0.8, respectively. These results suggest setting the user-defined K_{io} value (related to C_{max}, CO_{max}, and x_{min}) to be greater than or equal to 2. It aims to reduce the failure rate in the optimization process of a given network system. However, it is important to note that excessive increase of optimization parameters, such as C_{max} and CO_{max}, and consequently Δ_{io}, can lead to a higher likelihood of overloaded nodes and potential network downtime.

Assuming that Δ_{io} falls within the recommended range, we conducted measurements of computation time, averaged over 100 iterations on the small-scale topology as depicted in Figure 8. The results reveal that with no max-hop limit defined, the maximum computation time for optimization remains below 3.5 seconds. This finding underscores the suitability of a small-scale network of this size for accommodating time-constrained applications. When considering a threshold time value of 0.5 seconds, a recommended max-hop of 10 in the 4-k architecture is suggested. The strict threshold time requirement on the order of less than a second is necessary for facilitating the execution of multiple parallel optimization processes within DUST-Manager in addition to the actual offloading and rerouting

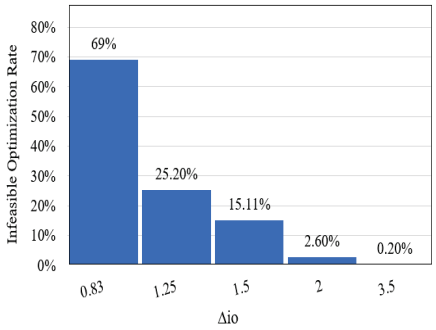


Fig. 7: Infeasible Optimization rate

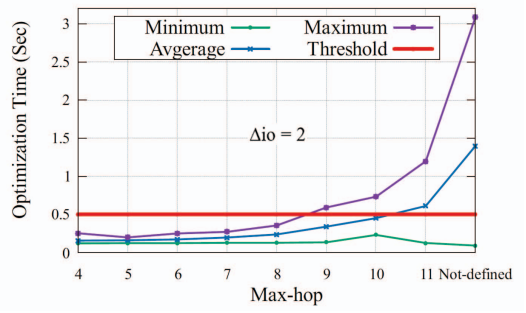


Fig. 8: Computation time in 4-k fat-tree,$\Delta_{io} = 2$

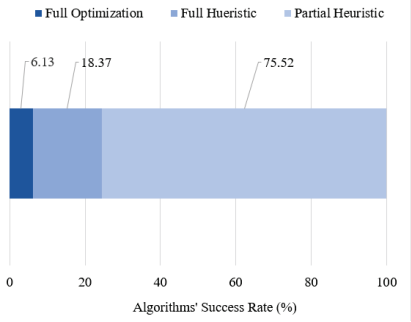
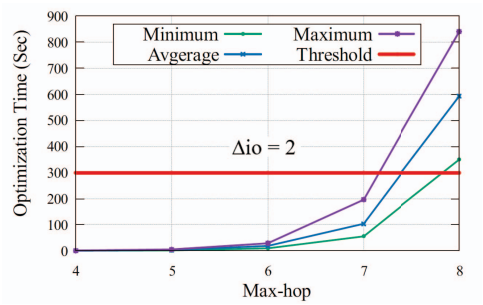


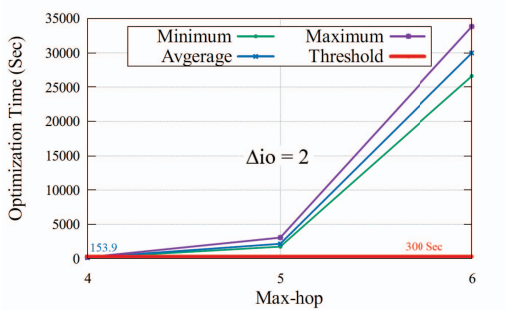
Fig. 9: Success rate comparison of optimization and heuristic baseline algorithm in 4-k port

of monitoring workloads processes. The choice of max-hop count may need to be reevaluated based on different Service Level Agreements (SLAs) and network requirements.

In our evaluations of over 100 iterations, we found that heuristic algorithms were able to offload all overloaded net-

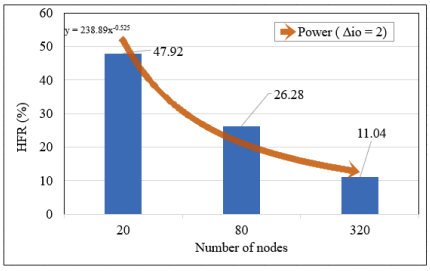


(a) 8-k ports fat-tree network

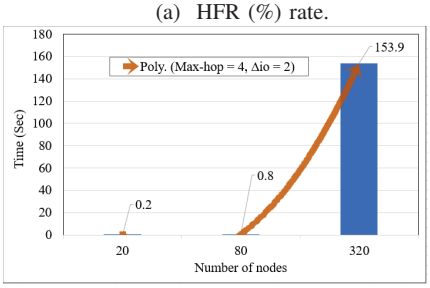


(b) 16-k ports fat-tree network

Fig. 10: DUST ILP optimization computation time in large-scale network equivalent of (a) 8-k and (b) 16-k ports fat-tree.



(a) HFR (%) rate.



(b) Average optimization time (sec).

Fig. 11: scalability evaluation (a) HFR rate of heuristic algorithm (b) average computation time of optimization algorithm.

work nodes in 18.37% of iterations where destination nodes were within one-hop distance, Figure 9. In 6.13% of iterations, heuristic algorithms could not offload any overloaded nodes, but optimizations were successful. The remaining 75.5% of experiments showed that heuristic algorithms partially offloaded some nodes, with the remaining offloaded by optimization. This demonstrates the trade-off between optimization and heuristic algorithm deployment, especially considering the higher cost of optimization in the customer field.

• **Large-Scale Network Evaluation:** In this section, we conducted a performance evaluation of the DUST optimization ILP offloading algorithm in large-scale networks and compared the results with those obtained in small-scale networks. The evaluation was based on a threshold response time value of 300 seconds, and the recommended max-hop values were determined as 7 for 8-k ports (Figure 10a), and 4 for 16-k ports (Figure 10b). By increasing the max-hop value from 4 to 5 in a 16-k ports network comprising 320 nodes resulted in a tenfold

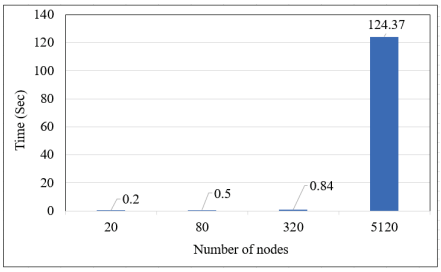


Fig. 12: Scalability evaluation of heuristic algorithm

increase in the average computation time. In conclusion, the number of hops is a significantly decisive and cost-effective optimization parameter in large-scale networks.

In Figure 11, a scalability comparison is presented between the optimization computation time and the HFR rate of a heuristic algorithm. Figure 11a shows that the HFR rate decreases from 47.92% to 11.04% as the network scale increases. Particularly in our evaluated testbed, this decrease can be estimated with a negative power function of ∼(-0.5). On the other hand, the average optimization time increases from 0.2 seconds to over 153 seconds, Figure 11b. This indicates that as the network scale grows, the heuristic algorithm can gain an advantage over optimization one by reducing the HFR rate.

Based on the metrics and findings presented in this paper, we suggest dividing large-scale networks into zones containing a maximum of 80 nodes. This approach has an acceptable optimization cost of 0.8 seconds for a max-hop value of 7 nodes for destination nodes. For larger networks, as depicted in Figure 12, the heuristic algorithm performs significantly better than the optimization algorithm, with an execution time of 124 seconds observed in a network with 5120 nodes.

VI. Conclusion and Future Work

The paper proposes a system-level solution for conserving resources in network switches by offloading telemetry services and deploying compute resources across the network to overcome the issue of resource-intensive in-device analytic engines. The suggested solution actively identifies heavily utilized nodes and redistributes monitoring resources to an optimal device, guided by controllable routes defined through

optimization models. Our evaluation of DUST's practical applications encompassed two distinct testbeds, wherein we deployed a genuine industry-scale switch prototype and an optimization simulator. The experimental results demonstrate the efficacy of the proposed approaches in terms of response time cost, scalability, and diverse use cases. Our solution enables remote offloading to overcome in-device supportability issues for network devices with lower capacities. It can be easily integrated into large-scale and cloud-based data centers and network providers e.g. AFC [38], AMD PSM [39], and our system has utility in diverse use cases and network services beyond service telemetry.

ACKNOWLEDGEMENT

This work is supported in part by the funds through the *Child Family Endowed Professorship*.

REFERENCES

[1] Vitalii Demianiuk, Sergey Gorinsky, Sergey Nikolenko, and Kirill Kogan. Robust distributed monitoring of traffic flows. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–11, 2019.

[2] Xiaohe Hu, Yang Xiang, Yifan Li, Buyi Qiu, Kai Wang, and Jun Li. Trident: Efficient and practical software network monitoring. *Tsinghua Science and Technology*, 26(4):452–463, 2021.

[3] Bruno Astuto A. Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys Tutorials*, 16(3):1617–1634, 2014.

[4] Roberto Bifulco and Gábor Rétvári. A survey on the programmable data plane: Abstractions, architectures, and open problems. In *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–7, 2018.

[5] Lizhuang Tan, Wei Su, Wei Zhang, Jianhui Lv, Zhenyi Zhang, Jingying Miao, Xiaoxi Liu, and Na Li. In-band network telemetry: A survey. *Computer Networks*, 186:107763, 2021.

[6] Anu Mercian, Puneet Sharma, Renato Aguiar, Chinlin Chen, and David Pinheiro. Udaan: Embedding user-defined analytics applications in network devices. In *Proceedings of the 2019 Workshop on Network Meets AI & ML*, pages 70–75, 2019.

[7] Mowei Wang, Yong Cui, Xin Wang, Shihan Xiao, and Junchen Jiang. Machine learning for networking: Workflow, advances and opportunities. *IEEE Network*, 32(2):92–99, 2018.

[8] Odej Kao Anton Gulenko, Marcel Wallschl¨ager. A practical implementation of in-band network telemetry in open vswitch. In *Research Advances in Cloud Computing*, pages 1–4. 2018.

[9] Network telemetry framework, ietf document. Link . [Online; accessed 2024].

[10] Qun Huang, Haifeng Sun, Patrick P. C. Lee, Wei Bai, Feng Zhu, and Yungang Bao. Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 404–421, New York, NY, USA, 2020. Association for Computing Machinery.

[11] Distributed services switches. Link . [Online; accessed 2024].

[12] Aruba cx 8325 switch series. Link . [Online; accessed 2024].

[13] Diman Zad Tootaghaj, Anu Mercian, Vivek Adarsh, Mehrnaz Sharifian, and Puneet Sharma. Smartnics at edge for transient compute elasticity. In *Proceedings of the 3rd International Workshop on Distributed Machine Learning*, DistributedML '22, page 9–15, New York, NY, USA, 2022. Association for Computing Machinery.

[14] Sean Choi, Muhammad Shahbaz, Balaji Prabhakar, and Mendel Rosenblum. -nic: Interactive serverless compute on smartnics. In *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, SIGCOMM Posters and Demos '19, page 151–152, New York, NY, USA, 2019. Association for Computing Machinery.

[15] Ming G. Liu. ipipe : A framework for building distributed applications onmulticore soc smartnics. 2019.

[16] Nersc: Perlmutter. Link. [Online; accessed 2024].

[17] Jie Li, George Michelogiannakis, Brandon Cook, Dulanya Cooray, and Yong Chen. Analyzing resource utilization in an hpc system: A case study of nersc's perlmutter. In Abhinav Bhatele, Jeff Hammond, Marc Baboulin, and Carola Kruse, editors, *High Performance Computing*, pages 297–316, Cham, 2023. Springer Nature Switzerland.

[18] Vitalii Demianiuk, Sergey Gorinsky, Sergey I Nikolenko, and Kirill Kogan. Robust distributed monitoring of traffic flows. *IEEE/ACM Transactions on Networking*, 29(1):275–288, 2020.

[19] L. Li, M. Thottan, B. Yao, and S. Paul. Distributed network monitoring with bounded link utilization in ip networks. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, volume 2, pages 1189–1198 vol.2, 2003.

[20] Basma Mostafa Hassan, Miklos Molnar, Mohamed Saleh, Abderrahim Benslimane, and Sally Kassem. Optimal proactive monitor placement & scheduling for IoT networks. *Journal of the Operational Research Society*, 73(11):2431–2450, 2022.

[21] Lorenzo Valerio, Marco Conti, and Andrea Passarella. Energy efficient distributed analytics at the edge of the network for iot environments. *Pervasive and Mobile Computing*, 51:27–42, 2018.

[22] Rustem Dautov, Salvatore Distefano, and Rajkumaar Buyya. Hierarchical data fusion for smart healthcare. *Journal of Big Data*, 6, 02 2019.

[23] Vitalii Demianiuk, Sergey Gorinsky, and Kirill Kogan. Telenoise: A network-noise module for in-band real-time telemetry. In *2021 IFIP Networking Conference (IFIP Networking)*, pages 1–9, 2021.

[24] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. Pint: Probabilistic in-band network telemetry. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 662–680, New York, NY, USA, 2020. Association for Computing Machinery.

[25] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. Hpcc: High precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 44–58, New York, NY, USA, 2019. Association for Computing Machinery.

[26] Smartnics vs dpus. Link . [Online; accessed 2024].

[27] Zhen Ni, Guyue Liu, Dennis Afanasev, Timothy Wood, and Jinho Hwang. Advancing network function virtualization platforms with programmable NICs. In *2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 1–6. IEEE, 2019.

[28] Aws nitro system. Link . [Online; accessed 2024].

[29] Google cloud and intel's new ipu. Link . [Online; accessed 2024].

[30] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure accelerated networking: Smartnics in the public cloud. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 51–66, 2018.

[31] Rest protocol. //https://en.wikipedia.org/wiki/Representational_state_transfer//. [Online; accessed 2024].

[32] Xingwei Wang, Hong Zhao, and Jiakeng Zhu. Grpc: A communication cooperation mechanism in distributed systems. *ACM SIGOPS Operating Systems Review*, 27(3):75–86, 1993.

[33] Cicso cpu usage critical-level thresholds. Link . [Online; accessed 2024].

[34] Data collection intervals of vsphere monitoring and performance. Link . [Online; accessed 2024].

[35] Pan-os device telemetry collection and transmission intervals. Link . [Online; accessed 2024].

[36] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, page 63–74, New York, NY, USA, 2008. Association for Computing Machinery.

[37] Gurobi optimization. Link . [Online; accessed 2024].

[38] Aruba fabric composer solution overview. Link . [Online; accessed 2024].

[39] Amd pensando policy and services manager. Link . [Online; accessed 2024].