Decentralized Search Engine for Information-Centric Network

Erwan Kessler - Yann Meyer
Computer Engineering Students
TELECOM Nancy - University of Lorraine
Villers-lès-Nancy, 54600, France
Email: {erwan.kessler, yann.meyer}@telecomnancy.eu

Thibault Cholez
Associate Professor
Telecom Nancy - University of Lorraine
Nancy F-54000, France
Email: thibault.cholez@loria.fr

***Abstract*—Information-Centric networking is a network paradigm based on naming data rather than physical entities within the network. One of the main purposes of such a paradigm is to decentralize data but data decentralization within the network is insufficient if the use of a search engine implies a centralized index. This paper focuses on a tree based architecture, allowing for a decentralized index while maintaining a deterministic and fast name resolution.**

The node hierarchy induced by the tree based structure allows an efficient network management, leading to a highly scalable architecture while avoiding flooding.

This architecture offers a clear separation between the logical and physical structure of the network, making it difficult to map the network and thus increasing the level of security provided by the underlying Named-data Link State Routing protocol.

***Index Terms*—Blockchain - Decentralized internet - Information Centric Networking (ICN) - Non Cryptographic Hash Functions (NCHF) - Named-data Link State Routing protocol (NLSR) - Named Data Networking (NDN) - Search engine**

May 11, 2020

I. Introduction

Information-Centric Networking (ICN) aims at introducing new communication protocols better suited for current Internet usage such as massive content broadcast and mobile use. The fundamental principle is that locating and addressing content directly, at the scale of the Internet, is more efficient than constantly addressing servers hosting said content (via IP addresses).

Thus, ICN solutions question the founding principles of the Internet by setting up a routing based on the content's name. This allows for a natural multi-source spread of the information, the use of caching or even the inherent handling of mobility and data security. Such features are not inherently supported by IP. Such architectures would theoretically allow for a worldwide spread of the information without requiring the use of a Content Delivery Network (CDN).

However, a worldwide, decentralized spread of the information remains insufficient if a search engine requires a centralized indexation of content. Those restrictions were all taken into consideration when designing this architecture.

Named-Data Networking (NDN) is currently the leading type of ICN.
The purpose of this paper is to introduce an architecture for a decentralized search engine in the NDN environment. It should allow for a retrieval of complete file names satisfying a set of given keywords.
Building a decentralized, fast and scalable search engine has proven to be a very challenging problem and resulted in numerous types of solutions both in the Peer to Peer (P2P) and ICN community.
By studying and exploiting weaknesses of existing solutions in the ICN literature and by considering existing search engines used in P2P file-sharing networks, this paper offers a new architecture for a scalable decentralized search engine for ICN networks.

The architecture presented in this paper uses a tree shaped Distributed Hash Table (DHT) network to spread the indexation of data while maintaining a fast and deterministic name resolution.
The tree shaped DHT differs from the usual DHT solutions by introducing vertical hierarchy between nodes, leading to better network management and better load balancing.
Moreover, this architecture fully splits the logical and physical layout of the network resulting in increased security as mapping the network proves to be more challenging.

In this paper, we will first go through already existing decentralized Search engines, then discuss the Search Engine network architecture, its integration in the Names Data Networking architecture. Finally, limitations and potential improvements will be discussed.

II. Related work

A. Named Data Networking

Named Data Networking (NDN) works on a Data/Interest structure. When a user wishes to obtain a certain data, he will send an Interest to his closest NDN router containing the name of the data. This interest will be routed through the

network until it reaches one of the instances of this Data in the network. Then, a Data packet is sent back to the end user.

NDN routing is based upon 3 main structures used jointly by NDN routers [1] :

- The Forwarding Information Base (FIB) : It stores the forwarding entries used to route Interest packets towards the most likely source of the required Data. Contrary to the IP protocol, FIB may offer multiple routing path for the same query. This table is usually populated by a NDN routing protocol such as the Named-data Link State Routing (NLSR) protocol currently implemented in the NDN testbed.
- The Pending Interest Table (PIT) : It stores all pending Interest packets of the router and the interface on which they were received. This structure is used to send back the result of a query to the end user(s).
- The Content Store (CS) : It is used to cache data of previous Interest packages and greatly improves network efficiency to deliver popular content.

When an Interest reaches an NDN router, it begins by checking its CS. If the CS can be used to answer the Interest, the Data is sent back on the interface it was received.
Else, the Interest is added to the PIT. If the entry is already existing in the PIT, the interface of the new Interest is simply added to the list of Interfaces for the said Interest.
At last, if the Interest cannot be resolved with the CS and does not exist is the PIT, it is forwarded to the next node according to the FIB.
Checking the FIB for a given Interest can result in multiple routes available. Then, forwarding strategy algorithms have to be called to compute the best forwarding process.

B. Named-data Link State Routing protocol

The Named-data Link State Routing protocol (NLSR) is a routing protocol designed to operate on top of NDN. It uses NDN's Data/Interest packets in order to exchange routing information [2].
The NLSR protocol aims at spreading Link State Advertisements (LSAs) across the network in order to create a network topology and spread name prefix reachability. Each node holds a Link State Database (LSDB) to store all LSAs [2]

NLSR distinguishes 2 different types of LSAs : Adjacency LSAs and Prefix LSAs.
An adjacency LSA is used to create a network topology for all NDN routers by indicating adjacency between 2 routers.
A prefix LSA is used to notify that a name prefix has been registered within the router [2]. Such LSAs are used to update NDN routers's FIB
Every LSA is stored in the router's LSBD and periodically replaced by newer versions of said LSA upon reception.

C. Information-Centric Networking Search Engine solutions

Regarding the keyword-based search feature for ICN, most of the time, the suggested solution is to flood the network with the user's request either with its original form or already broken down to network understandable atoms. This approach relies heavily on caching and the fact that the request's result will be available in the closest node most of the time.

Some of those solutions are Keyword-based Breadcrumbs (KBC) [3], Independent Search and Merge (ISM) [4], Integrated Keywords Search (IKS), Similarity content search (SCS) [5]. All of them work on the premise of having a way to identify the keyword and flooding the network, using some of the information about how previous requests were reported to cut the branch of search in the network. This form of query is inefficient when conducted for non existent keywords. Once the network has reached enough request and keep receiving enough, results [3] [4] [5] shows that the overhead is low enough to be manageable.

Another approach is to use prefix based search [6] which will use the keyword to match on Data names directly. This approach is very efficient for multimedia server since most of the time the keyword are indeed contained in the file name. However such a solution falls short when keywords are not represented in the Data name.

An architecture for Object Resolution in Information-Centric Environment (ORICE) [7] introduced a new feature. The Object Resolution Service (ORS) allows name resolving based on keywords. However, the implementation is still fuzzy as it relies on a certification service that will keep track of a dual index, the names added to the network as well as the keyword.

Another implementation also exist with actual CDN with a flat server space and using a hash function, this has the advantage to reduce overhead, however it is not applicable to whole network as it require a knowledge of the entire network. [8]

D. Peer to peer name resolution

Named resolution and content search in Peer to Peer (P2P) network is a very challenging problem. Indeed, the feasibility of such a Search Engine (SE) is widely discussed [9]. Google, as the biggest centralized index, contains more than 2 billion documents. In comparison, traditional full-text keyword search in P2P network tend to struggle when faced with over 100 000 documents [9].
Among P2P name resolution, 2 main approaches stand out : flooding and distributed hash tables (DHTs) [9]–[11].

Flooding architectures such as Gnutella proved to be very poorly scalable and DHTs stands out as the most promising type of scalable P2P name resolution.

The most common approach to DHTs follows a horizontal layout such as CHORD [10]. Such a network achieves a

lookup in $\mathcal{O}(\log n)$ with n being the number of nodes in the network without introducing any form of hierarchy between nodes. While $\mathcal{O}(\log n)$ can seem very efficient, a P2P network can end up containing a very high number of nodes, essentially increasing search times.

By introducing vertical partitioning of the network, [11] new optimisation methods such as Bloom filter and incremental results become available. Those techniques can help increasing the scalability and speed of name resolution services [11].
In particular, when the amount of documents in the decentralized index becomes very high, the system only needs to return a fraction of the matching documents. Bloom filters associated with caching can result in a very efficient way of implementing this feature, increasing the system's speed and reducing the network load. [11]

III. Subject analysis

Performing a name resolution in the ICN network is inherently different than using the network to retrieve a named data.
Indeed, for the retrieval of said data, the network simply has to return to the user one instance of this data that exists somewhere in the network. But when it comes to a search engine query (SEQ), the user wishes to find all named data that exists in the network matching a set of keywords.
The main difference between those 2 operations is that the amount of matching data is unknown in a SEQ, and thus, if we were to operate a SEQ in the same way as we operate a named data resolution, we would have no way to know when all the matching data has been found other than visiting every single ICN node on the network. This method is obviously very time and resources consuming and thus, unusable.
The natural response to that issue would be to introduce an index of all (public) data on the network and use it to return the list of matching data when a user issues a SEQ. Unfortunately, this solution is not satisfying as introducing a centralized index of all data on the network defeats the purpose of a decentralized network.

Thus, establishing a decentralized, high performance and scalable Search Engine in the ICN architecture comes down to finding a way of making sure we can retrieve in an efficient way all the search results without the use of a centralized index.

IV. Search Engine Network architecture

One of the main features of the architecture presented in this paper is that it fully separates its logical structure from its physical one and use replication of index nodes, similarly to distributed indexes in P2P networks. This means that, the logical connections between servers do not match the reality of the network topology.
The advantages of such a feature is it makes very resilient and scalable, while delivering good performances thanks to caching and worldwide replication. Moreover it is more difficult for an attacker to guess the logical structure of the network based on a server's physical connections.
However, this implies that the SEN has to establish a physical routing matching its logical structure. This will be further discussed later in this paper.

A. Structure of the Search Engine Network

The logical structure of the SEN can be represented as a N-ary tree [Fig. 1]. The tree can be seen as a Distributed Hash Table (DHT) where each node is responsible for a given hash range. The hash range represents the keywords that will be stored in said node.

Each node of the tree can be further broken down in a pool of SESs [Fig. 2]. Each server in this node holds the same data and has the same responsibility within the network. The number of servers in a node is determined by how much network traffic said node has to handle. This means that the cardinally of 2 adjacent nodes is not necessarily the same.
This structure allows for data replication as well as a better handling of data and network congestion. While the difference of cardinally between two adjacent nodes might appear to create network congestion, the network can adapt to solve such issue.
Network and data balancing will be further discussed later in this paper.

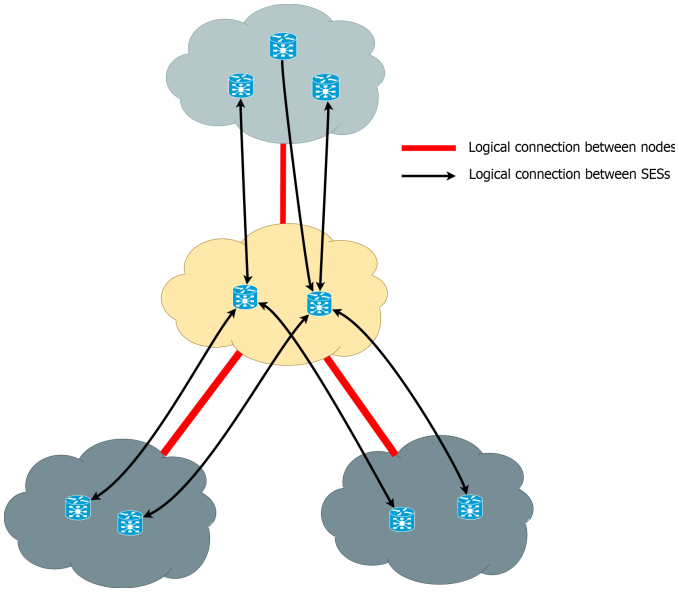

Fig. 2: Logical structure of the search engine

B. Query propagation within the network

This structure allows for a fast and deterministic way to resolve a query. When a user wants to send a query to the SEN, it is first processed in a list of keywords. For each of

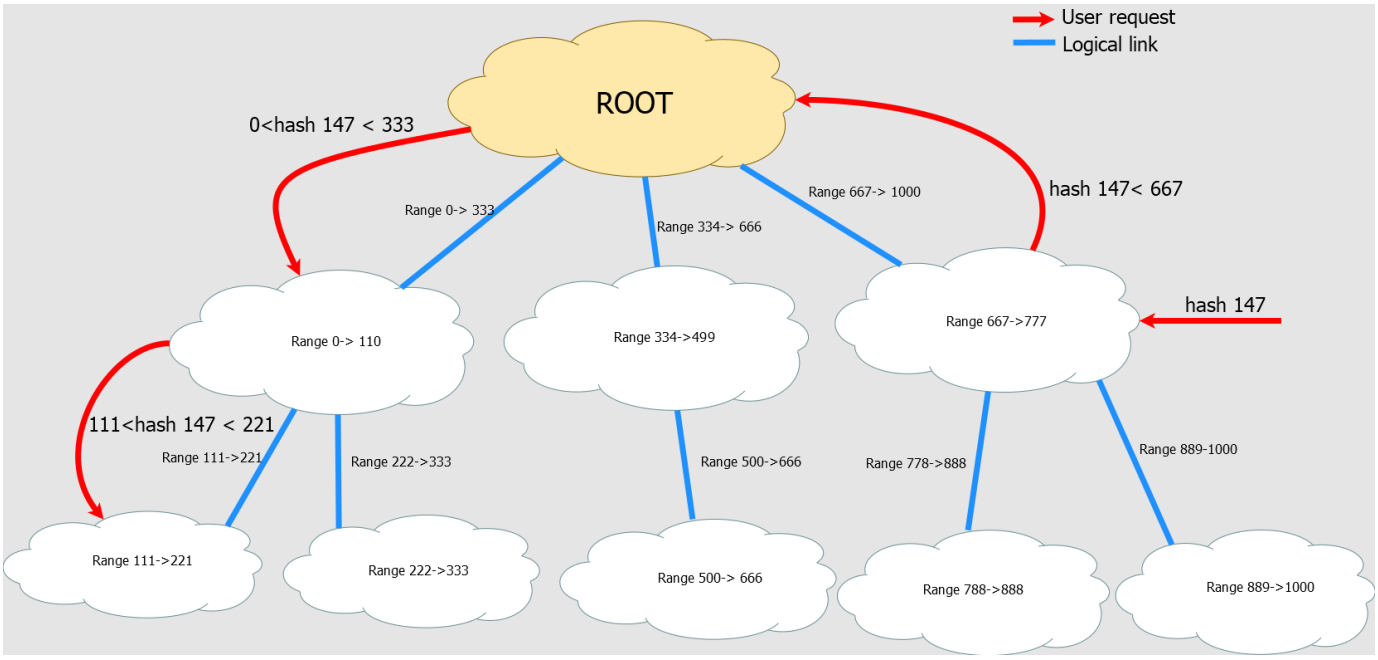

Fig. 1: Logical structure of the search engine query propagation

these keywords, a query is sent to the SEN. Each SES is a possible entry point for a user and he effectively enters the network from the closest SES physically.
Starting at the entry point, the query is then sent through the network by comparing the target hash known by the current node. First, if the target hash is within the current node's hash range, the query is over and can be returned. Else, it checks its children for the hash ranges associated to their branches. If no range matches, then the query is sent to its parent [Fig. 1].
Thus, the worst case scenario for such a search is $\mathcal{O}(\log n)$ with n the depth of the SEN tree.
This resolution scheme can be seen as an analogy of the recursive domain resolution in the DNS tree.

C. Adding a name to the SEN

In order to be functional, the SEN has to allow users to publish new names on the network.
When it wishes to do so, the producer publishes the name of the data, a set of main keywords, a set of secondary keywords and potentially an abstract.
This data will be published once in the network for each main keyword associated. For security and data management reasons, the amount of main keywords will be limited to a certain value.
The process of publication itself is very similar to the retrieval, as the name will hop from node to node until it finds the fitting node according to the same hash range algorithm [Fig. 3]

D. Handling network congestion

In order for a SEN to be highly scalable, it needs to feature a way of handling network congestion in specific parts of the network. A network congestion means that a segment of the network is under more traffic than it can handle.
Such an operation is made possible through the node structure. When a segment between 2 nodes is under heavy traffic, the parent node will increase its size and recreate connections with the concerned child. For each server of the parent pool, it submit a request to each server of the child pool and retains the faster one to answer as his link with its child [Fig. 4]. By doing this, the amount of possible connections increases and the average load on each connection lowers. A notable side effect of this operation is an increase in data replication. It is also worth noting that this operation is completely reversible meaning that the network capacity of the SE can be modulated according to trends quite easily.

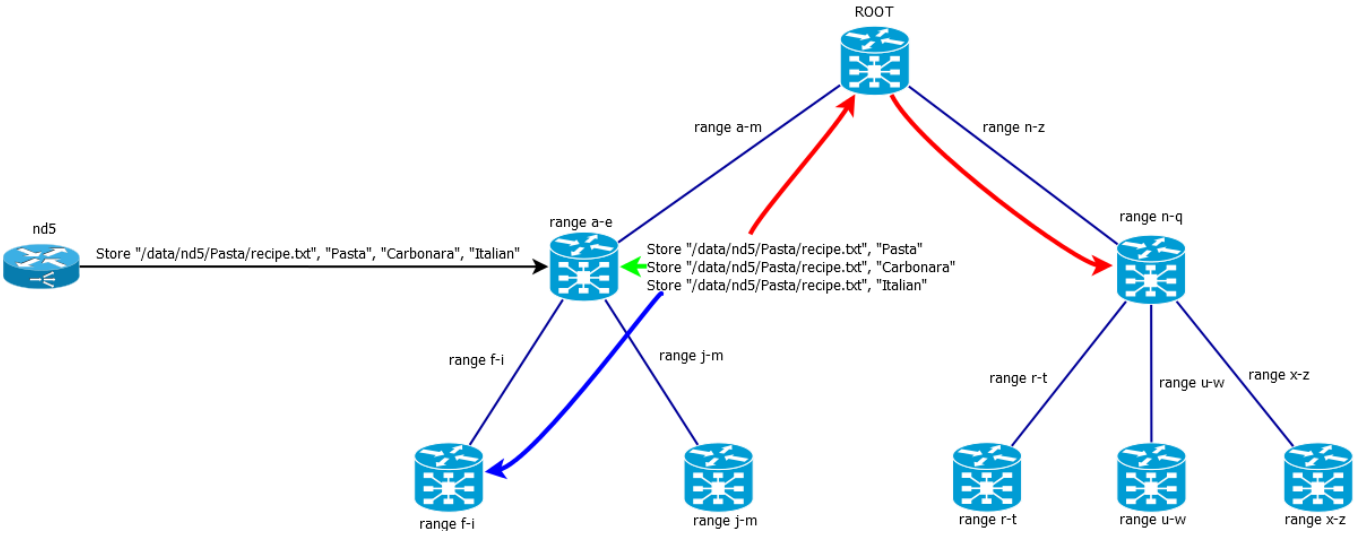

Fig. 3: Adding a name to the SEN

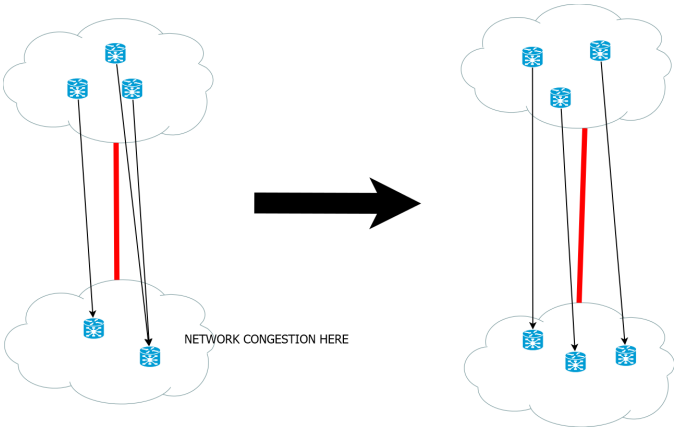

Fig. 4: Network congestion management

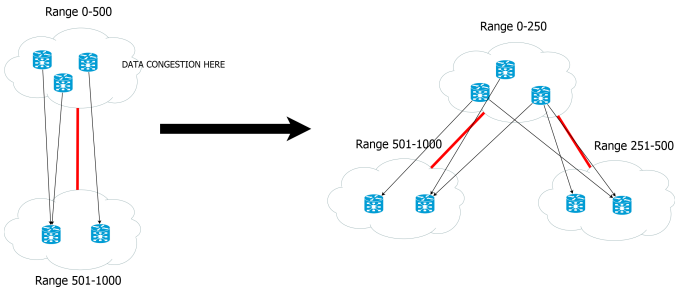

Fig. 5: Data congestion management

E. Handling data congestion

Just like the SEN has to provide a solution to network congestion, it has to be able to deal with a great amount of data. In some nodes, the amount of data to be held might be stored in a node. Because of the tree structure of the SEN and the classification of data by their keyword, the load will not be naturally balanced across the tree.
Accordingly, a solution [Fig. 5] to data congestion has to be integrated to the SEN. The suggested solution is to split the node under heavy data load in order to create a new node that will help him balance the data.

This operation allows for an overloaded node to split his data by giving a portion of it to its newly created child. In order to know which hash range has to be given to our new child, a function is applied to select the hash range resulting in the best load balance.
Notably, this operation does not change the way the rest of the tree will address this node as nodes are addressed by the range of hashes in their branch and not by the range of hashes contained within said node.

F. Root responsibilities

In this structure, the root stands out as the only node that does not hold data. Indeed, the root node holds other responsibilities : Its first and main purpose is to handle the arrival of new servers in the SEN.
Those that wishes to join the network have to ask the root to be part of the network. Then, the root will periodically assign those new servers where they are the most needed. Servers can be needed to replace servers that are willing to leave the network or servers that already left the network. Besides, servers may also be needed in order to carry out a network or data congestion handling.
Having the root handle server placement is a way to increase

the network security by preventing potential attackers from overtaking a node by deciding to add numerous corrupted servers to said node.
The root's second responsibility is a direct consequence of the first one. As a matter of fact, the root has to be aware of the network's whole status. Due to the NLSR routing protocol, this is best achieved by having the root periodically request for the network's status.
This request would be first sent directly to the root's children and recursively disseminate through the network, eventually gathering a complete rundown of its state.
As stated previously, the network congestion operations results in the assignation of new servers to a node in order to handle more traffic on a segment of the network. However such traffic can prove to be very volatile. In fact, a given keyword's popularity can fluctuate very quickly and according to a wide variety of parameters.
Thus, the SEN needs to be able to reallocate servers when they are affected to a node that is now oversized. This operation is within the responsibility of the root. With the information obtained from its steady network analysis, the root can identify which node is oversized and act accordingly by reducing the amount of servers dedicated to a node.
This operation can be seen as the reverse operation of the network congestion handling and requires once more, to update physical paths between the concerned node and its parent.

G. The hash function

In this network, the hash function is used to hash main keywords when data is added to the network and when a query is issued by a user. Hashing keywords also helps smoothing out the keyword balancing within the network. If English words were used and sorted by lexicographical order, balancing the load on the network would prove to be much harder. This would be likely to result in a deeper logical tree and thus, worse performances on average.

V. Search Engine Network implementation in NDN

In order for the SEN to work as intended, physical connections between logical links have to be established. In reality, ICN nodes are not connected to one another in any particular way. These connections then have to be exploited in order to create the logical graph structure of the SEN.[Fig. 6]

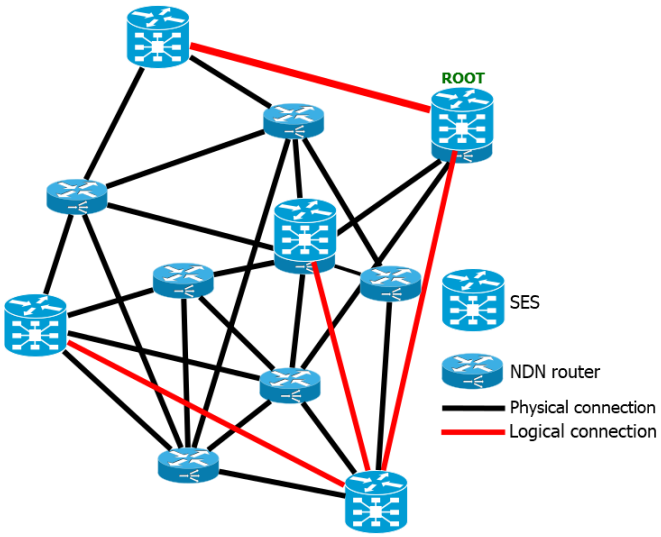

Fig. 6: Physical and logical architecture separation

A logical link between 2 nodes can then be seen as a chain of physical links between two NDN nodes[Fig. 7]

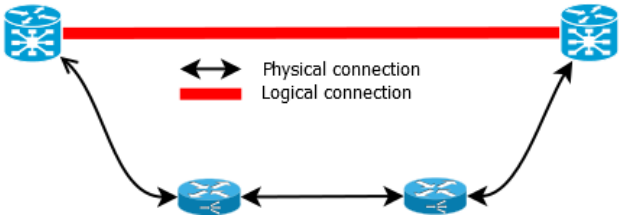

Fig. 7: Physical connection between 2 SESs

In order to establish the routing between 2 logical nodes, a LSA is issued (Link State Advertisements) for all intermediary NDN router in between the two SESs. In order to reduce the knowledge of the physical route for each router an onion routing scheme is used. In particular each intermediary NDN router only knows its following and preceding router[Fig. 8]. The onion encryption of the path of interface guarantees that a malevolent middle router or actor cannot deduce the full path between 2 nodes.

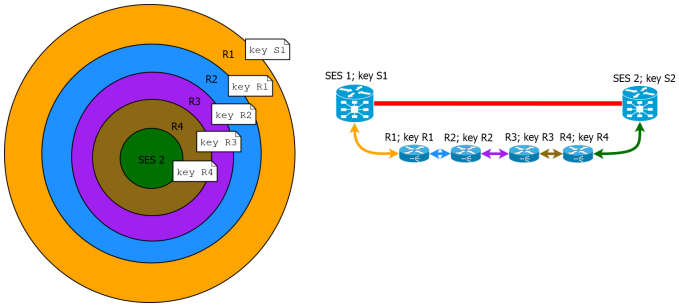

Fig. 8: Establishment of a physical connection between two SESs

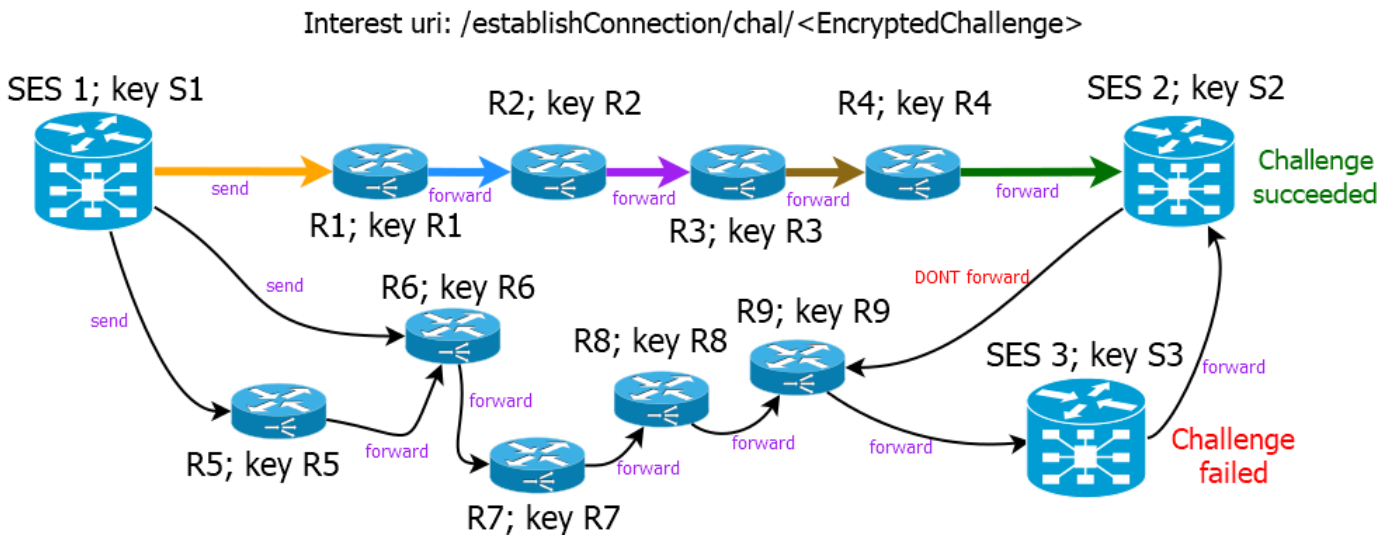

Fig. 9: Interest sent with a challenge to determine the route to the physical linked SES

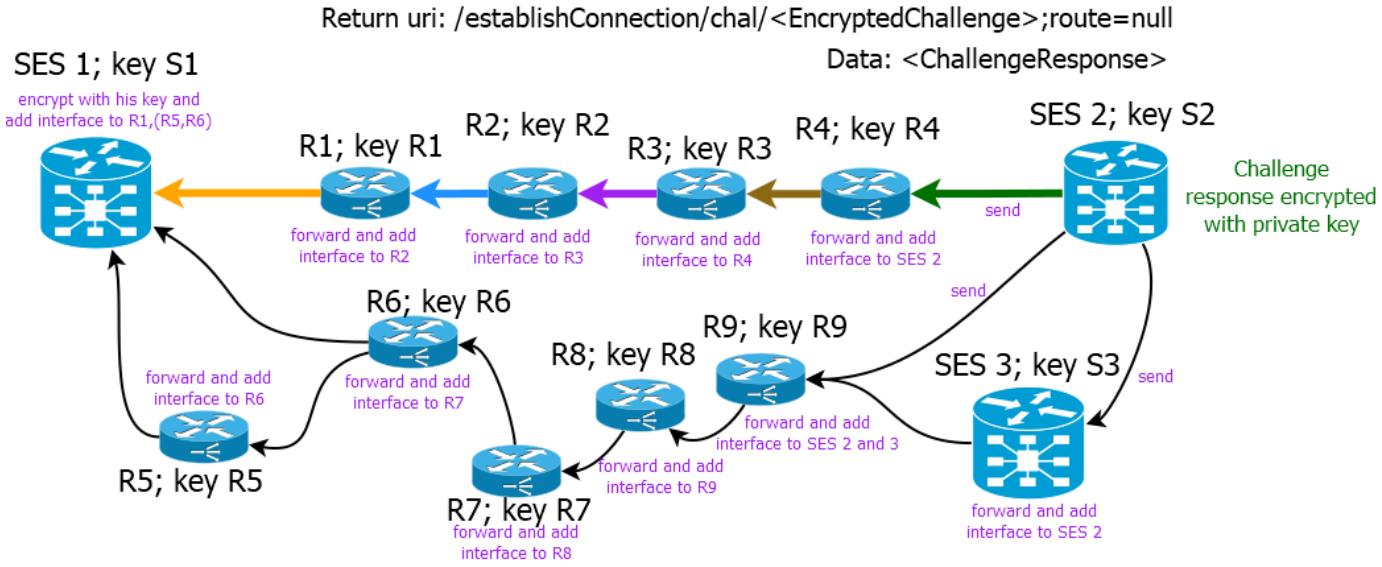

Fig. 10: Route sent back with the completed challenge

A. *Establishing the physical routing*

In order to be able to establish a physical routing, a SES needs to know the public key of the destination. This information is either already known by the SES (in case of a route update) or given by the root.

To actually establish the physical routing, the source SES will send an Interest packet on a specific name (for instance /establishConnection/chall/*value*) that will contain a challenge created with the public key of the target SES. This challenge is propagated through the NDN routers until it reaches its destination, along the way, each LSDB is updated [Fig. 9].

Once the challenge is completed, the destination SES sends a Data packet on the same specific name (for instance /establishConnection/chall/*value*) containing the answer to the cryptographic challenge.

On each hop, this name will trigger a specific behaviour that makes the current NDN router add the name of the interface on which he received this Data packet, encrypted with its private key, as a parameter of the name [Fig. 10].

Once this Data packet reaches the source SES, the name of the data will actually contain the onion route to its logical relative. This route is stored and will be used to communicate within the SEN.

A notable propriety of this operation is that the source SES might receive multiple answers to its original interest. In this

case, it will simply choose to remember only the shortest path (in number of hops) to its relative SES and keeps the others as backup.

Each SES can then use its established physical route to forward user queries in the physical network without establishing it every time, however it will be refreshed every so often.
Authenticity is ensured by the NLSR protocol. The encrypted route allows for a correct forwarding through intermediaries routers by deciphering only interface on which then will send the query.

B. Client connection

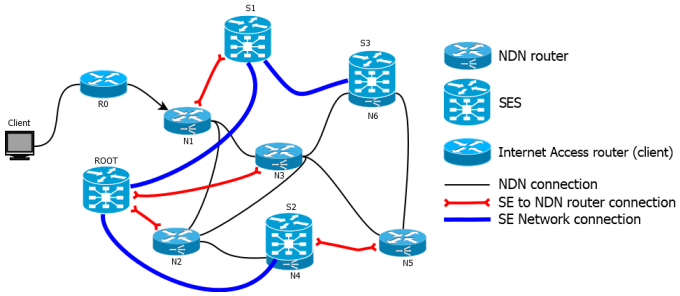

Fig. 11: Client connection

In order to be easily usable, the SEN needs to be accessible by any user. In this architecture, users can access the SEN by using a specific NDN prefix, for instance, /SearchEngine and simply addressing the closest NDN router. Then, the concerned NDN router will spread the request according to the NLSR protocol until it reaches the first SES [Fig. 11]. Once it does so, the query will navigate through the network like previously stated.

C. Description of a SES's connection

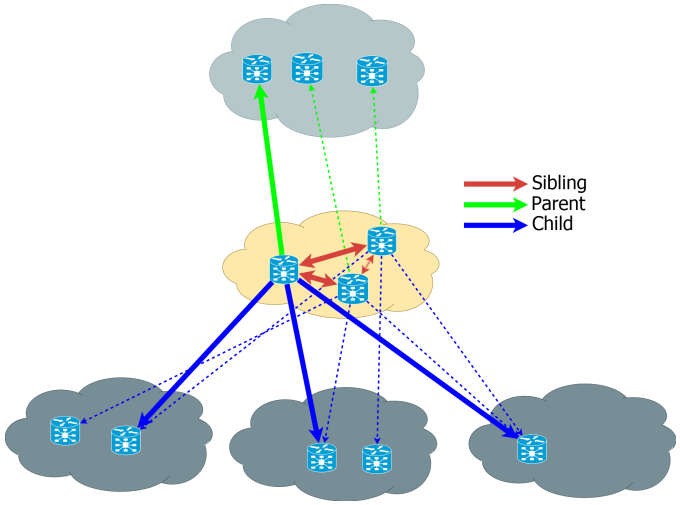

Fig. 12: Node relations

In order to achieve operations previously detailed in this paper, each SES has to be able to establish a certain list of connections.
For a given node, we can distinguish 3 types of related nodes: its siblings, its parents and its children [Fig. 12].

As shown on the [Fig. 12], for each server in the network, 2 data structures containing the SES's routing path can be defined : a list of siblings and a list of tuples <child,hashRange> where hashRange is the range of the hash contained in the branch of the logical network under the said child. On top of that, a single SES representing its father is registered.

Choosing the shortest path, in number of hops, during the physical routing part leads to an interesting property. Each SES will only use the shortest link to its relatives, this means that each servers will tend to be geographically close to its relatives. This way, the tree structure could be seen as a stack of multiple trees, with each tree being relatively compact from a geographical standpoint.

1) Handling data congestion:

When the data congestion handling event (described earlier in subsection.4.5) is triggered, the root node assigns a set of new SESs to create the new child of the overloaded node.

SESs of the overloaded node will receive from the root the list of public keys associated to the new node.
Upon reception, SESs of the overloaded node will calculate the hash range delegated to the new node. This calculation aims at splitting the data load evenly between the overloaded node and the new node.
Then, each SES in the overloaded node will establish the shortest route to each of its new child SES and hold on to the shortest one of those. This way, each server of the overloaded node has created its link to its newly created son.

SESs of the newly created node will receive from the root the list of public keys associated with their siblings and all servers contained in their father.
Upon reception, all SESs will establish the route to their siblings and the shortest one to their father.
Once all routes are established, each SES of the new node sends an Interest package to its father in order to begin the Data transfer. The father will then send the hash range delegated to the new node along with all the Data that needs to be migrated [Fig. 13].

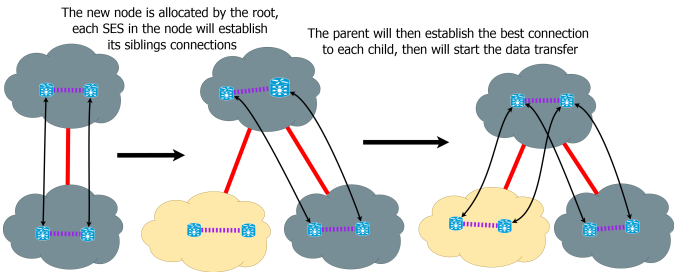

Fig. 13: Data congestion operations at the physical level

D. Handling network congestion

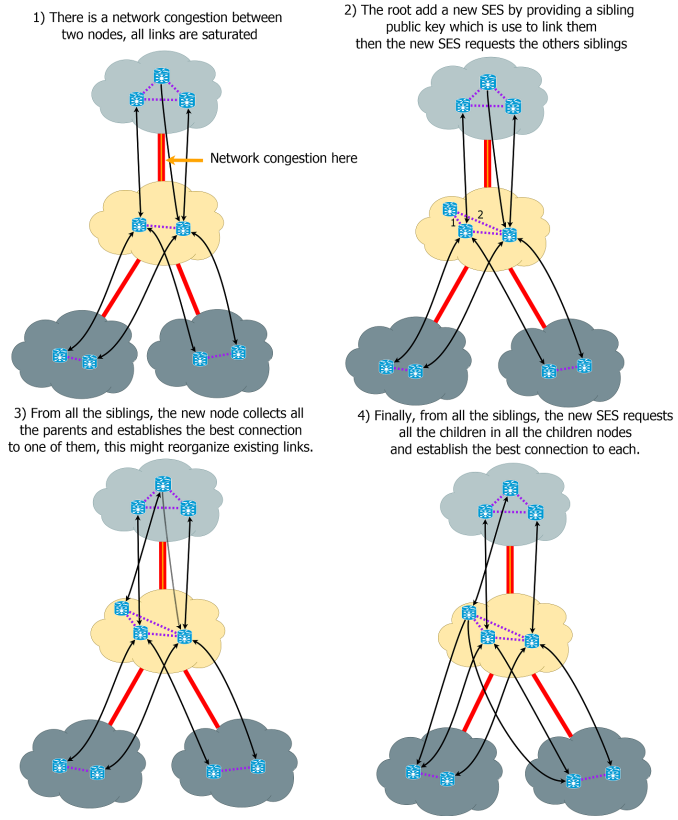

Fig. 14: Network congestion operations at the physical level

When the network congestion handling event (described earlier in subsection.4.4)) is triggered, the root node assigns one or more new servers to the child side of the overloaded segment.

When the root assigns new servers to handle network congestion, it provides the new SES with the public key of one of the SESs contained in the node.

Then the new server uses that key to establish the physical route with this SES and retrieves the list of public keys from all its siblings.
Once those routes are created, the SES retrieves all public keys of its parent node from its siblings and establishes the path to its siblings.
Once all of those operations are done, the new SES can start retrieving data from its siblings.
Finally, once all the data is retrieved it has to pull the list of its parent and child nodes public keys and sets up the shortest route to one of his parent and child node's SES.

Each SES in the parent (respectively child) of the updated node has to recreate its connections to its child (respectively parent). In order to do so, it requests the list of public keys of the updated node to its contact in the child node (respectively in the parent node). Then, it recalculates its shortest physical route with its child (respectively parent).

E. The root structure

The physical structure of the root is a cornerstone of this architecture. Indeed, the root represents the biggest concentration of information about the network and is the entity responsible for server placement in the network.
This means that the root must be trusted with important data and important decision.
Centralizing the root to any institution would be fundamentally against the principle of ICN and defeat the whole purpose of a decentralized SE.
Thus, the root needs to be trustworthy and decentralized.
One of the main technology to achieve such decentralized trust today is blockchain. Blockchain allows for the use of a decentralized shared ledger on which actions and information can be permanently written.
Such features appear particularly fitted to this use case. By using a blockchain to support the root of the SEN, we break the need of a centralized trusted institution by migrating trust in the technology behind the blockchain.

For this particular case, a private blockchain, with no cryptocurrency, could execute smart contracts in order to trigger data and network congestion handling and record those actions on the distributed ledger.
This ledger will in term make members of the root accountable for decisions and if behaviours outside of the original scope are noticed then actions can be taken to evict such nodes from the root.
However the root does not store everything in the ledger, notably the distribution of the nodes is not stored to avoid a full reconstitution of the network.

F. The choice of the hash function

The choice of the hash function (function that map any arbitrary-length data to a fixed-size value) is critical for the network viability. Indeed, several parameters are important for the stability and the performance of the network.

- The hash function should create as few collisions as possible within English words. Collision within keywords

would mean that a user would receive data associated with 2 different keywords when issuing a specific query. If the amount of collision is very low, those cases would be individually handled, for instance with salt.
- The hash function should also have good performance in order to avoid slowing down the network.
- Ideally, it should also have a consistent spread of English words across its value range. The more consistent the spread, the more balanced the logical structure and the faster the average name resolution.

In order to chose which non-cryptographic hash function (NCHF) is best suited for this application, we used the results obtained in the paper Performance of the most common non-cryptographic hash functions [12][Fig. 15]

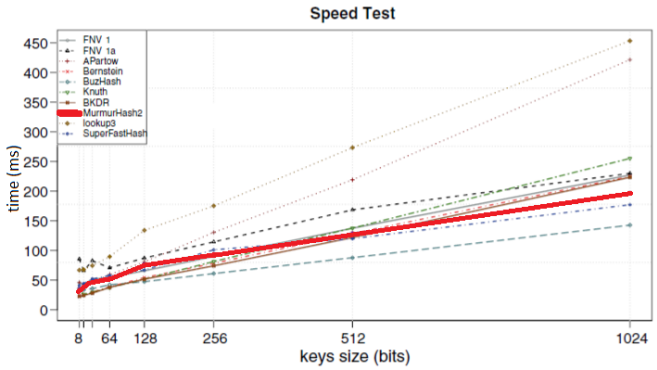

Fig. 15: Comparison of common NCHF speed [12]

In addition to those data, the list of collision in English words was taken into account.
Those data lead us to chose the MurmurHash. This NCHF is fast to compute, has a high spread among its values and only counts 6 collisions in the English dictionary.

VI. Limitations and potential improvements

Being a DHT based SE, only exact match on keywords are possible. This means that, in order to provide satisfying results, this type of SE relies on a heavy preprocessing of the natural language query. This preprocessing, while not detailed in this paper, is required for the keywords to be understood by the system. However as it can be done at any step of the process, the preprocessing can be incorporated inside the user's entry point to the network or even the user's browser.

It is important to note that this approach requires the deployment of a specific application layer on SESs which in itself requires some configuration, the same will apply for NDN router that need to be compatible with the SE routing protocol.

While caching is not explicitly discussed in this paper, it is induced by the underlying NDN protocol. This means that high level of caching for usual request is easily obtained.

When the number of names indexed in the DHT becomes very high, the implementation of bloom filters can help reducing the size of the request results. This improvement might prove to be crucial when it comes down to handling very high amounts of names.

For time and resource reasons, we were unfortunately unable to thoroughly test this architecture within the NDN stack and using the NLSR protocol.

VII. Future work

As a future work, testing the SE implementation would go as follows. First, a crude implementation of a matching physical and logical structure would allow to verify that the different operations, such as data or network congestion, can be used even when the network scale exponentially. Then, an applicative add-on to the routing protocol would be deployed on all NDN routers which would allow to establish the physical routing as depicted previously and then test it on a large scale. Finally, the last test would be to flood the network with never seen before requests testing evermore the resiliency of the network as well as the capacity of caching, this test could notably include spike in new trending keyword as well as keyword used massively in the current Internet.

VIII. Conclusion

We have presented a novel index scheme for keyword search in ICN, unlike existing approaches that are based on flooding or flat DHT, we use a tree shaped DHT structure which allows for a better network management and spread of the data. In contrast with other design, we made sure to account for potential security vulnerabilities at the core of our implementation using zero knowledge tactics and a physical/logical separation by design.
Although we cannot present extensive results, our initial tests have led us to be confident that the architecture is scalable, reliable and usable as a search engine.

Acknowledgment

This paper was written in the context of a Resarch Discovery Project, offered by TELECOM Nancy (University of Lorraine) as a part of an engineering course.
We would like to thank T.Cholez for his work, help and availability, the NDN team for its extensive documentation on the NLSR protocol and its implementation.
Thanks to D.Kondo for his insight on the project.
Thanks to TELECOM Nancy (University of Lorraine) for making this project possible.
We would like to thank the anonymous referees for their valuable comments.

REFERENCES

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies, CoNEXT 09, (New York, NY, USA), p. 112, Association for Computing Machinery, 2009.

[2] A. Hoque, S. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, "Nlsr: Named-data link state routing protocol," pp. 15–20, 08 2013.

[3] Y. T. Kévin Pognart and H. Tode, "Keyword-based breadcrumbs: A scalable keyword-based search feature inbreadcrumbs-based content-oriented network," AFIN 2014, The Sixth International Conference on Advances in Future Internet, 2014.

[4] Y. Mao, B. Sheng, and M. C. Chuah, "Scalable keyword-based data retrievals in future content-centric networks," Proceedings of the IEEE Conference on Mobile and Ad Hoc Sensor Networks (MSN), 12 2012.

[5] P. Daras, T. Semertzidis, L. Makris, and M. Strintzis, "Similarity content search in content centric networks," pp. 775–778, 10 2010.

[6] Youngin Bae, J. Kim, M. Jang, and B. Lee, "A prefix-based smart search in content-centric networking," pp. 104–105, Jan 2012.

[7] S. S. Adhatarao, J. Chen, M. Arumaithurai, X. Fu, and K. K. Ramakrishnan, "Orice: an architecture for object resolution services in information-centric environment," pp. 1–6, April 2015.

[8] J. Chen, H. Xu, S. Penugonde, Y. Zhang, and D. Raychaudhuri, "Exploiting icn for efficient content dissemination in cdns," in 2016 Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), pp. 14–19, 2016.

[9] J. Li, B. Loo, J. Hellerstein, M. Kaashoek, D. Karger, and R. Morris, "On the feasibility of peer-to-peer web indexing and search," 10 2003.

[10] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," IEEE Transactions on Networking, vol. 11, 02 2003.

[11] P. Reynolds and A. Vahdat, "Efficient peer-to-peer keyword searching," pp. 21–40, 2003.

[12] C. Estébanez, Y. Sáez, G. Recio, and P. Isasi, "Performance of the most common non-cryptographic hash functions," Software: Practice and Experience, vol. 44, 06 2014.