future internet

Article

Data Structure and Management Protocol to Enhance Name Resolving in Named Data Networking

Manar Aldaoud [1,*], Dawood Al-Abri [1,*], Medhat Awadalla [1,2] and Firdous Kausar [1,3]

[1] Department of Electrical and Computer Engineering, Al-Khoudh, Sultan Qaboos University, Muscat P.O. Box 33, Oman; medhatha@squ.edu.om (M.A.); firdous@squ.edu.om (F.K.)
[2] Department of Computers and Systems, Helwan University, Cairo 12612, Egypt
[3] Computer Science Department, Fisk University, Nashville, TN 37208, USA
* Correspondence: s117905@student.squ.edu.om (M.A.); alabrid@squ.edu.om (D.A.-A.); Tel.: +968-99565201 (M.A.)

Abstract: Named Data Networking (NDN) is a future Internet architecture that requires an Inter-Domain Routing (IDR) to route its traffic globally. Address resolution is a vital component of any IDR system that relies on a Domain Name System (DNS) resolver to translate domain names into their IP addresses in TCP/IP networks. This paper presents a novel two-element solution to enhance name-to-delivery location resolution in NDN networks, consisting of (1) a mapping table data structure and a searching mechanism and (2) a management protocol to automatically populate and modify the mapping table. The proposed solution is implemented and tested on the Peer Name Provider Server (PNPS) mapping table, and its performance is compared with two other algorithms: component and character tries. The findings show a notable enhancement in the operational speed of the mapping table when utilizing the proposed data structure. For instance, the insertion process is 37 times faster compared to previous algorithms.

Keywords: Named Data Networking; Named Data Border Gateway Protocol; Peer Name Provider Server; component trie; NameComponent; hashing; Longest Prefix Match; Type-Length-Value

Citation: Aldaoud, M.; Al-Abri, D.; Awadalla, M.; Kausar, F. Data Structure and Management Protocol to Enhance Name Resolving in Named Data Networking. *Future Internet* **2024**, *16*, 118. https://doi.org/10.3390/fi16040118

Academic Editor: Paolo Bellavista

Received: 28 February 2024
Revised: 18 March 2024
Accepted: 26 March 2024
Published: 30 March 2024

1. Introduction

Named Data Networking (NDN) architecture is one of the revolutionary projects of the future Internet architectures, where data retrieval is based on the content name rather than the destination IP address. Within NDN, there are two packet types: (1) Interest packets, which convey the content name requested by the consumer, and (2) Data packets, which transport the requested content and trace the path of the Interest packet back to the consumer. The content contained in Data packets is copied and stored within the in-network content routers. Each router consists of three main components: (1) network interfaces, referred to as "Faces", (2) three tables: Content Store, Pending Interest Table, and Forwarding Information Base (FIB), and (3) a forwarder known as Named Data Networking Forwarding Daemon. NDN has many characteristics, such as caching, mobility, hierarchical names, multi-pathing, and security [1].

Named Data Border Gateway Protocol (N-BGP) is a BGP extension that is backward compatible with the existing inter-domain routing protocol. N-BGP conveys, processes, and stores IP-based and name-based routing information between the BGP domains via the N-BGP hybrid speakers [2]. Peer Name Provider (PNP) [3] is a name directory that works as a name resolver and works alongside N-BGP to provide global connectivity between NDN consumers and producers. This name directory has two main components: Server (PNPS) and Client (PNPC). PNP improves NBGP scalability by (1) saving the original gigantic routing table as a mapping table in the PNPS instead of having it on the speaker's FIB table and (2) offloading the name lookup process from the routers to the consumers, where the consumer will perform a lookup to locate the content directly. The proposed scheme in [3] has some shortcomings: (1) The scheme data structure was not taken into consideration. However, the mapping table data structure must be optimized to speed up the operations that occur on it such as searching, adding, and removing, and (2) the way of filling, populating, and operating the mapping table was out of scope, which must be automated to increase the simplicity and decrease the human mistakes when modifications occur to the mapping table entries.

In this paper, we propose a two-element solution that is suitable for any two-column mapping and resolving tables in NDN networks to populate and keep the global database of producers' names and their delivery locations up to date rapidly and automatically.

The two elements in the proposed solution are (1) a mapping table data structure that combines the trie data structure and hashing mechanism to build the PNP mapping table where the name is split into components, and each component is encoded, hereafter referred to as Encoded Component Hash Trie (ECHT), and (2) a PNP protocol that manipulates the pairs of producer and delivery locations in the mapping table via four messages (add, remove, get, and getbcm). This aims to automate population and operations within the mapping table, enhancing efficiency by simplifying processes and reducing human errors during modifications to mapping table entries.

Moreover, we propose a new search mechanism that is based on the Best Component Match (BCM) instead of using the Longest Prefix Match (LPM) mechanism. Utilizing BCM as a lookup mechanism ensures accuracy by components-based matching and helps avoid incorrect replies due to partial component matching errors. Additionally, employing BCM enables faster iteration over components compared to iterating over individual characters, enhancing the efficiency of the operations. Finally, the test results show that the lookup process is 98.47% faster when ECHT is used in comparison with normal string component trie. Accordingly, the topological-locator resolution shortcoming, mentioned in [4], is overcome with an optimal solution.

The remainder of this paper is structured as follows: Section 2 provides a review of related works. Section 3 includes the proposed data structure and search mechanism and then presents our proposed PNP protocol. Section 4 describes the lab setup and presents the performance evaluation of the proposed mechanism and a use case scenario. Section 5 includes the conclusion and future work.

2. Related Work

Several research papers have focused on implementing NDN tables using trie-based solutions. The majority of works related to name lookup in NDN networks are summarized in two surveys cited in [4,5]. However, we are going to focus on component-based and encoded component-based tries, since our work is based on this trie type. In [6], Wang et al. proposed the Name Prefix Trie (NPT), which is the first NDN component trie that represents the name components in edges and the lookup states in nodes. Since NPT is a component trie, the memory consumption is less than in the character trie. However, the depth of NPT can be considered unbounded, and accordingly, the name lookup speed is relatively high.

Seo and Lim introduced a priority NPT (p-NPT) in [7], which separates on-chip processing from off-chip processing so that most of the Longest Prefix Matches (LPMs) are executed with on-chip memories, and accordingly, the off-chip memory access is reduced. The essential difference between p-NPT and normal NPT is that the empty nodes in NPT are replaced with leaf nodes. Consequently, it has less memory consumption and less lookup time.

Lee and Lim [8] utilize Patricia Trie to propose a path-compressed NPT (PC-NPT) to resolve issues in NPT. This was achieved by eliminating empty nodes and consolidating any single child with its parent, thereby decreasing the trie's depth. According to their findings, PC-NPT decreases the number of accessed nodes by 21% compared to NPT, thereby improving lookup and search speed.

In [9], Wang et al. tried to improve NPT by proposing a Name Component Encoding (NCE) that implements an Encoded Name Prefix Trie (ENPT) for the FIB table. They also proposed a State Transition Array (STA) to maintain the state and transition of the ENPT. It has been noticed that NCE reduces the size of the FIB table. However, the NCE complexity is increased by using STA, and the mapping between codes and components reduces the lookup speed.

Feng et al. [10] proposed a Component Hash Encoding (CHE), which uses recursive hashing to hash all children belonging to the same parent, i.e., the hash code of the fourth child is related to the third one, the hash code of the third child is related to the second child, and so on. After encoding the components, the lookup process is performed by an STA. According to their results, the CHE consumes less memory compared to Name Character Tries (NCT). However, the recursive hashing of related children increases the encoding complexity.

Saxena et al. [11] proposed RaCE, which uses both component encoding and Patricia Radix path compression trie where nodes with one child are merged. Accordingly, the depth of the trie decreases, and the lookup speed increases. Nevertheless, lookup time in RaCE is affected by the component-to-code mapping process.

In our proposed data structure, ECHT, we combine multiple specifications to speed up the main operations that may occur in the PNPS mapping table, such as utilizing (1) NameComponent encoded in Type-Length-Value (TLV) format, (2) component trie-based data structure, and (3) a hash mechanism.

NDNS is the main work that discusses global name resolution. Afanasyev et al. [12] proposed a DNS-like zone-based name service for NDN to fulfill the demand for a name look-up service in the NDN network. Although NDNS considers multiple aspects and mimics the Domain Name System (DNS), which is the main mapping system in the current Internet, it has a centralized issue, and the name registry is controlled by the Internet Corporation for Assigned Names and Numbers. However, we offer a new mapping management protocol that has the capability to update and modify PNPSes mapping and resolve tables. Our proposed protocol helps in solving the global inter-domain scalability issue. Moreover, our protocol gives the producer the liberty to choose its consumers at their corresponding Autonomous System (AS) domain level.

3. Proposed Solution

Our proposed two-element solution is shown in Figure 1. The first element of the solution involves improving a two-column mapping table data structure and streamlining insertions, deletions, and lookups through the introduction of the Enhanced Coding Hash Trie (ECHT) data structure. The second element of the solution entails the implementation of the PNP management protocol, aimed at automating and enhancing the efficiency of the population and operations within the mapping table.

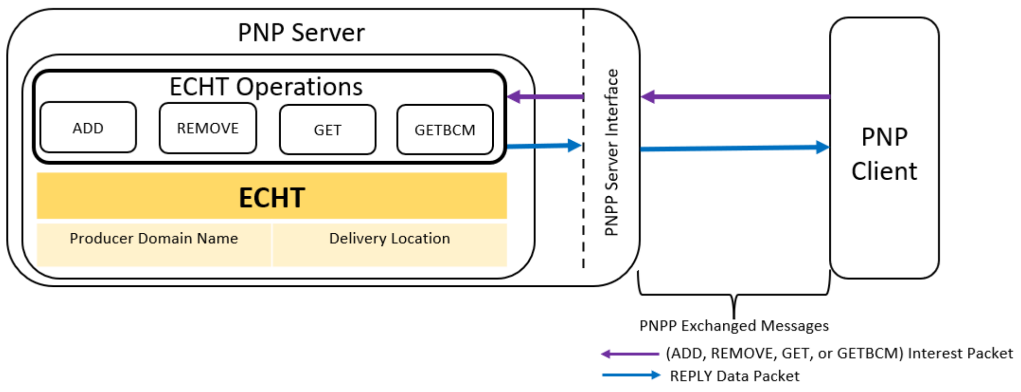


Figure 1. Proposed two-element solution.

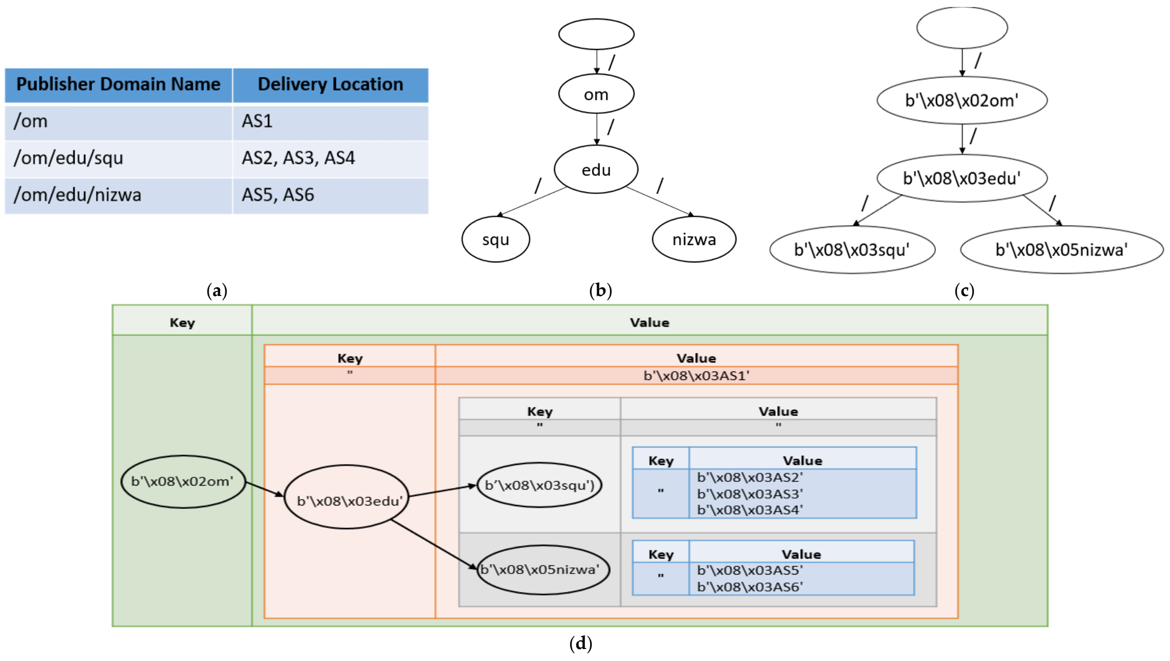
3.1. Proposed Encoded Component Hash Trie (ECHT) Data Structure

The main goal of proposing Encoded Component Hash Trie (ECHT) is to have a data structure that stores and organizes entries in the PNPS mapping table in order to access and process them efficiently. An entry is a pair made up of the domain name, which is the entry key, and the delivery location AS number, which is the entry value, i.e., ("/om/edu/squ", "/AS2, /AS3, /AS4"). In ECHT, we focused on speeding up lookup operations (get, getbcm) over modification operations (add, remove). The main characteristics that distinguished our proposed data structure include (1) combining component trie with hashing to rapidly retrieve the delivery location of the requested NDN name, (2) encoding each component as a NameComponent in Type-Length-Value (TLV) format [13], (3) hashing the encoded component with a SipHash proposed in [14] and using the Open Addressing Collision Resolution technique to resolve the hashing collision problem, and finally, (4) proposing a Best Component Match (BCM) mechanism, where the matching is done per component to retrieve the corresponding delivery location instead of using the LPM mechanism.

3.1.1. ECHT Design Overview

Trie is a tree data structure that is comprised of nodes and edges. It starts with an empty node called the "root node", which forks into several child nodes using edges. Both nodes and edges are associated with values. The ordered concatenation of the traversed nodes and edges while walking a path represents a tree key.

In our work, a component trie represents a traditional trie in which each node value is associated with an NDN NameComponent in a human-readable string format. Edges, on the other hand, are associated with the NDN name delimiter '/'. Consequently, an encoded component trie is a component trie where the NDN NameComponent is encoded as a pure byte string encapsulated in a TLV-style header [15], e.g., the TLV format of the NameComponent 'squ' is b'\x08\x03squ', where 'b' stands for byte, '\x08' represents the type of the name, which is a GenericNameComponent [16], '\x03' represents the length of the NameComponent, which is 3 bytes, and 'squ' is the value of the NameComponent. We use this type of encoding to match the NDN wire format for encoded name components, thus reducing the data type conversion when processing NDN messages and replies. Figure 2 represents both component and encoded component tries.



Publisher Domain Name	Delivery Location
/om	AS1
/om/edu/squ	AS2, AS3, AS4
/om/edu/nizwa	AS5, AS6

Figure 2. ECHT design: (**a**) original mapping table, (**b**) ECHT component-based trie, (**c**) TLV encoded trie, (**d**) ocmbined structure (trie and hash table).

A hash table is a collection of hashed key and value pairs, where data are considered to be a value uniquely identified with a key. After having a key–value pair, hash tables require a hashing function that requests both the key and the value as input. According to the generated value, an index is generated to the location of the value corresponding to the specific key stored. The hash table's significant value over other data structures is its speed in insertion, lookup, and deletion [17]. In our proposed ECHT, SipHash is chosen due to its simplicity and high speed [14].

However, SipHash suffers from internal collisions. Therefore, we studied two famous collision resolution techniques, separate chaining and open addressing, which are compared in Table 1 [18]. We chose open addressing over separate chaining due to its cache performance and operation complexity.

Table 1. Separate chaining vs. open addressing.

	Separate Chaining	**Open Addressing (Linear Probing)**
Implementation	Simple	More computation
Cache performance	Worse since keys are stored using a linked list	Better, since keys are stored in the same table
Space utilization	1. Waste spaces 2. Extra space for links	1. No waste spaces 2. No need for extra spaces
Clustering	No clustering	Suffers from clustering
Mapping time complexity (worst case)	$O(n)$	$O(n)$
Operations time complexity (worst case)	$O(1+\alpha)$ [19], where $\alpha = \frac{Number\ of\ keys}{Number\ of\ slots\ in\ hash\ table}$	$O\left(\frac{1}{1-\alpha}\right)$ [19], where $\alpha = \frac{Number\ of\ keys}{Number\ of\ slots\ in\ hash\ table} < 1$

Figure 2 depicts the data structure and transformation in ECHT.

3.1.2. ECHT Operations

Three different operations are supported by the ECHT data structure: (1) insertion, (2) deletion, and (3) lookup. Hereafter, we refer to the NDN name of the producer's domain as entry_key and to the producer's corresponding delivery locations as entry_value. The following subsections discuss the ECHT operations.

- **ECHT Insertion and Deletion Operations**

When an ADD Interest is received via our proposed protocol PNPP, which will be covered in the next section, the new name will be added to the mapping table as in the following steps. The name and the location are extracted and represented in the form of entry key value, where the key is the TLV encoded name in byte format as received from the Interest, and the value is a list of encoded delivery locations. After that, each key and its value are passed to ECHT for processing.

The insertion process starts by dividing the key into multiple components, e.g., /om/edu/squ is divided into three components: om, edu, and squ. The first component (e.g., om) is checked against the trie's nodes, starting from the root node. If the component is not found in the trie, it will be added as a new node in the first level with an empty value. This newly created node becomes the root node for the next component, and so on. Once the last component is reached, it will be added with its corresponding past value -if exists-. Each parent node has its own children's keys stored in a hash to speed up the look-up process between keys sharing the same parent. Upon process completion, the corresponding entry_value is returned to the caller.

As for the deletion process, the received entry_key is divided into components, the same as in the insert process. The first component is checked against the trie's nodes, starting from the root node. Once the last component is reached, if found, it will be either deleted if it has no children, or the key_value is set to nil if the current node has one child or more to preserve the child nodes. Upon process completion, the corresponding entry_value is returned to the caller. However, it will be nil if all corresponding delivery locations are removed, or the whole node is removed.

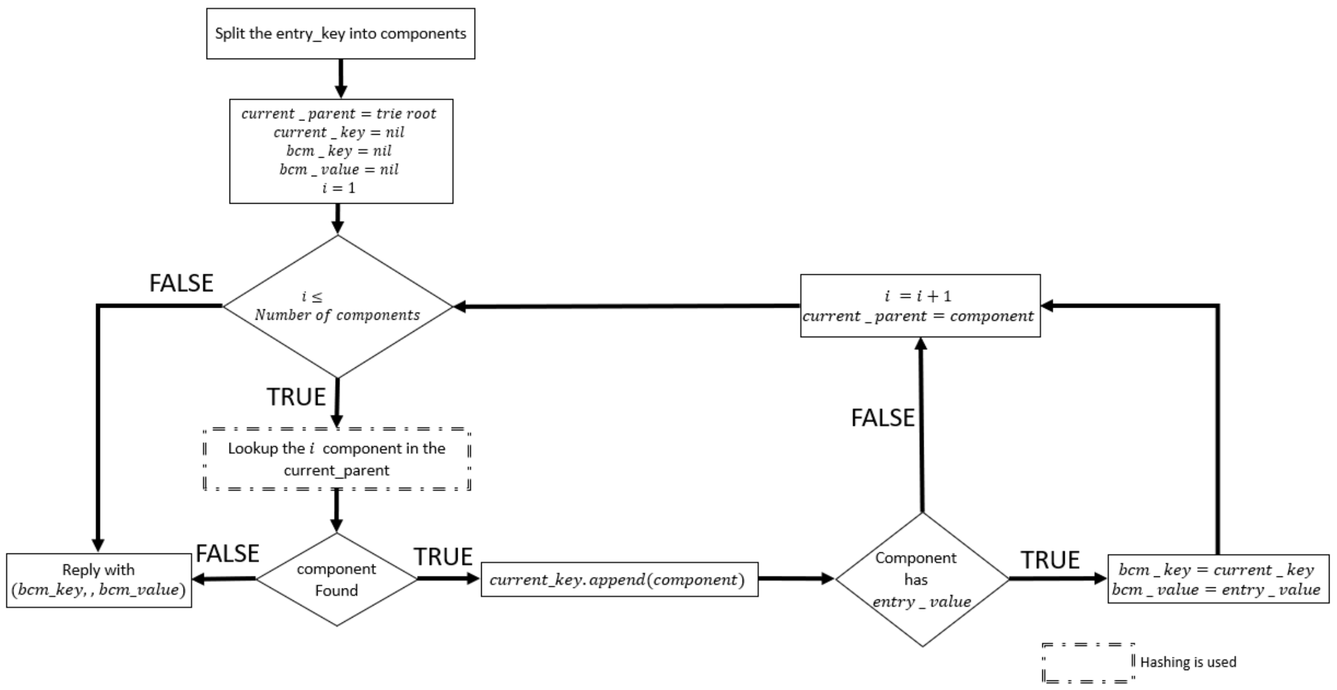
- **ECHT Lookup Operation**

ECHT is designed to provide two types of lookups: searching for an exact match and searching for the best component match. In exact match searching, the trie is traversed based on the components of the entry_key. If the last component is found, its value will be returned to the caller; otherwise, nil is returned.

Instead of the traditional Longest Prefix Match (LPM) search mechanism, we propose a new search mechanism called Best Component Match (BCM). By using BCM, we avoid returning a wrong reply due to an incorrect partial component matching. Also, iterating over components while using hashing is faster than iterating over characters, as shown in Section 5.

In BCM, two variables (bcm_key, bcm_value) are initialized to nil, and then the entry_key is divided into components, and the trie is traversed based on these components. While walking the path, if the traversed node has a valid non-empty value, this value will be stored as bcm_value, and the node key will be appended to the bcm_key. This means that the BCM pair will be updated as needed while traversing the path towards the last component. Both the BCM key and value will be returned as a reply for the BCM lookup. Hence, BCM works the same way as exact matching, with two differences: (1) it stores the last best match while traversing the tree, and (2) it returns two parts of a reply instead of one.

Therefore, there may be multiple cases for the BCM reply to the caller: (1) when no component is matched, a (nil, nil) is returned; (2) when all components are matched, a (bcm_key, bcm_value) is returned, which is equivalent to (entry_key, entry_value); and (3) when few components are matched, a (bcm_key, bcm_value) is returned. The ECHT lookup operation process is shown in Figure 3.



Split the entry_key into components
$current\_parent = trie\ root$
$current\_key = nil$
$bcm\_key = nil$
$bcm\_value = nil$
$i = 1$
$i \leq$ Number of components
FALSE
TRUE
Lookup the i component in the current_parent
component Found
FALSE
TRUE
Reply with $(bcm\_key, , bcm\_value)$
$current\_key.append(component)$
Component has $entry\_value$
TRUE
FALSE
$bcm\_key = current\_key$
$bcm\_value = entry\_value$
$i = i + 1$
$current\_parent = component$
Hashing is used

Figure 3. BCM lookup operation process.

3.2. Proposed PNP Protocol (PNPP)

This section discusses the proposed protocol used by the PNP Client (PNPC) to communicate with the PNP Server (PNPS), which is named PNP Protocol (PNPP). PNPP client-to-server messages (add, remove, get, getbcm) are encapsulated within the ApplicationParameters element of an NDN Interest packet. On the other hand, server-to-client message (reply) is encapsulated within the Content element of an NDN Data packet. PNPP client-to-server messages are mapped one-to-one to ECHT messages, and the ECHT reply is mapped to a PNPP server-to-client message.

The client-to-server messages are usually generated from a producer or normal consumer side either to modify and update the mapping table or to obtain the delivery location(s) of a requested domain name.

The process starts with the PNPS receiving one of the following messages from the client/caller:

- ***ADD:*** A two-component message that contains the producer's domain name and its corresponding delivery locations. Their data types are of NDN name and a list of NDN names, respectively. This message is encapsulated in an ApplicationParameters element of an NDN Interest packet. The Name element of this Interest is set to the NDN name of the PNPS. Upon receiving this message, the PNPS triggers the ECHT insert operation, which will decide if a new node will be added or an entry_value will be modified.
- ***REMOVE:*** On the structure level, REMOVE is similar to ADD, with one difference being the second component is an optional one. If the delivery location component is present, PNPS will modify the associated ECHT entry_value, and when it is absent, the whole node along with its entry_value will be removed from the ECHT.
- ***GET:*** A one-component message that contains the producer's domain name in NDN name format. This message is encapsulated in an ApplicationParameters element of an NDN Interest packet. The Name element of this Interest is set to the NDN name of the PNPS. Upon receiving this message, the PNPS triggers the ECHT exact matching lookup operation.
- ***GETBCM:*** This is the same as GET, but it triggers a BCM lookup operation instead, and therefore, it receives a two-component reply instead of one from ECHT. A sample of a GET message is shown in Figure 4.

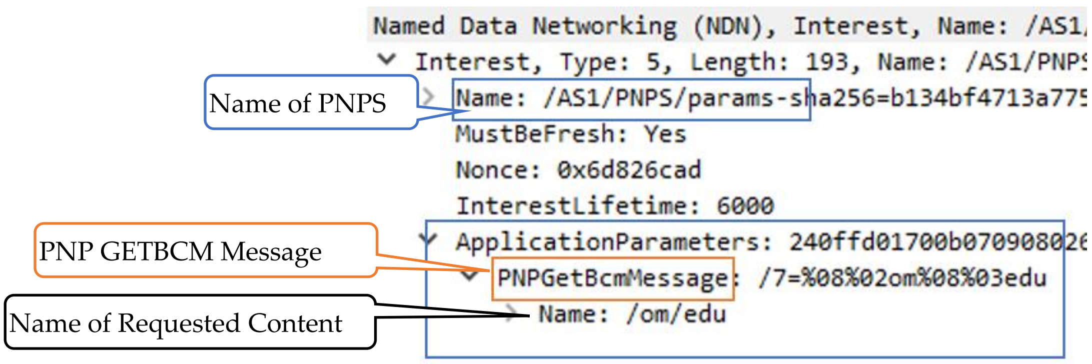

```
Named Data Networking (NDN), Interest, Name: /AS1
  Interest, Type: 5, Length: 193, Name: /AS1/PNP
    Name: /AS1/PNPS/params-sha256=b134bf4713a77
    MustBeFresh: Yes
    Nonce: 0x6d826cad
    InterestLifetime: 6000
    ApplicationParameters: 240ffd01700b07090802
      PNPGetBcmMessage: /7=%08%02om%08%03edu
        Name: /om/edu
```

Figure 4. PNPP GETBCM Interest message.

The process ends by encapsulating the ECHT reply in a Content element of an NDN Data packet and sending it back to the caller.

4. Experimental Setup and Results

Seven experiments were conducted to compare ECHT performance with two text-based algorithms, i.e., component and character tries, which are Google Python-based tries. These tries were implemented following the instructions in [20]. The purpose of considering the component trie was to study the TLV decoding impact, and the purpose of considering the character trie was to study the impact of implementing the trie based on components versus characters. The experiments were measured on a workstation with a

4 Cores Intel(R) Xeon(R) E5-1620 v2 CPU running at 3.7 GHz with 32 GB RAM that runs Ubuntu.22.04 [21]. The three tries were implemented in Python 3.10.

The performance evaluation was carried out using one million names provided in "Majestic Million", which were used for producers' names [22]. All name prefixes were converted from URL form, such as "squ.edu.om", into NDN form, such as "/om/edu/squ". The number of components in a name ranged from two to five. This constitutes the first column of our experiments' dataset. As for the second and last column in our dataset, 10 delivery locations, 4 bytes each, were assigned to each entry in the first column. Some experiments assessed the performance of loading one million names in terms of time and memory consumption. Additional experiments were conducted to evaluate the lookup and addition operations of 100,000 names. Table 2 contains a sample of the dataset used in these experiments. Furthermore, Table 3 contains all the experimental parameters.

Table 2. Dataset sample.

Entry	Producer Name	Delivery Location
1	/com/google	/AS1, /AS3, /AS5, /AS6
2	/com/googletagmanager	/AS2, /AS4, /AS5, /AS6
3	/org/Wikipedia/en	/AS1, /AS7, /AS8, /AS9
. .	. .	. .
1,000,000	/ru/audimanual	/AS2, /AS5, /AS7, /AS9

Table 3. Experimental parameters.

Parameter	Specification
Names dataset	Majestic Million
Number of delivery locations	10
Each delivery location length	4 bytes
Trie used	component, character, ECHT
Number of components per name	2 to 5
Initial dataset	1,000,000 names
Lookup names	100,000 names
Added names	100,000 names

4.1. Time Performance

Five experiments were conducted to test the time performance of (1) the initial loading of one million names to the mapping table and (2) the lookup operation. The first experiment was conducted to study the time for locally loading the entire dataset (one million entries) into an empty trie progressively. By locally, we mean injecting the entries from memory directly into the three algorithms' backends and not through PNPP ADD messages. This allows us to fairly test the trie insertion performance without TLV decoding/encoding.

For this experiment, the dataset was loaded to memory in two copies: (1) native text format, to be used by Component and Character Trie algorithms, and (2) TLV encoded format, to be used by the ECHT algorithm. By using this approach, we ensure fair start points for all algorithms. The dataset was loaded in incremental batches of 100,000 names, as shown in Figure 5.

Figure 5 shows that component trie took the least loading time (4.56 s), since the character trie (33.52 s) had more nodes (character-based). Moreover, component trie also took less loading time than ECHT (30.65 s). This is because ECHT loads NDN names, which need more processing time, since they are longer than string names, i.e., "squ" contains 3 bytes in text format and 5 bytes in NDN TLV encoding format, i.e., b'\x03squ'.

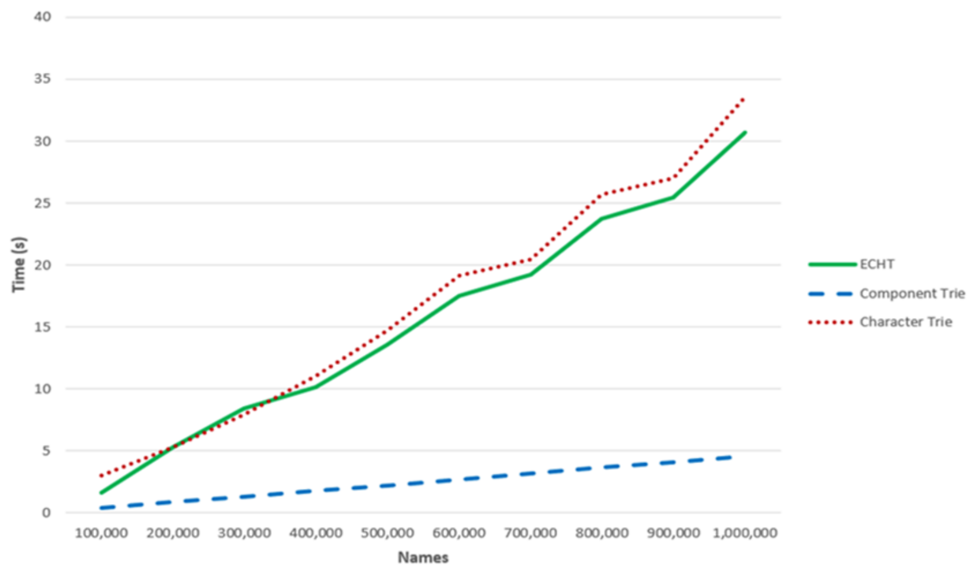


Figure 5. Time performance of initial loading.

The following four experiments aimed to evaluate the time performance of the lookup operation. The main metrics that were used for these experiments are as follows:

- ***GET Positive:*** This experiment focused on testing the exact matching mechanism using a preset of 100,000 different names that were randomly selected from the one million dataset. The preset has one column, and each row is in TLV encoded format, i.e., a list of byte arrays.
- ***GET Negative:*** This experiment focused on testing the exact matching mechanism using the previous preset, but where each row was manipulated as follows, For the sake of simplicity, we will use human-readable strings for names instead of the actual list of byte-arrays:
  - Insert 1-byte-char "x" as a prefix for each name component, i.e., the name "/om/edu/squ" becomes "/xom/edu/squ", "/om/xedu/squ", and "/om/edu/xsqu". This technique ensures that competing algorithms are tested for fast query exits at various depths.
  - Insert 1-byte-char "x" as a suffix for each name component, i.e., the name "/om/edu/squ" becomes "/omx/edu/squ", "/om/edux/squ", and "/om/edu/squx". This technique ensures that competing algorithms are tested for delayed query exits at various depths.
  - Insert 1-byte-char "x" as a suffix component for each name, i.e., the name "/om/edu/squ" becomes "/om/edu/squ/x". This technique ensures that competing algorithms are tested for maximum depth at each query.

This preset size was multiple times bigger than the original 100,000 preset. In the interest of clarity, we normalized the result to 100,000.

- ***BCM Positive:*** This experiment focused on testing the best component matching mechanism using the GET Positive preset.
- ***BCM Negative:*** This experiment focused on testing the best component matching mechanism using the GET Negative preset. This preset size was multiple times bigger than the original 100,000 preset. In the interest of clarity, we normalized the result to 100,000.

As illustrated in Figure 6, the lookup performance of the proposed ECHT was much better than both component and character tries—approximately 76 times faster when the exact matching mechanism was used for the GET Positive case and 11.7 times faster for the Get Negative case. Moreover, when the BCM mechanism was used, ECHT was 65.3 to 71.63 times (98.47 to 98.6%) faster than component trie and character trie, respectively. For the case of BCM Positive and BCM Negative, ECHT was 97.06% faster than component trie and 98.61% faster than character trie. The main reason is that ECHT eliminates the need to decode/encode NDN names while parsing received NDN Interests or while creating NDN Data replies. In Component/Character tries algorithms, domain names have to be decoded to text before searching the trie, and then the results are encoded before sending them back to the caller.

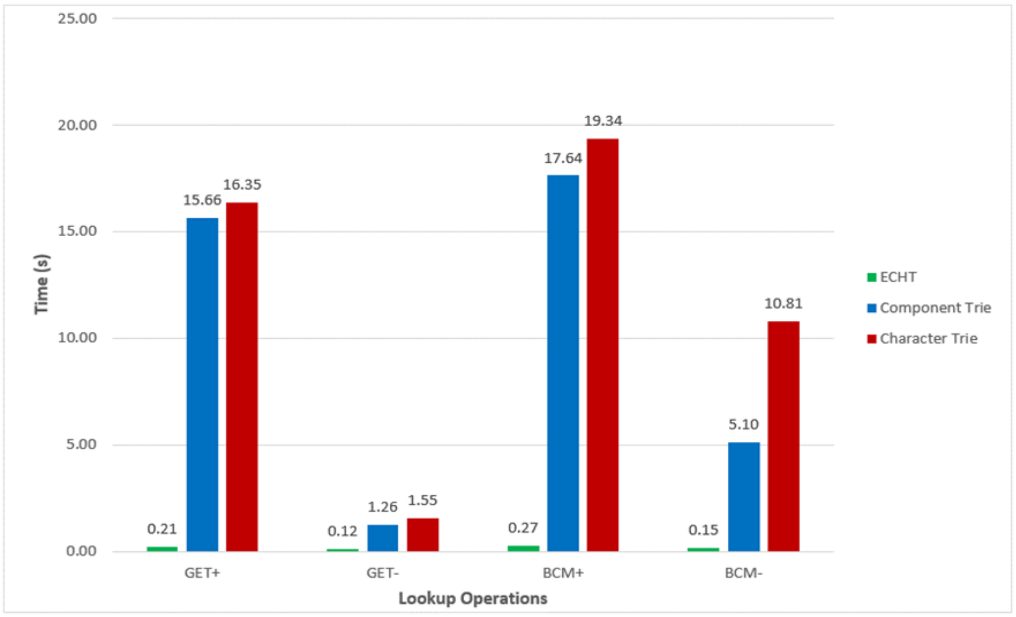


Figure 6. Time performance of 100,000 different lookup operations.

4.2. Throughput Performance

This experiment tested the local insertion operation of 100,000 new entries in TLV encoded format into the previously "1 million" populated tries. In this experiment, ECHT processes the entries and makes replies without any conversion, while Component/Character tries need to TLV decode the entries before processing them and encode the replies before sending them to the caller. The results showed that the insertion throughput is more than 37 times faster in ECHT than in the other tries, which is around 336 names/ms for ECHT and 9 names/ms for the other tries, as detailed in Table 4. The results also indicated that the trie type is of less significance when TLV encoding/decoding is involved, i.e., the conversion is way more expensive than the actual entry addition.

Table 4. Throughput performance of 100,000 entries insertion operation.

	ECHT	**Component Trie**	**Character Trie**
Name/ms	335.964	9.196	8.576

4.3. Memory Consumption

This experiment measured memory consumption while conducting the first experiment. The results showed that to achieve high operation performance, ECHT consumes approximately 4600 MB of memory, which is roughly two-and-a-half times greater than the memory consumption of other trie structures and amounts to around 1500 MB, as shown in Figure 7.

Though both ECHT and component trie operate on a component-based approach, ECHT requires additional memory due to its handling of decoded names, resulting in longer initial dataset loading times. This is because ECHT allocates more bytes for Type, Length, and Value (TLV).

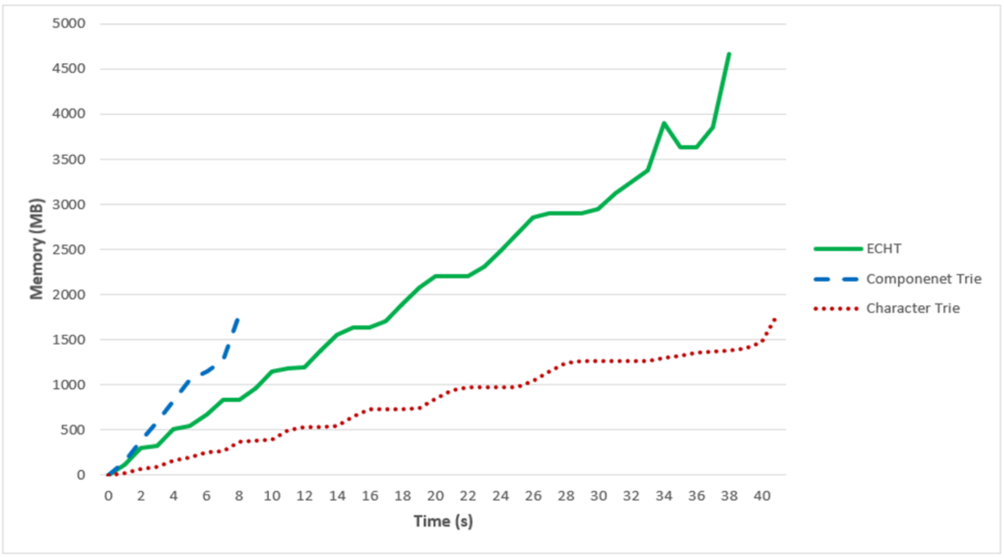


Figure 7. Memory consumption while loading 1 million entries into an empty trie.

4.4. Use Case Scenario

Let us start our scenario with the following assumption: (1) a producer named SQU with the NDN domain name "/om/edu/squ" is located in "AS1", (2) "AS1" and "AS2" administrators are running in their respective domains: "/AS1/PNPS", "/AS2/PNPS", (3) "AS1" and "AS2" are peered using N-BGP [2], and each domain is advertising its own PNPS name, and (4) "C1" and "C2" are three consumers located in "AS1" and "AS2", respectively.

Based on the above assumptions, "C1" can reach SQU via static/intra-domain routing, whereas "C2" cannot reach the SQU domain name, as it is not globally advertised. Therefore, if C2 sends an NDN Interest to the SQU domain, it will receive an NDN Nack (Figure 8a). Additionally, if it sends GET/GETBCM to its domain PNPS, it will receive a nil result, since "/AS2/PNPS" does not have a mapping entry for the SQU domain name (Figure 8b).

SQU decided to solve this issue by providing reachability to its domain for clients in "AS2". To provide reachability, the SQU network administrator uses PNPC to send an ADD message to "/AS2/PNPS", as follows:

pnpc --server = /AS2/PNPS --message = add --name = /om/edu/squ --location = /AS1

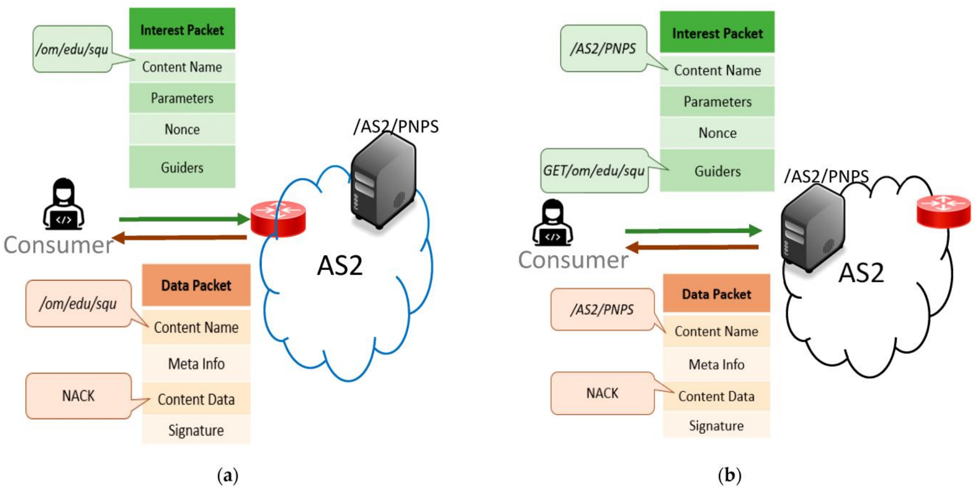
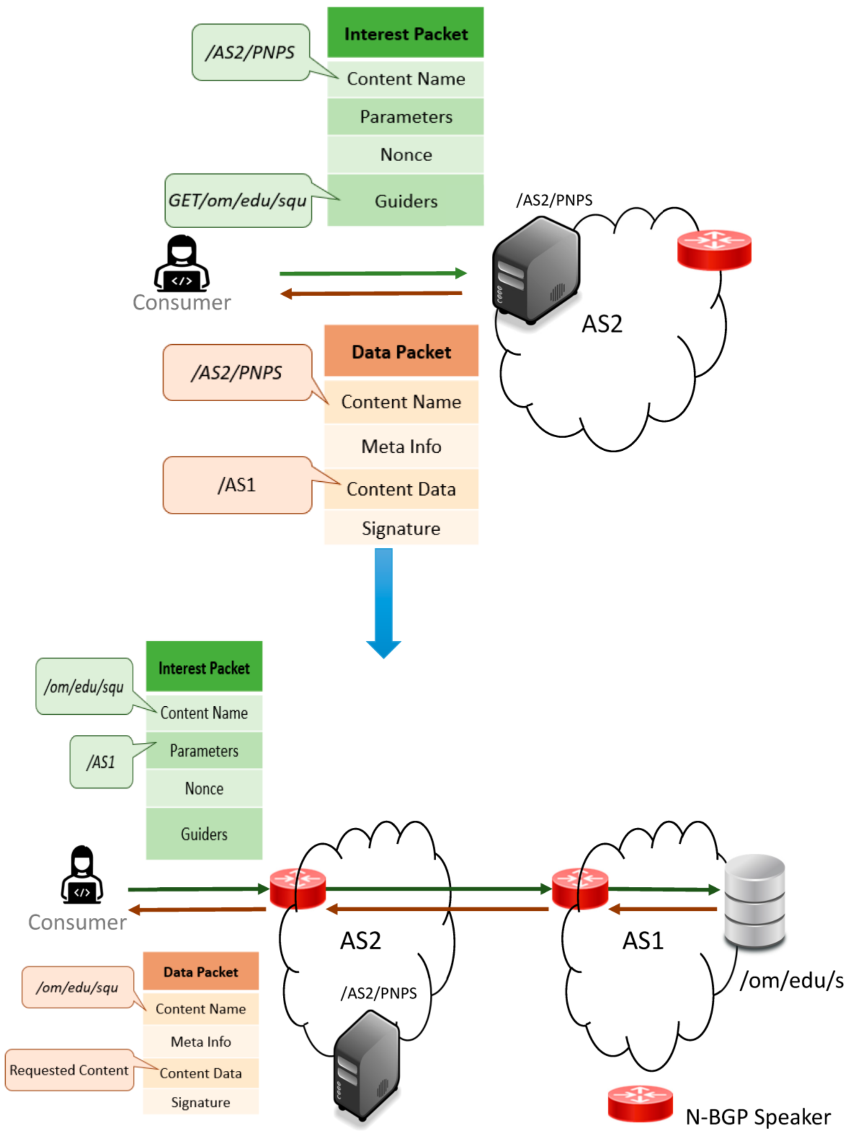


Figure 8. *Cont.*



(c)

Figure 8. Use case scenario, (**a**) No PNPS, (**b**) No entry in mapping table, (**c**) Full solution for global NDN traffic.

Since "/AS2/PNPS" is globally advertised using N-BGP, the NDN Interest packet holding the ADD message can reach its destination. When PNPS of "AS2" receives the NDN Interest, it validates the Interest signature (outside the scope of this paper) against the name "/om/edu/squ". If the Interest packet is signed by an entity that is authorized to modify entries of this domain name, access will be granted, and the ECHT add operation will take place. Otherwise, a "not authorized" data message is sent back.

Now, "C2" will still receive a Nack message when sending an NDN Interest to the SQU domain. However, "C2" can get the delivery location(s) of the SQU domain by sending a GET/GETBCM message to "/AS2/PNPS", which will be "/AS1". "C2" will use "/AS1" as a ForwardingHint [23] for a new Interest packet with the name element "/om/edu/squ". As "/AS1" is publicly advertised via N-BGP, "AS2" can now route the packet to its destination (Figure 8c).

5. Conclusions and Future Work

This paper proposes a two-element solution to enhance name resolution in NDN networks. The two elements in the proposed solution include (1) a data structure of the NDN mapping table and (2) a management protocol. The proposed data structure, called

Encoded Component Hash Trie (ECHT), combines the trie data structure with a hashing mechanism to build the NDN mapping table, where each producer name is decomposed into components, and each component is encoded in NDN NameComponent TLV format. ECHT supports three operations: insertion, deletion, and lookup. The lookup operation in ECHT is based on a proposed searching mechanism called Best Component Match (BCM) as a replacement for the Longest Prefix Match (LPM) mechanism, which assists in avoiding a wrong reply due to an incorrect partial component matching. Moreover, the management protocol is proposed to automatically populate, update, and modify the entries of the mapping table (producer name, delivery location) via four messages (add, remove, get, getbcm). The proposed scheme performance was tested and evaluated against two other tries: component and character tries. The results show that the operations' speed was enhanced and was much faster than other tries, e.g., lookup and insertion operations are around 65 and 37 times faster, respectively, when the ECHT data structure is used in comparison with normal text-based component trie. This is because ECHT does not require decoding/encoding NDN names while parsing received NDN Interests or creating NDN Data replies.

For future work, we aim to develop an authentication framework that can be used by an NDN producer who wants to modify (add or remove) an entry in the mapping table. Accordingly, the origin, authenticity, and integrity of the content producer are verified.

For the convenience of the reader, the abbreviations used in this paper are listed in Table 5.

Table 5. Abbreviations in alphabetical order.

Abbreviation	Description
BMC	Best Component Match
DNS	Domain Name System
ECHT	Encoded Component Hash Trie
FIB	Forwarding Information Base
IDR	Inter-Domain Routing
LPM	Longest Prefix Match
N-BGP	Named Data Border Gateway Protocol
NDN	Named Data Networking
PNP	Peer Name Provider
PNPC	Peer Name Provider Client
PNPP	Peer Name Provider Protocol
PNPS	Peer Name Provider Server
TLV	Type-Length-Value

Author Contributions: Conceptualization, M.A. (Manar Aldaoud), D.A.-A., M.A. (Medhat Awadalla) and F.K.; methodology, M.A. (Manar Aldaoud); validation, M.A. (Manar Aldaoud); formal analysis, M.A. (Manar Aldaoud); investigation M.A. (Manar Aldaoud); resources, M.A. (Manar Aldaoud); data curation, M.A. (Manar Aldaoud); writing—original draft preparation, M.A. (Manar Aldaoud); writing—review and editing, M.A. (Manar Aldaoud), D.A.-A., M.A. (Medhat Awadalla) and F.K.; visualization, M.A. (Manar Aldaoud); project administration, M.A. (Manar Aldaoud) and D.A.-A.; funding acquisition, M.A. (Manar Aldaoud). All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available in this article.

Acknowledgments: The authors would like to thank Sultan Qaboos University for financially supporting this research and publication.

Conflicts of Interest: The authors declare that they have no competing interests.

References

1. Aldaoud, M.; Al-Abri, D.; Awadalla, M.; Kausar, F. Leveraging ICN and SDN for Future Internet Architecture: A Survey. *Electronics* **2023**, *12*, 1723. [CrossRef]
2. Aldaoud, M.; Al-Abri, D.; Awadalla, M.; Kausar, F. N-BGP: An efficient BGP routing protocol adaptation for named data networking. *Int. J. Commun. Syst.* **2022**, *35*, e5266. [CrossRef]
3. Aldaoud, M.; Al-Abri, D.; Awadalla, M.; Kausar, F. Towards a Scalable Named Data Border Gateway Protocol. In Proceedings of the 2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), Male, Maldives, 16–18 November 2022; pp. 1–5. [CrossRef]
4. Anjum, A.; Agbaje, P.; Mitra, A.; Oseghale, E.; Nwafor, E.; Olufowobi, H. Towards named data networking technology: Emerging applications, use cases, and challenges for secure data communication. *Future Gener. Comput. Syst.* **2024**, *151*, 12–31. [CrossRef]
5. Majed, A.; Wang, X.; Yi, B. Name Lookup in Named Data Networking: A Review. *Information* **2019**, *10*, 85. [CrossRef]
6. Wang, Y.; Dai, H.; Jiang, J.; He, K.; Meng, W.; Liu, B. Parallel Name Lookup for Named Data Networking. In Proceedings of the 2011 IEEE Global Telecommunications Conference—GLOBECOM 2011, Houston, TX, USA, 5–9 December 2011; pp. 1–5. [CrossRef]
7. Seo, J.; Lim, H. Bitmap-based priority-NPT for packet forwarding at named data network. *Comput. Commun.* **2018**, *130*, 101–112. [CrossRef]
8. Lee, J.; Lim, H. A new name prefix trie with path compression. In Proceedings of the 2016 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia), Seoul, Republic of Korea, 26–28 October 2016; pp. 1–4. [CrossRef]
9. Wang, Y.; He, K.; Dai, H.; Meng, W.; Jiang, J.; Liu, B.; Chen, Y. Scalable Name Lookup in NDN Using Effective Name Component Encoding. In Proceedings of the 2012 IEEE 32nd International Conference on Distributed Computing Systems, Macau, China, 18–21 June 2012; pp. 688–697. [CrossRef]
10. Feng, S.; Zhang, M.; Zheng, R.; Wu, Q. A Fast Name Lookup Method in NDN Based on Hash Coding. In Proceedings of the 3rd International Conference on Mechatronics and Industrial Informatics (ICMII 2015), Zhuhai, China, 30–31 October 2015. [CrossRef]
11. Saxena, D.; Raychoudhury, V.; Becker, C.; Suri, N. Reliable Memory Efficient Name Forwarding in Named Data Networking. In Proceedings of the 2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES), Paris, France, 24–26 August 2016; pp. 48–55. [CrossRef]
12. Afanasyev, A.; Jiang, X.; Yu, Y.; Tan, J.; Xia, Y.; Mankin, A.; Zhang, L. NDNS: A DNS-Like Name Service for NDN. In Proceedings of the 2017 26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, BC, Canada, 31 July–3 August 2017; pp. 1–9. [CrossRef]
13. Type-Length-Value (TLV) Encoding. Available online: https://docs.named-data.net/NDN-packet-spec/current/tlv.html.accessed (accessed on 6 September 2023).
14. Aumasson, J.-P.; Bernstein, D.J. SipHash: A Fast Short-Input PRF. In *Progress in Cryptology—INDOCRYPT 2012*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 489–508.
15. NDN-Team. NDN Technical Memo: Naming Conventions. (NDN, Technical Report NDN-0022). 2022. Available online: https://named-data.net/techreports.html (accessed on 27 February 2024).
16. NDN-Team. TLV Type Registry: Assigned Numbers. Available online: https://docs.named-data.net/NDN-packet-spec/current/types.html#types (accessed on 12 October 2023).
17. Li, Z.; Xu, Y.; Zhang, B.; Yan, L.; Liu, K. Packet Forwarding in Named Data Networking Requirements and Survey of Solutions. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 1950–1987. [CrossRef]
18. Yusuf, A.D.; Abdullahi, S.; Boukar, M.M.; Yusuf, S.I. Collision Resolution Techniques in Hash Table: A Review. *Int. J. Adv. Comput. Sci. Appl.* **2021**, *12*, 757–762. [CrossRef]
19. GeeksforGeeks. Open Addressing Collision Handling Technique in Hashing. Available online: https://www.geeksforgeeks.org/open-addressing-collision-handling-technique-in-hashing/?ref=lbp (accessed on 4 September 2023).
20. Pygtrie. Available online: https://pygtrie.readthedocs.io/en/latest/ (accessed on 12 October 2023).
21. Hevey, D. Network analysis: A brief overview and tutorial. *Health Psychol. Behav. Med.* **2018**, *6*, 301–328. [CrossRef] [PubMed]
22. The Majestic Million. Available online: https://majestic.com/reports/majestic-million (accessed on 9 October 2023).
23. NDN-Team. NDN Packet Format Specification 0.3 Documentation. Available online: https://named-data.net/doc/NDN-packet-spec/current/intro.html (accessed on 6 September 2023).