iMPROVE: information-centric Multi-Parameter Routing Optimization Via Evaluation

***Abstract*—Secure and reliable communication is needed to support critical applications that have stringent requirements, such as low-latency and high-bandwidth data transmission. In this paper, we propose *iMPROVE*, a novel architecture for reliable and resilient communication. In *iMPROVE* each node autonomously evaluates its own performance metrics and scores available links, enabling packets to be forwarded over the highest-scoring paths according to their priority. This decentralized approach supports rapid, adaptive local decision-making. To assess *iMPROVE* performance, we evaluated the performance of the system against the state-of-the-art. Our results show that *iMPROVE* significantly enhances the resiliency of high-priority traffic compared to state-of-the-art, substantially reducing packet loss in congested network environments and thereby improving overall network performance.**

***Index Terms*—Flow Control, link failure, link scoring, QoS, traffic prioritization.**

I. Introduction

RESILIENCY is key when striving to achieve a reliable network architecture to support critical applications. Current deployed network architectures use an IP-based network architecture, where source and destination IP addresses are used for packet delivery. However, the IP-based architecture cannot fully and adequately address the unique requirements of providing QoS to critical applications. A novel communication architecture called Named Data Networking (NDN) has emerged as one of the most common realizations of the Information-Centric Networking (ICN) paradigm [1]. The fundamental idea of NDN is to replace the existing host-centric communication model with a data-centric paradigm, in which unique names are used to retrieve data rather than the IP address of the server storing it.

To reduce communication latency and overhead as well as enhance availability, NDN uses pervasive data caching, which enables any network entity to satisfy data requests. NDN's unique features allow flexible and resilient data forwarding, in-network computation, and built-in data integrity and authenticity. NDN-based solutions to provide improved scalability, QoS, low latency, and improved reliability have been proposed [2]. However, these solutions do not have systems in place to maintain QoS for high-priority traffic in a congested network.

To address these shortcomings, we propose *iMPROVE*, a communication architecture with systems in place to maintain QoS for high-priority traffic within a congested network. It is capable of assessing the impact of various networking conditions on critical applications. The communication architecture of *iMPROVE* is built on top of iCAAP [3], which is an NDN-based communication architecture. In particular, we extended iCAAP by devising a smart forwarding strategy (inspired by DICE [4]) to achieve traffic prioritization and QoS-aware forwarding in addressing the stringent requirements of different traffic flows.

Contribution: Our novel contributions can be summarized as follows: **(i)** We outline the communication architecture of *iMPROVE*, a novel NDN-based communication architecture that utilizes a smart forwarding strategy to promote traffic prioritization and QoS-aware packet forwarding. **(ii)** A systematic sensitivity analysis over a comprehensive discrete set of weight combinations, enabling the identification of the most influential network parameters for minimizing loss and latency. **(iii)** An evaluation of *iMPROVE*'s network architecture against state-of-the-art.

II. Related Work

A. Smart forwarding

Kamboj et al. [5] proposed a QoS-aware multipath routing scheme for SDN networks consisting of three phases: splitting incoming flows, using a cost-optimized heuristic to route subflows, and reordering packets at the destination. This method meets high-bandwidth and low-latency requirements by balancing load across disjoint paths while ensuring service guarantees.

Panwar et al. in [4] introduced Dynamic Multi-RAT Selection in the ICN-enabled Wireless Edge (DICE), a dynamic forwarding strategy for Information-Centric Networking (ICN). This strategy allows mobile devices to select the best subset of Radio Access Technologies (RATs) based on real-time network conditions such as link quality and congestion. DICE dynamically selects the minimum set of interfaces required to transfer packets concurrently. It optimizes network resource utilization by reducing the number of interfaces needed for data

delivery. DICE lacks key performance metrics such as bandwidth, jitter, and packet count, which are crucial for critical applications. Incorporating these metrics would enhance DICE's ability to optimize network performance and improve throughput and latency stability for high-throughput, latency-sensitive applications.

In their work on the MOCell algorithm [6], Gonzalez-Trejo et al. introduced a multi-objective optimization approach that considers various Quality of Service (QoS) metrics such as bandwidth, delay, packet loss, and hop count to identify the best path for data transmission. The algorithm uses weights to balance these metrics and employs evolutionary techniques to refine potential paths, ultimately selecting the path that maximizes overall network performance. However, this approach has limitations in networks where nodes handle packets with different priorities. Furthermore, the authors do not address the impact of recently transmitted packets on a link or the queue fill ratio, which can significantly affect performance in high-traffic conditions.

B. Background traffic and link failure

Kothandaraman et al. [7] propose a decentralized dynamic alternate-routing algorithm for IoT networks that uses stochastic node-meeting models to anticipate and reroute around link failures. Their evaluation shows that this approach significantly improves end-to-end delivery reliability, reduces control overhead, and saves node energy compared to traditional routing methods.

III. Smart Forwarding Communication Architecture

A. NDN Overview

NDN is a pull-based communication architecture that enables a consumer to request data by name from a provider in a secure and resource-efficient manner. In NDN, each piece of data, *i.e.,* , data chunk, is assigned a unique name, which follows a hierarchical human-readable convention similar to URL addresses. This enables the consumers to request the data chunks by name and the communication network entities to cache data chunks. In NDN, the network entities forward the consumer's request to the data provider, which in turn, returns the requested chunk of data to the consumer using the reverse path. As the data chunks travel back to the consumer, the network entities decide whether to cache the data chunk or not based on factors such as data popularity. NDN's built-in security mandates that data providers sign their data upon creation to promote source authenticity and data integrity.

In NDN, each network entity is equipped with a Pending Interest Table (PIT), Forwarding Information Base (FIB), and a Content Store (CS) [8]. The content store acts as a temporary cache to store popular data. Similar to the existing routing table, FIB helps forward the request to the provider. NDN's stateful forwarding plane uses PIT to keep track of in-flight requests and further enable request aggregation. NDN also features a strategy layer, which enables flexible and fine-grain forwarding decision-making, such as least-cost path, multicast, broadcast, or a customized forwarding strategy for meeting the unique requirements of a given application.

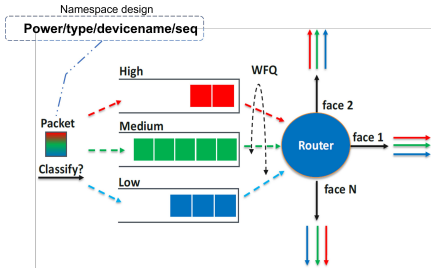


Fig. 1: *iMPROVE* Communication Architecture.

B. iCAAP Architecture

In designing *iMPROVE*, we adopt iCAAP [3], our previous NDN-based architecture for QoS-aware communication, and extend it with a smart forwarding strategy inspired by [4]. As shown in Fig. 1, iCAAP has two major components: traffic prioritization and QoS-aware traffic management via token bucket. In what follows, we explain these components in more detail.

1) Traffic Prioritization: Considering the various types of applications and their requirements, with *iMPROVE*, we classify the network traffic into three different priority classes to promote QoS-aware traffic management. The three classes include *Type I* as high priority, *Type II* as medium priority, and *Type III* as low priority. To encode these priority classes into network traffic, each packet includes its priority class in the "`type`" component of the requested data name. To meet the QoS requirements of various traffic classes, we devised three distinct priority queues for each interface of every given node–one per priority class–which facilitates traffic prioritization by assigning similar priority traffic to the same queue. Upon arrival at a node, a packet will be pushed into the appropriate queues and then dequeued when resources are available for packet transmission. The order of dequeuing packets from all queues is determined by the Weighted Fair Queuing (WFQ) algorithm.

2) Token Bucket: To shape network traffic and control the communication rate of the traffic classes, we used a token bucket algorithm, where each priority queue is associated with a token bucket (*i.e.,* , a limited number

Algorithm 1: *iMPROVE* Forwarding Strategy

```
1  pr ← getSuccessProbabilityRequirement(p)
2  L = getInterfaces(p)
   // Remove faces that do not meet the spr
3  FL = filterInterfaces(L, spr)
4  if (notEmpty(FL)) then
5      outFace = chooseBestInterface(FL)
6      forward(p, outFace)
7  else
8      i = 0; ps = 0
9      sortInterfacesBySuccessProbability(L)
10     while ps < pr and i < length(L) do
11         forward(p, L[i])
12         ps += getSuccessProbability(L[i])
13         i++
```

of tokens). Thus, a packet can be dequeued from the corresponding queue only if a sufficient number of tokens are available for the given queue; one token will be used to dequeue one packet. In the design of *iMPROVE*, the token generation rate for each priority class is adjusted based on the priority requirements. For our use case, we ran a large number of Monte Carlo simulations to identify the most pertinent token generation rate for each priority class. We then set the token generation rates accordingly in our isolated network simulations.

C. iMPROVE Architecture

In iCAAP [3], we used a customized QoS-aware forwarding strategy to handle prioritization and a queuing mechanism where Type I and Type II traffic classes used multicast routing. In contrast, Type III used the best-route (unicast) mechanism. In *iMPROVE*, we propose a novel smart forwarding strategy for reliable packet forwarding. To improve reliability, the proposed strategy forwards packets over multiple interfaces in specific situations. More specifically, our strategy uses a subset of interfaces to meet the expected data delivery rates of different traffic classes based on network performance metrics, such as network congestion and statistical information of available interfaces. Note that this design contrasts with iCAAP's static approach, where the number of selected interfaces is always the same for a given traffic class.

Algorithm 1 describes the *iMPROVE* forwarding strategy. Upon receiving the packet (p), the network entity node (*e.g.,* router u) extracts the priority class (pClass) of the packet from the type component of the packet name. Each type has a unique success probability requirement (spr), intuitively chosen as 100%, 80%, and 0% for Types I, II, and III, respectively. Router u identifies the priority class of p and sets p's probability requirement (pr) based on the class's spr (Lines 1).

Each viable interface for the packet's next hop is then assembled into a list. This list of available interfaces is filtered to remove any options that do not meet the pr(Lines 2-3). If at least one interface is present in the resulting list, then the remaining interfaces are assigned a score ($\mathsf{l_s}$). This score is calculated based on six metrics: max bandwidth, success probability, queue fill rate, number of packets sent, jitter, and throughput. The packet is then forwarded on the interface with the highest resulting score (Lines 4-6).

If no interfaces meet the pr, then router u will sort the interfaces based on the probability of success. Router u will then go down the sorted list $\mathcal{L}$ and enqueue p on the interfaces, adding their success probability to the total probability score ps, until ps is greater than or equal to the pr for p, or we reach the end of $\mathcal{L}$ (Lines 7-13).

Algorithm 2: Interface Metric Update Procedure

Input: Packet p with named prefix, interface state, and packet history
Output: Updated metrics: Loss rate, jitter, number of packets sent, throughput, queue fill ratio

```
1  α ← 0.83 ;   // EWMA smoothing factor
2  TimeoutThreshold ← 5 seconds;
3  if packet p is forwarded on an interface then
4      Store {prefix, timestamp, size} in packet history;
5      Increment PacketsSent;
6  if packet with prefix is successfully satisfied then
7      Retrieve sentTimestamp, size from packet history using prefix;
8      latency ← currentTime − sentTimestamp;
9      Update throughput history with {latency, size};
10     Update jitter based on latency variation;
11     LossRate ← α × LossRate;
12     Remove entry from packet history;
13 else if packet times out (currentTime − sentTimestamp > TimeoutThreshold) then
14     LossRate ← α × LossRate + (1 − α);
15     Remove entry from packet history;
16 Update queue fill ratio based on current interface queue length;
```

Each interface maintains a history of packets for every prefix entry, which is updated based on packet events as shown in Algorithm 2. When a packet is forwarded, the

interface records the prefix, timestamp, and size in the packet history and increments the total number of packets sent. If the packet is subsequently satisfied, the corresponding timestamp and size are retrieved to compute the packet's latency and update the throughput history. Jitter is calculated as the variation in packet latency, measured as the absolute difference in latency between successive packets. The loss rate is updated using an exponentially weighted moving average (EWMA) [9] with a smoothing factor $\alpha = 0.83$. If a packet is satisfied, the loss rate is updated as *LossRate* $\leftarrow \alpha \times$ *LossRate*, and if a packet times out (i.e., not delivered within 5 seconds), it is updated as *LossRate* $\leftarrow \alpha \times$ *LossRate* $+ (1 - \alpha)$.

The raw values from the six metrics are typically calculated based on the packet history. The exceptions are the max bandwidth and queue fill rate, the first of which is fixed and immutable, while the other varies based on how full the queue is at the time of scoring. The success probability is updated as packets are either satisfied or timed out. In contrast, the packets sent is a running history of the total number of packets sent on an interface in recent history. At each scoring interval, throughput and jitter are calculated using a five-second rolling history. The total size in the running history is totaled to calculate the throughput, while the jitter is calculated using the latency variance recorded in said history.

$$\begin{aligned}\text{Score} = {} & (1 - \text{Loss Fraction}) \times w_1 + \text{Bandwidth} \times w_2 \\ & + (1 - \text{Queue Fill Ratio}) \times w_3 - \text{Packet Sent} \times w_4 \\ & + (1 - \text{Jitter}) \times w_5 + \text{Throughput} \times w_6\end{aligned} \tag{1}$$

All metrics are collected per interface at the time of scoring and normalized to the range $[0, 1]$. The overall score for each interface is then computed as a weighted sum, as shown in Equation 1, where each weight is constrained to lie between 0 and 1 and the sum of all weights equals one. The process of determining suitable weights requires thorough and systematic exploration, which is detailed in the following sections.

IV. Performance Evaluation

In this section, we discuss our simulation setups and our evaluation. We first conducted a sensitivity analysis to find the optimal weight parameters for *iMPROVE*. We then defined four simulation scenarios, which varied with respect to background traffic or link failure. We analyze the performance of each architecture in all scenarios, which include baseline NDN, iCAAP, DICE, and *iMPROVE* with two combinations of weights (Best and Average combination of weights). We analyzed the latency, loss rate, and overhead performance of each of the architectures. We ran this on an experiment using ns-3 on a computer running Ubuntu 20.04 and Intel(R) Xeon(R) W-2245 CPU @ 3.90GHz, 128 GB of RAM.

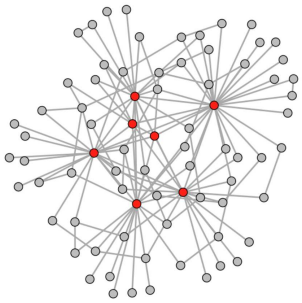


Fig. 2: 77-Node topology

This experiment ran on a network composed of an undirected graph $G(V, E)$ with $|V| = 77$ and $|E| = 135$ as shown in Fig 2.

$$H \subset V, \quad |H| = 7, \quad R = V - H, \quad |R| = 70.$$

The induced subgraph $G[H]$ is the complete graph K_7, which forms a highly reliable central backbone. Regular nodes in R are embedded in $\mathbb{R}^2$ to simulate geographic dispersion and approximately $0.8\,|R|$ nodes connect exclusively to their nearest hub, while the remaining $0.2\,|R|$ nodes each connect to multiple hubs to ensure redundancy.

A. Optimal Weights

To determine the optimal weight parameters, we conducted a sensitivity analysis on a smaller network using four different topologies, each consisting of 18 nodes. These topologies were generated using the Barabasi-Albert algorithm, the Erdos-Renyi algorithm, and two variations based on the Watts-Strogatz model. In the first Watts-Strogatz variation, nodes were connected to four neighbors, while in the second, nodes were connected to either 4, 6, or 8 neighbors.

To identify the optimal set of parameter weights, we performed a sensitivity analysis for each topology by systematically exploring all feasible combina-

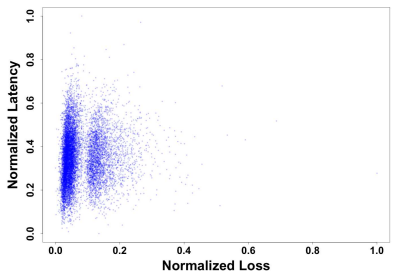
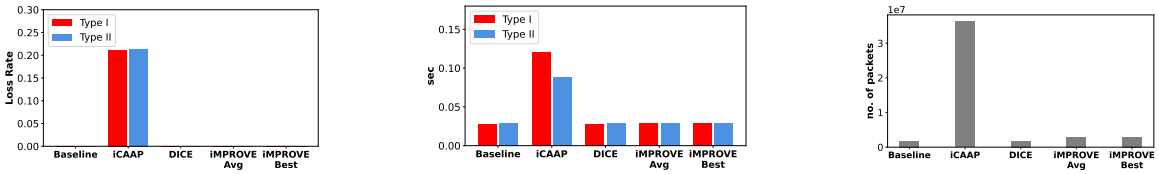
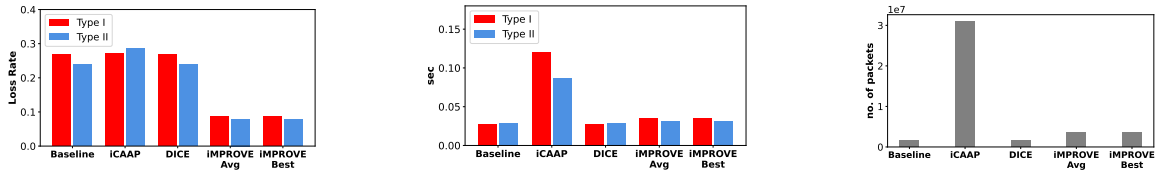


Fig. 3: Parteo Frontier - Type I traffic.



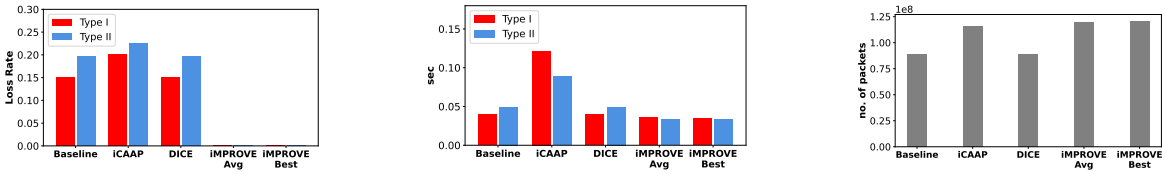
(a) Loss (b) Latency (c) Packet overhead

Fig. 4: Loss, Latency, and Packet Overhead in NBT and NLF scenario.



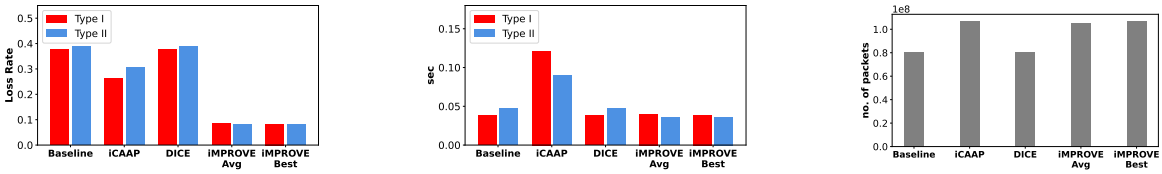
(a) Loss (b) Latency (c) Packet overhead

Fig. 5: Loss, Latency, and Packet Overhead in NBT and LF scenario.



(a) Loss (b) Latency (c) Packet overhead

Fig. 6: Loss, Latency, and Packet Overhead in BT and NLF scenario.



(a) Loss (b) Latency (c) Packet overhead

Fig. 7: Loss, Latency, and Packet Overhead in BT and LF scenario.

tions of six weights, $w_1, w_2, \ldots, w_6$, each representing weight for one of the following network metrics: maximum bandwidth, success probability, queue fill rate, number of packets sent, jitter, and throughput. Each weight was restricted to values in the discrete set $\{0.05, 0.10, \ldots, 1.00\}$, and all combinations satisfying the constraint $\sum_{i=1}^{6} w_i = 1$, as shown in Equation (2), were considered. This exhaustive evaluation enabled a comprehensive sensitivity analysis, providing insight into how varying the relative importance of each parameter affects overall network performance.

$$w_i \in \{0.05, 0.10, \ldots, 1.00\}, \forall i \in \{1, \ldots, 6\}, \sum_{i=1}^{6} w_i = 1 \tag{2}$$

Each scenario involved running a 100-second simulation, during which we calculated the loss and latency metrics for three different types of traffic. Average loss and latency values were computed for each traffic type across all four topologies for each weight combination. This process provided us with average loss and latency metrics for all three traffic types across every combination of weights. To evaluate these weight combinations, we assigned a score to each set based on the following weighted sum:

$$\begin{aligned} \text{Score} = {} & 0.3 \times \text{loss}_{\text{Type I}} + 0.3 \times \text{latency}_{\text{Type I}} \\ & + 0.24 \times \text{loss}_{\text{Type II}} + 0.06 \times \text{latency}_{\text{Type II}} \\ & + 0.10 \times \text{loss}_{\text{Type III}} + 0 \times \text{latency}_{\text{Type III}}. \end{aligned} \tag{3}$$

As per Equation 3, loss and latency metrics

were weighted most heavily for Traffic Type I, with Traffic Type II receiving the next highest weights. From the full set of candidate weight vectors, the twenty highest-scoring combinations were selected and their arithmetic mean computed, yielding the average weight vector $\mathbf{w}_{\text{Avg}} = [0.1525, 0.1900, 0.1600, 0.2175, 0.1475, 0.1325]$. Type I traffic was designated as high-priority, and the average loss and latency for Type I flows were calculated across all four network scenarios for each candidate weight combination. To illustrate the inherent trade-off between loss and latency, a Pareto frontier was constructed with normalized values of loss and latency as seen in Fig. 3, where each point represents a non-dominated weight vector for which neither loss nor latency can be simultaneously reduced. The datapoint located at the lower left corner of the Pareto frontier corresponds to the weight vector that achieves the lowest values for both loss and latency. Accordingly, the globally optimal weight vector for Type I traffic was identified as $\mathbf{w}_{\text{Best}} = [0.25, 0.10, 0.05, 0.45, 0.10, 0.05]$.

B. iMPROVE *Performance Evaluation*

We conducted simulations on a 77-node network (Fig. 2) over a 151-second interval. The study considered four scenarios: No Background Traffic and No Link Failure (NBT-NLF), No Background Traffic with Link Failure (NBT-LF), Background Traffic with No Link Failure (BT-NLF), and Background Traffic with Link Failure (BT-LF). Background traffic corresponds to elevated Type III traffic, where 44 nodes generate approximately 2500 Type III packets per second. Link failures were randomly induced by disabling 26 out of 135 edges during the intervals 25–50s, 75–100s, and 125–150s. Average loss, latency, and packet overhead were calculated over ten independent runs for each scenario.

Fig. 4 presents the results for the NBT-NLF scenario. The iCAAP approach performs poorly, even in the absence of background traffic and link failures, primarily due to token bucket overloading. The other three methods exhibit comparable performance under these conditions. Notably, iCAAP incurs an overhead approximately 15 times greater than the baseline method, while *iMPROVE* exhibits roughly twice the overhead. Fig. 5 illustrates the results for the NBT-LF scenario. iMPROVE achieves significantly lower packet loss than the other strategies. Although latency increases slightly, this can be attributed to the utilization of additional routes and longer paths. It is also observed that *iMPROVE* incurs higher overhead in NBT-LF compared to NBT-NLF due to increased loss rates necessitating additional packet transmissions for successful delivery. The performance of DICE remains similar to that of the baseline approach across both scenarios.

Fig. 6 shows the results for the BT-NLF scenario. *iMPROVE* demonstrates substantial improvements over the other approaches and achieves the lowest latency by effectively distributing background traffic and preventing link congestion. In this scenario, iMPROVE incurs slightly higher overhead than iCAAP, primarily due to increased packet disruption from background traffic. To ensure successful delivery, iMPROVE transmits packets across multiple paths, which results in more packets being counted in the overhead metric. In contrast, iCAAP reports lower overhead largely because approximately $\sim$20% of its traffic is lost and not included in the measurement. Fig. 7 presents the results for the BT-LF scenario. Consistent with previous cases, iMPROVE demonstrates a marked improvement over the other methods. This can be attributed to its ability to adapt rapidly to link failures, in contrast to the other strategies. While loss rates are elevated in this scenario due to remaining links being heavily congested with background traffic, iMPROVE experiences only a slight increase in loss compared to NBT-LF. The latency for *iMPROVE* remains favorable, and follows the same trend observed in BT-NLF. Overhead in this scenario is comparable to that of BT-NLF, while iMPROVE exhibits slightly lower overhead than iCAAP.

V. Conclusion

The proposed iMPROVE architecture introduces a network-aware forwarding strategy that makes routing decisions based on current network conditions. By leveraging NDN for traffic prioritization and employing a token bucket mechanism for flow regulation, iMPROVE achieved lower loss rates, latency, and packet overhead than traditional approaches. The method maintained strong performance even with heavy background traffic and during link failures, ensuring reliable service for high-priority packets while making efficient use of network resources. For future work, we aim to investigate adaptive parameter tuning so that iMPROVE can better respond to changing traffic patterns. We also plan to integrate iMPROVE with additional congestion control techniques to further improve efficiency under high network loads.

Acknowledgment

This research was partially funded by the US Department of Defense grant #W911QX23D0009, US National Science Foundation under grants CNS-2148358, OIA-2417062, #2148358 and #1914635, and the US Department of Energy grant #DE-SC0023392. Any opinions, findings, conclusions, or recommendations expressed in

this material are those of the authors and do not necessarily reflect the views of the US federal agencies.

REFERENCES

[1] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang *et al.*, "Named Data Networking," *ACM SIGCOMM Computer Commun. Rev.*, vol. 44, no. 3, pp. 66–73, 2014.

[2] S. K. Ramani and A. Afanasyev, "On Using NDN to Vertically Secure Smart Power Distribution," in *2020 IEEE CyberPELS (CyberPELS)*, 2020, pp. 1–6.

[3] A. K. James, G. Torres, S. Shrestha, R. Tourani, and S. Misra, "iCAAP: information-Centric network Architecture for Application-specific Prioritization in Smart Grid," in *2021 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, 2021, pp. 1–5.

[4] G. Panwar, R. Tourani, T. Mick, A. Mtibaa, and S. Misra, "DICE: Dynamic multi-RAT selection in the ICN-enabled wireless edge," *ACM SIGCOMM Computer Communication Review*, vol. 47, pp. 67–72, 10 2017.

[5] P. Kamboj, S. Pal, S. Bera, and S. Misra, "QoS-Aware Multipath Routing in Software-Defined Networks," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 2, pp. 723–732, 2023.

[6] J. E. Gonzalez-Trejo, R. Rivera-Rodriguez, A. Tchernykh, J. E. Lozano-Rizk, S. Villarreal-Reyes, A. Galaviz-Mosqueda, and J. L. Gonzalez Compean, "A Novel Strategy for Computing Routing Paths for Software-Defined Networks Based on MOCell Optimization," *Applied Sciences*, vol. 12, no. 22, p. 11590, 2022.

[7] D. Kothandaraman, M. Manickam, A. Balasundaram, D. Pradeep, A. Arulmurugan, A. K. Sivaraman, and R. Balakrishna, "Decentralized link failure prevention routing (DLFPR) algorithm for efficient internet of things," *Intelligent Automation and Soft Computing*, vol. 34, no. 1, pp. 655–666, 2022.

[8] A. Afanasyev, J. Burke, T. Refaei, L. Wang, B. Zhang, and L. Zhang, "A Brief Introduction to Named Data Networking," in *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, 2018, pp. 1–6.

[9] G. Torres, S. Shrestha, and S. Misra, "iCAD: information-Centric network Architecture for DDoS Protection in the Smart Grid," in *2022 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2022, pp. 154–159.