In-Network Split Inference with Named Data Networking under Lossy Edge Connectivity

Marica Amadeo*, Claudia Campolo†, Antonella Molinaro†, Giuseppe Ruggeri†
* Department of Engineering, Università degli Studi di Messina, Italy
† DIIES Department, Università Mediterranea di Reggio Calabria, Italy
Emails: {marica.amadeo@unime.it}, {claudia.campolo, antonella.molinaro, giuseppe.ruggeri}@unirc.it

***Abstract*—In-Network Computing (INC) is emerging as a key enabler of Sixth-Generation (6G) systems, allowing programmable network nodes to provide not only connectivity, but also storage and processing across the cloud-to-edge continuum. Machine learning (ML) tasks, particularly Deep Neural Network (DNN) inference, stand to benefit significantly from this shift. Under the Split Inference (SI) paradigm, different layers of a DNN can be distributed across multiple in-network nodes that cooperate with the end-device requesting inference. In this work, we explore the potential of Named Data Networking (NDN) as an enabler for in-network SI. We demonstrate how NDN's native features, such as in-network caching and routing-by name, can reduce inference delays and improve robustness under lossy edge connectivity, compared to traditional host-centric networking. Simulation results validate the effectiveness of NDN-based in-network SI, highlighting its potential to enable resilient and efficient ML services in future 6G environments.**

***Index Terms*—Named Data Networking, Split Inference, In-network computing**

I. Introduction

The emerging paradigm of In-Network Computing (INC) takes advantage of programmable network elements not only for connectivity but also for computation. By enabling software execution directly within networking devices, particularly in the data plane, INC allows packets to be processed at line speed, providing low latency access to computing resources [1], [2], and reducing the load on purpose-built edge or cloud infrastructures. Furthermore, in-network computing (INC) aligns closely with the requirements of Sixth-Generation (6G) networks, offering greater flexibility and responsiveness compared to traditional edge computing models.

Although not yet largely investigated, distributed inference is one of the many computing tasks that can benefit from INC [3], [4]. For instance, Digital Twin (DT) applications, such as real-time monitoring of vehicles, industrial equipment, or smart city infrastructures, require continuous inference over large volumes of sensed data, which can be efficiently supported by distributing the execution across in-network nodes. Similarly, immersive EXtended Reality (XR) services require continuous inference on video streams, which should be performed in edge nodes with ultra-low latency, to ensure a seamless user experience. The Split Inference (SI) paradigm addresses these needs by partitioning Deep Neural Network (DNN) models into multiple blocks distributed across different devices, typically a mobile device and an edge or cloud server. The mobile device locally executes the initial layers of the DNN model and sends the intermediate output to a more capable node to complete the remaining computation [5]. This collaborative execution may reduce end-to-end inference latency compared to fully local processing on resource-constrained devices or full offloading to distant cloud servers.

The decision about *where* to split a DNN model is not easy, since different layers generate intermediate outputs of different sizes and entail different computing demands. Therefore, an effective split decision should take into account the capabilities (e.g., computing, memory, battery) of the involved nodes, as well as the conditions of the network links connecting them [6], [7].

While the traditional split typically involves only two nodes, this concept can be extended to multi-point splits involving different devices [8]. In particular, INC enables programmable network nodes along the data path to participate in DNN execution by processing selected layers [9], [10], [11]. However, distributing execution across multiple in-network nodes exacerbates challenges related to the practical SI implementation. Beyond identifying an optimal splitting policy, efficient orchestration of the distributed execution becomes essential.

To date, no dedicated communication protocol has been specifically designed to support the needs of INC-driven applications. However, service-centric communication architectures have emerged as highly promising in this context [1]. Unlike traditional host-centric models, such as the Internet protocol suite, these architectures decouple communication from fixed endpoints, enabling routing based on named data or services. This paradigm allows for dynamic task allocation, seamless data retrieval from any available in-network cache, and more resilient execution of distributed inference, particularly in heterogeneous and lossy network environments.

Information-Centric Networking (ICN) [12], and in particular its Named Data Networking (NDN) implementation, embodies the service-centric vision and holds strong potential to support INC [13]. Building on this premise, our previous work [14] introduced a preliminary solution that leverages NDN to orchestrate in-network SI tasks efficiently. By exploiting NDN's core features, such as name-based service provisioning, stateful forwarding and in-network caching, we demonstrated that NDN can enable dynamic, low-latency service placement across distributed nodes, aligning well with the requirements of SI workflows.

In this work, we take a step forward by providing the

978-3-903176-75-1 ©2025 IFIP

following key contributions.

- We present a deeper investigation into the use of NDN for SI, with particular attention to the role of in-network caching. In contrast to the legacy Least-Recently-Used (LRU) cache replacement policy, we present a customized caching mechanism tailored to SI workloads. This approach reduces storage overhead while ensuring robustness against packet losses.

- We develop and evaluate the proposed NDN-based solution using ndnSIM [15], a widely adopted simulation platform for NDN research. This allows us to faithfully capture packet forwarding and caching dynamics.

- We conduct simulations under realistic and lossy network conditions to assess the effectiveness of NDN-based in-network SI. Our evaluation focuses on the impact of packet losses and determines *if and to what extent NDN outperforms traditional host-centric solutions* in supporting distributed inference tasks.

The remainder of this paper is organized as follows. Section II scans the literature and discusses the main motivations of this work. The proposed NDN-based solution for in-network SI is presented in Section III. Section IV discusses the simulation setup and evaluates the performance of our approach. Finally, Section V concludes the paper.

II. Background and motivations

A. In-network computing and split inference

The recently introduced paradigm of INC presents promising opportunities to improve Machine Learning (ML) performance by enabling ML algorithms to run directly on programmable network devices [4]. For instance, in [9] the authors propose executing inference tasks on enhanced User Plane Functions (UPFs) deployed within the network infrastructure. Similarly, the work in [10] explores the use of INC for enabling split inference, while in [11] the authors investigate cooperative inference execution involving switches, end-devices and powerful servers.

Most existing studies focus either on architectural network enhancements or on the decision-making criteria for partitioning DNN models. However, to the best of our knowledge, little attention has been given to the design of communication protocols specifically tailored to support in-network SI operations.

In our previous work [14], we proposed leveraging NDN to address this gap. While that study offered preliminary insights, it did not fully explore the comparative advantages of NDN, particularly under challenging network conditions, relative to traditional host-centric communication protocols, which is instead the main focus of this work.

B. NDN in a nutshell

The NDN paradigm represents a fundamental shift from the traditional Transmission Control Protocol (TCP)/Internet Protocol (IP) architecture by allowing packets to name data objects rather than communication endpoints [12].

In NDN, consumers send *Interest* packets to request contents identified by hierarchical names. Intermediate nodes forward these Interests based on the name, directing them toward potential data provider(s). In response, *Data* packets are returned containing the requested content, also identified by the same name.

Each node maintains three key data structures: *(i)* a Content Store (CS) that temporarily caches received Data packets; *(ii)* a Pending Interest Table (PIT) that keeps track of Interests awaiting Data responses; and *(iii)* a Forwarding Information Base (FIB) that maps name prefixes to outgoing interface(s) for forwarding Interests. Interfaces may correspond to physical network links or application-level interfaces, in cases where the node itself runs an application capable of generating the requested data. Notably, Data packets can be retrieved not only from the original content producer but also from any intermediate node that has cached the content (or a portion of it) in its CS.

C. NDN for in-network computing

Deploying INC entails the management and orchestration of heterogeneous resource pools distributed across network nodes. These nodes often differ in their available computing, storage, and energy capabilities, which may vary over time and across locations.

The traditional host-centric communication model of today's Internet, designed for static, end-to-end interactions between fixed clients and servers, poorly fits the dynamic and distributed INC nature [13]. The TCP/IP protocol stack inherently binds communication to specific physical addresses and machines in a static manner, limiting flexibility and adaptability. In contrast, INC demands location-independent and on-demand allocation of computing tasks.

In this context, NDN, as a leading ICN architecture, offers key advantages and has been considered as a promising candidate to support INC at scale, as extensively discussed in [13]. Focusing on ML workloads, NDN has been identified as a key enabler for both distributed training and inference tasks [14], [16], [17], [18]. Studies such as [17] and [18] have demonstrated performance improvements over the TCP/IP architecture, particularly in the context of federated learning.

As theoretically discussed in [14], NDN offers several native advantages for managing SI. Most notably, its in-network caching capability allows data to be exchanged efficiently and reliably, even in the presence of channel errors or congestion-induced packet losses. By caching traversing packets, intermediate nodes can autonomously recover lost data without requiring huge end-to-end retransmissions, which are typically necessary in host-centric approaches. This mechanism provides a twofold benefit: it reduces inference latency in distributed environments and alleviates network load, thereby supporting more scalable and resilient ML inference.

In this work, we elaborate on the specific data exchange routines involved in SI, with a special focus on the data replacement policy in the CS, and we evaluate the extent to which NDN can enhance overall network efficiency, particularly in comparison to conventional host-centric communication models.

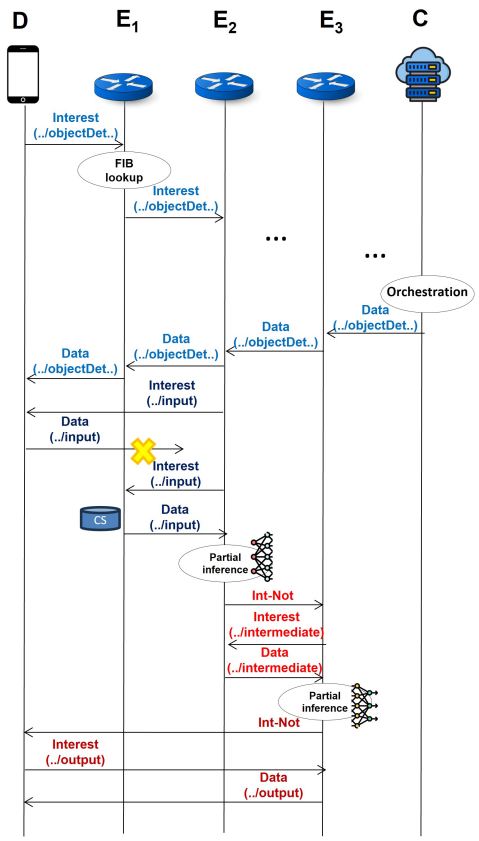

Fig. 1. Interest/Data exchange for NDN-based in-network SI. A mobile device, D, requests the execution of an inference task. Along the path to the cloud, in-network nodes process the request. Among these, only two nodes, E_2 and E_3, volunteer, and the model is split between them.

III. NDN-BASED IN-NETWORK SPLIT INFERENCE

The framework assessed in this work was initially introduced in [14]; it is briefly revisited here for completeness, along with the extensions proposed in this study.

A. Framework design

As illustrated in Fig. 1, supporting SI tasks through NDN involves a centralized orchestration decision and a distributed execution, based on a sequence of key steps, as detailed in the following.

Discovery. In the proposed design, clients wishing to get an inference output initiate the SI task by transmitting an augmented Interest packet that triggers the discovery of in-network nodes capable of participating in the task. This Interest carries a hierarchically structured name encoding: *(i)* the service type, *(ii)* the target ML model, *(iii)* an identifier for the input data, and *(iv)* application-specific constraints, such as the maximum tolerated latency. For instance, the name `/splitAI/objectDetection/-ResNet50/img.jpg/lat=0.1` requests an object detection service using the ResNet model on the input image `img.jpg`, with a latency constraint of 0.1s. Similarly, the name `/splitAI/emotionRecognition/-MobileNet/audio.wav/lat=0.2` refers to a speech-based emotion recognition service using the MobileNet model on the input `audio.wav`, to be accomplished within 0.2s.

As the Interest traverses the network, each intermediate network node looks up in its FIB to determine whether it can serve as an executor. A longest-prefix match involving both the service type and the ML model, with the matching outgoing FIB interface pointing to the local application layer, indicates that the node is eligible to execute (part of) the SI task. This case occurs when the requested ML model (or parts of it) has been previously cached to serve similar requests. Eligible nodes respond by advertising their availability, appending their capabilities (e.g., available compute resources and latency to the client), to the Interest. Otherwise, if no match is found, the Interest is simply forwarded towards the reference SI orchestrator, e.g., a cloud-based node.

Orchestration. The discovery process continues as the Interest packet propagates through intermediate nodes along the path until it reaches the SI orchestrator, e.g., the remote cloud. At this point, the orchestrator evaluates the candidate nodes and determines the optimal splitting. It then issues a Data packet containing the selected splitting strategy and the necessary execution instructions for the involved nodes.

Input data retrieval. The node instructed to execute the initial portion of the DNN model (if different from the client) sends an Interest towards the client to request the input data.

Inference execution and intermediate output data exchange. Once the input data is received, the first node in the computing chain starts executing its assigned portion of the DNN model. Upon completion, it issues an Interest Notification (*Int-Not*) message to signal that the intermediate output is ready. The subsequent node responsible for executing the next set of DNN layers issues a legacy Interest packet to retrieve the intermediate output. Once the data are fetched, it proceeds with executing its assigned blocks. The procedure is iteratively repeated by each intermediate node involved in the SI task until inference is complete.

Inference completion. Upon completing the final portion of the DNN model, the node responsible for the last stage of the inference issues an *Int-Not* message to notify the client that the final output is ready for retrieval. The client then retrieves the result using the legacy Interest/Data packet exchange, completing the split inference process.

B. The splitting policy

The entity responsible for orchestration determines the appropriate splitting strategy based on the capabilities of the discovered in-network nodes and the client-specific requirements carried in the augmented Interest packets. Depending on the desired optimization goals, the policy may aim at minimizing the overall inference latency [19], or reducing energy consumption [20], limiting network data exchange, or ensuring fair distribution of computational load across nodes [21].

The proposed NDN-based framework is agnostic to the specific policy employed and is designed to support any such strategy.

C. The caching strategy

Due to the size and nature of data exchanged during a SI task, whether the input data, the intermediate output or the final inference result, typically multiple Data packets may need to be retrieved and the corresponding Interest packets be issued accordingly.

If Data packets are lost along the path, intermediate nodes can facilitate local recovery by retransmitting cached copies of the lost packets, avoiding costly end-to-end retransmissions. For example, considering the topology in Fig. 1, if a Data packet transmitted from client D is lost on the link between E_1 and E_2, node E_2 simply reissues the Interest. Since E_1 had cached the packet in its CS, it can quickly retransmit it, enabling hop-by-hop recovery.

The size of the intermediate output varies depending on the DNN model and the chosen splitting point. In particular, convolution layers often amplify the data volume, producing features significantly larger than the raw input data [7], a phenomenon known as the data amplification effect [22]. Conversely, the final inference output is typically negligible compared to both input data and intermediate outputs [23].

Furthermore, different types of Data packets generated during an SI task may warrant different caching policies. Input and intermediate outputs are usually task-specific and unlikely to be reused to serve other requests, whereas final inference results may be relevant to multiple clients. For example, at a road intersection, several vehicles may request object detection on the same or similar image/video frames.

To address this, we extend the vanilla NDN design, which typically applies a uniform LRU replacement policy to all cached packets, to introduce differentiated caching mechanisms based on packet type. The use of name prefixes, such as $/input$, $/intermediate$, and $/output$ embedded in Interest and Data packet names, enables nodes to easily identify and apply the appropriate caching policy:

- Input and intermediate outputs. These type of packets are temporarily cached only until the inference task is complete, to support local recovery in the event of packet loss. To manage storage efficiently, when intermediate nodes are traversed by Data packets with the $/output$ name prefix (indicating task completion), they purge corresponding Data packets with name prefix $/input$ and $/intermediate$ from their CS.

- Final inference results. These type of packets are cached according to a freshness-based policy. The node executing the last DNN block sets the FRESHNESSPERIOD field in the resulting Data packet based on the maximum tolerated latency carried in the original Interest. Once the freshness period expires, the packet is considered stale and discarded.

This differentiated approach minimizes unnecessary storage overhead, ensuring that caching remains efficient and tailored to the needs of SI applications.

IV. PERFORMANCE EVALUATION

Simulations were carried out using ns-3, specifically leveraging ndnSIM [15], a dedicated ns-3 module developed by the NDN community to accurately model and simulate the NDN architecture.

A. Simulation settings

1) Network topology: Since SI is assumed to occur along the path between the end-device and the cloud, we simulate a chain topology with five nodes. This setup is representative of a typical linear forwarding path in access/edge networks, where traffic from an end-device must traverse multiple intermediate nodes before reaching a cloud server.

The topology, along with the bandwidth and latency settings of the interconnecting links, are depicted in Fig. 3.

In our scenario, the end-device (D) initiates inference requests; three in-network nodes (E_1, E_2, E_3) located within the edge domain are potential DNN executors; and a cloud server (C) acts both as a powerful executor and the orchestrator of the SI task. The nodes differ in computing capabilities, with clock frequencies respectively equal to 0.9, 1.5, 2.5, 2.5 and 4 GHz for D, E_1, E_2, E_3, C.

To emulate realistic edge conditions, we introduce packet loss on the link between E_1 and E_2, representing a congested or intermittently degraded segment. We assume a variable packet loss probability, ranging from 0.01 to 0.2, allowing us to evaluate system behavior under different levels of network impairment.

2) Model: Without loss of generality, we adopt MobileNetV1 [24] as the Convolutional Neural Network (CNN) model for inference tasks. The input data size is set to 330 kB. The model is logically divided into seven consecutive blocks, each corresponding to a group of adjacent layers, according to the functional split approach in [6], [9], [10].

Figure 2 summarizes the characteristics of each block in terms of included layers, computing demands (expressed in GFLOPS), and size of the generated intermediate output. Results in terms of GFLOPS, analytically derived similarly to [14], have been validated with the TensorFlow profiler[1].

Block 1, corresponding to model layers 0-6, is in charge of the initial processing of raw data, starting with the extraction of relevant information. It has also the largest intermediate output size. Block 2, encompassing model layers 7-13, refines the features identified in the previous block. Block 3, from layer 14 to layer 26, reorganizes the information by reducing its spatial complexity. More abstract and complex patterns are captured by Block 4, including layers from 27 to 39. Block 5, the most computation-demanding, includes layers from 40 to 76, arranges the final representation and optimizes it for the synthesis phase. Block 6, from layer 77 to 87, collects and condenses the information into a synthetic and easily interpretable form. The final Block 7, from layer 87 to 90, assigns the input to one of the expected categories.

B. Benchmarks and metrics

We evaluate the performance of the proposed NDN-based approach against a legacy TCP/IP solution, which serves as the

[1]https://www.tensorflow.org/api_docs/python/tf/compat/v1/profiler/profile

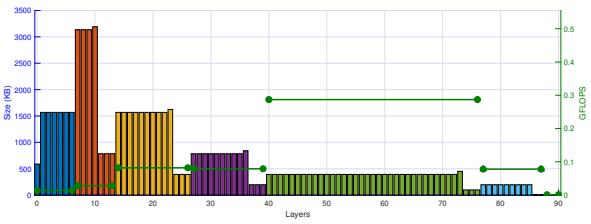

Fig. 2. MobileNetV1 blocks structure.

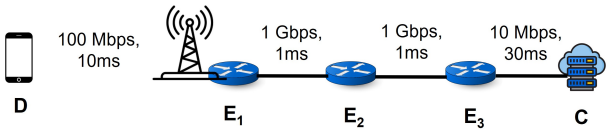

Fig. 3. Simulated network topology.

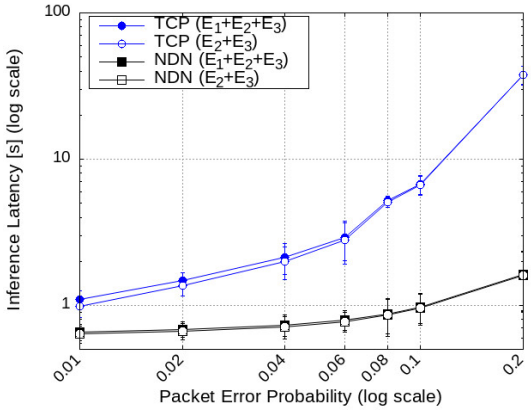

Fig. 4. Inference accomplishment time.

de facto standard for end-to-end, host-centric communication in today's Internet. The TCP/IP protocol stack is indeed currently adopted by contemporary distributed applications, ranging from cloud-hosted services to edge computing deployments and thus, provides a relevant and representative benchmark for our study.

We consider two distinct SI configurations that differ in the placement of the edge executors. In the first case, inference is executed in nodes E_2 and E_3, with E_2 handling blocks B1-B4 and E_3 handling blocks B5-B7. In the second case, inference is executed in nodes E_1, E_2 and E_3, with E_1 handling blocks B1-B2, E_2 handling blocks B3-B5 and E_3 handling blocks B6-B7. The former configuration is representative of a solution aiming at minimizing the data exchange in the network, the latter one instead targets load balancing by splitting the task among all the in-network nodes, according to their computing capabilities. For both cases, we assume that data exchange can occur through TCP/IP and through NDN.

The payload size of the data packets is set to 1000 bytes in all TCP and NDN configurations.

The following performance metrics are considered:

- *Inference latency:* the end-to-end delay required to complete an inference request, measured since the instant the request is issued (first Interest or first TCP segment) until the inference result becomes available at the requester. This metric includes both the computation time of the involved model blocks and the transfer time of input and intermediate output data.

- *Data packets:* the total number of payload-carrying packets (or TCP segments) transmitted across all hops involved in delivering input and intermediate output. A packet traversing h hops contributes h to this count. Control traffic (e.g., TCP ACK/SYN/FIN, NDN Interests) is excluded. The final inference output is negligible in size compared to input and intermediate output [23] and is therefore also omitted.

Results are averaged over 20 runs and reported with 95% confidence intervals.

C. Results

Metrics of interest are reported in the following when varying the packet error probability experienced over the link interconnecting E_1 and E_2.

The inference latency is shown in Fig. 4. The highest latency values are observed when TCP is considered (blue curves) with poorer performance as the packet error probability increases. This is because TCP must perform end-to-end retransmissions from the original sender of a given packet, increasing inference completion time. Whatever the splitting configuration, NDN-based solutions (black curves) exhibit a smaller latency growth with increasing loss probability, due to their hop-by-hop recovery mechanism based on in-network caching, which localizes recovery to the lossy link rather than requiring end-to-end retransmissions.

No remarkable differences are observed for the different SI strategies, which instead differ in terms of exchanged Data packets, Fig. 5. As expected, the splitting among E_1, E_2, E_3 entails a large amount of data to be exchanged. As the packet error probability increases, all TCP-based configurations experience a growth in packet count, reflecting the overhead introduced by end-to-end retransmissions in response to losses. Conversely, NDN-based configurations exhibit a more moderate increase in packet count with rising error probability. This behavior stems from NDN's hop-by-hop recovery, which confines retransmissions to the lossy link rather than requiring complete end-to-end retransmissions.

To better illustrate the benefits of in-network caching, Fig. 6 reports the average number of hops traversed by Data packets when considering the input data retrieval from E_2, in the SI configuration $E_2 + E_3$. In the TCP case, packets transmitted from the end-device D to E_2 always traverse two hops, since any lost packet must be retransmitted end-to-end from the original source (i.e., node D). In contrast, with NDN the average hop count decreases as the packet error probability increases. This effect stems from the higher likelihood of retrieving lost packets from the intermediate cache (node E_1), which shortens the retrieval path and thus reduces latency.

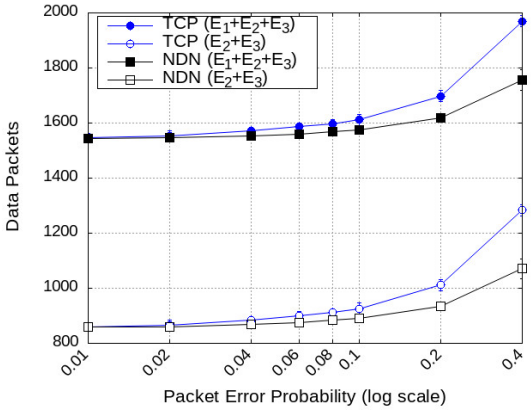

Fig. 5. Number of transmitted Data packets.

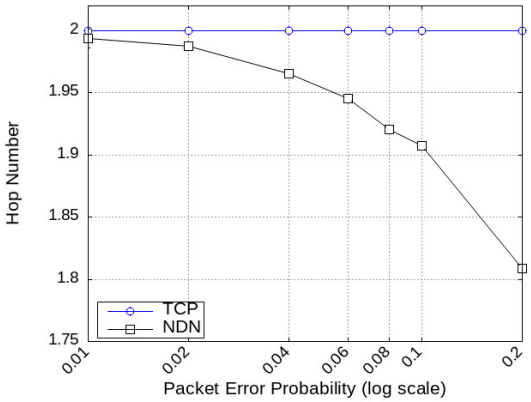

Fig. 6. Average number of hops for Data packets exchanged between the end-device D and node E_2 when considering the E_2+E_3 SI configuration.

V. Conclusions

In this work, we advocate NDN as a key enabler for in-network SI. We extend this networking paradigm to orchestrate SI tasks across nodes distributed along the cloud-to-device continuum, enabling efficient and reliable data exchange among them. Simulation results in a chain topology with induced packet losses demonstrate that NDN consistently achieves lower inference latency and reduced retransmission overhead compared to legacy TCP/IP approaches, primarily due to its name-based data retrieval and hop-by-hop recovery mechanism.

Acknowledgment

This work has been partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART") and partially carried out within the national research project PRIN 2022 "TOGETHER: models and algoriThms fOr 6G Era digital Twin orcHEstratoR" funded by the European Union - Next Generation EU, Mission 4 Component 1, CUP C53D23000450006.

References

[1] S. Kianpisheh and T. Taleb, "A survey on in-network computing: Programmable data plane and technology specific applications," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 701–761, 2022.

[2] R. Silva, D. Corujo, J. Quevedo, and R. Aguiar, "In-network computing—challenges and opportunities," *Internet Technology Letters*, vol. 7, no. 3, p. e487, 2024.

[3] I. Kunze *et al.*, "RFC 9817: Use Cases for In-Network Computing," 2025.

[4] C. Zheng, X. Hong, D. Ding, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "In-network machine learning using programmable network devices: A survey," *IEEE Communications Surveys & Tutorials*, vol. 26, no. 2, pp. 1171–1200, 2023.

[5] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–30, 2022.

[6] D. Xu, X. He, T. Su, and Z. Wang, "A survey on deep neural network partition over cloud, edge and end devices," *arXiv preprint arXiv:2304.10020*, 2023.

[7] W. Shi, Y. Hou, S. Zhou, Z. Niu, Y. Zhang, and L. Geng, "Improving device-edge cooperative inference of deep learning via 2-step pruning," in *IEEE INFOCOM WKSHPS*, pp. 1–6, IEEE, 2019.

[8] H. Liu, M. E. Fouda, A. M. Eltawil, and S. A. Fahmy, "Split DNN inference for exploiting near-edge accelerators," in *2024 IEEE International Conference on Edge Computing and Communications (EDGE)*, pp. 84–91, IEEE, 2024.

[9] J. He *et al.*, "Functional split of in-network deep learning for 6G: A feasibility study," *IEEE Wireless Communications*, vol. 29, no. 5, pp. 36–42, 2022.

[10] M. Spina *et al.*, "In-Network Computing and Split-AI in 6G: Enablers and Proof-of-Concept Studies," in *IEEE 6GNet*, pp. 135–143, 2024.

[11] H. Lee, H. Ko, C. Bae, and S. Pack, "Accelerating convolutional neural network inference in split computing: An in-network computing approach," in *IEEE ICOIN*, pp. 773–776, 2024.

[12] L. Zhang *et al.*, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[13] M. Amadeo and G. Ruggeri, "Exploring In-Network Computing with Information-Centric Networking: Review and Research Opportunities," *Future Internet*, vol. 17, no. 1, p. 42, 2025.

[14] M. Amadeo, C. Campolo, A. Molinaro, G. Ruggeri, and G. Singh, "In-Network Edge Split Inference via Named Data Networking," in *2025 IEEE 11th International Conference on Network Softwarization (NetSoft)*, pp. 1–4, IEEE, 2025.

[15] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM 2.0: A new version of the NDN simulator for NS-3," *NDN, Technical Report NDN-0028*, 2015.

[16] C. Campolo *et al.*, "Towards named AI networking: Unveiling the potential of NDN for edge AI," in *ADHOC-NOW 2020*, pp. 16–22.

[17] A. Agiollo, E. Bardhi, M. Conti, N. Dal Fabbro, and R. Lazzeretti, "Anonymous federated learning via named-data networking," *Future Generation Computer Systems*, vol. 152, pp. 288–303, 2024.

[18] M. Amadeo, C. Campolo, G. Ruggeri, and A. Molinaro, "Improving communication performance of Federated Learning: A networking perspective," *Computer Networks*, p. 111353, 2025.

[19] J. Lee, H. Lee, and W. Choi, "Wireless channel adaptive DNN split inference for resource-constrained edge devices," *IEEE Communications Letters*, vol. 27, no. 6, pp. 1520–1524, 2023.

[20] X. Li and S. Bi, "Optimal AI model splitting and resource allocation for device-edge co-inference in multi-user wireless sensing systems," *IEEE Transactions on Wireless Communications*, 2024.

[21] Y. Xu, T. Mohammed, M. Di Francesco, and C. Fischione, "Distributed assignment with load balancing for DNN inference at the edge," *IEEE Internet of Things Journal*, vol. 10, no. 2, pp. 1053–1065, 2022.

[22] J. Shao and J. Zhang, "Communication-computation trade-off in resource-constrained edge inference," *IEEE Communications Magazine*, vol. 58, no. 12, pp. 20–26, 2021.

[23] C. Campolo, G. Genovese, A. Iera, and A. Molinaro, "Virtualizing AI at the distributed edge towards intelligent IoT applications," *Journal of sensor and actuator networks*, vol. 10, no. 1, p. 13, 2021.

[24] A. Howard *et al.*, "Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint arXiv:1704.04861*, 2017.