RESEARCHERID
THOMSON REUTERS

[Singh, 6(8): August 2019]
DOI- 10.5281/zenodo.3362262

ISSN 2348 – 8034
Impact Factor- 5.070

GLOBAL JOURNAL OF ENGINEERING SCIENCE AND RESEARCHES

BLOOM FILTER APPLICATIONS IN NAMED DATA NETWORK: A COMPREHENSIVE REVIEW

Arshdeep Singh[*1] & Gurjit Singh Bhathal[2]

[*1,2]Department of Computer Science and Engineering, Punjabi University Patiala

ABSTRACT

Internet was designed to provide source to destination communication and it had shown good resilience over the time. In today's world, evolution of content-centric applications has led to failure of internet as it was designed to interact over a pre-established communication channel. Named Data Networking (NDN) came up as a solution to this which works with content specific requests and returns the contents to its requester regardless of its location. It uses different data structures where it stores information about various content and perform a search among these when a request encounters. As the requests and content size increases it leads to increase in complexity of query as well as memory consumption. So main challenge is to find the solution to efficiently store, query and retrieve large number of entries related to content names in real time interactions, Probabilistic Data Structures (PDS) came up as the solution for this as they are suitable for large data processing, approximate predications, fast retrieval and storing unstructured data, thus improving space and search complexity in Big data processing. Bloom filter is a PDS which is used for approximate membership query. Many variants of BF have been already successfully employed in various domains in NDN like, scalable forwarding and routing, caching and security. In this paper, we try to discuss the study of applications of BF in different NDN domains in depth. We conclude our survey by identifying a set of open challenges in NDN which should be addressed by using PDS.

Keywords*: Named Data Networking(NDN), Probabilistic Data Structures (PDS), Bloom Filters(BF)*

I. INTRODUCTION

A. Named Data Networking(NDN)

In early 90's, internet was designed to provide point to point communication among two entities (hosts, computers etc.). After the introduction of TCP/IP [1], it became easy to share multimedia like pictures, videos etc. in form of packet through the means of packet switching.

Internet has shown good resilience/flexibility over the time. In modern times with the evolution of many information centric applications like Facebook, YouTube, Twitter, Amazon etc. has led to the failure of internet as it was not designed to support the content distribution model. In today's world most of the applications focus on which data is needed, not on the data location. Also internet does not have in-built support for security and mobility.

Above mentioned reasons encouraged researchers to find an alternative to internet, that will support information-centric communications. So, NDN rise up as a candidate that deals with applications generated variables and getting the user requested content irrespective to its host. [2]

NDN design based on basic design principles of Internet. It uses IP services like DNS and inter-domain routing. Data packets used in NDN are independent and self-contained. It uses unique content names to communicate instead of source and destination addresses. These features allow the NDN routers to cache the packets (in-network caching) to fulfill the future requests of consumers in-turn supporting the consumer mobility.

ISSN 2348 – 8034
Impact Factor- 5.070

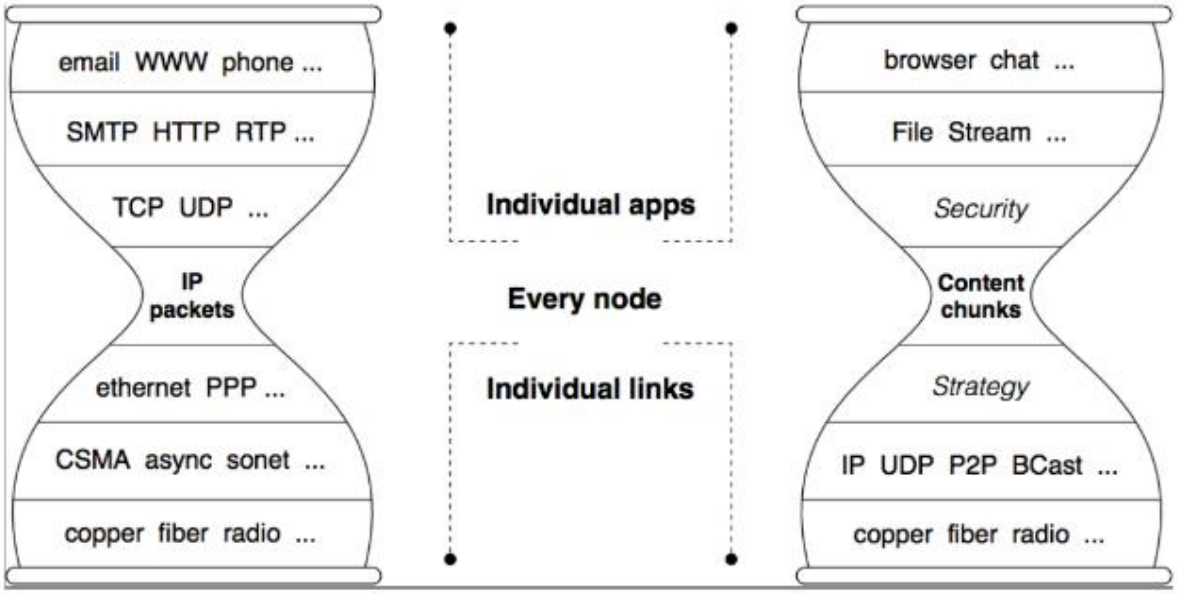

Figure 1. Hourglass architecture of internet and NDN. [2]

As shown in Fig. 1, Internet and NDN uses same layered hourglass architecture with difference in functionality of the different layers. Internet OSI model only has IP in network layer which is difficult to modify with changing needs. NDNs network layer must support scalability, security, resilience and efficiency. As shown in Fig. 1, NDN architecture has two new layers added Security and Strategy. Security layer is to provide the security for every content shared over the NDN. Strategy layer is mainly used for decision making process for routing of the incoming data packets. [3,4]

NDN has resolved many of the issues that were there with the use of IP packets in internet given as:
Address Space exhaustion: As in IP we have fixed number bytes to specify the address, but in NDN we specify the packets using names of variable length.

Address management: No need to acquire the addresses in NDN, as it was in IP to communicate.
Congestion Control: NDN sends single response packet for multiple requests that reduces the network congestion, which was not the case in IP.

Security: In IP we need to provide secure communication channel to ensure secure communication, while NDN provides the in-built security features as discussed above.

NDN communication takes place in form Interest packet – I_pkt (initiated by consumer) and data packet – D_pkt (returned by the provider or router) as shown in Fig. 2. Both I_pkt and D_pkt have content names embedded in them to maintain uniqueness. Each NDN router maintains following data structures [5]:

- **Content Store(CS)** – Each router caches the copy of all D_pkt's passing through it in CS to fulfill the future consumer requests to reduce the response time and save bandwidth.
- **Pending Interest Table(PIT)** – If a D_pkt is not found in CS, then an entry is made in PIT for the I_pkt along with the interface requesting it. PIT maintains this entry until the D_pkt is arrived or lifetime of I_pkt is expired.
- **Forwarding Information Base(FIB)** – FIB is used to send the I_pkt upstream to web requesting the corresponding D_pkt for it.

ISSN 2348 – 8034
Impact Factor- 5.070

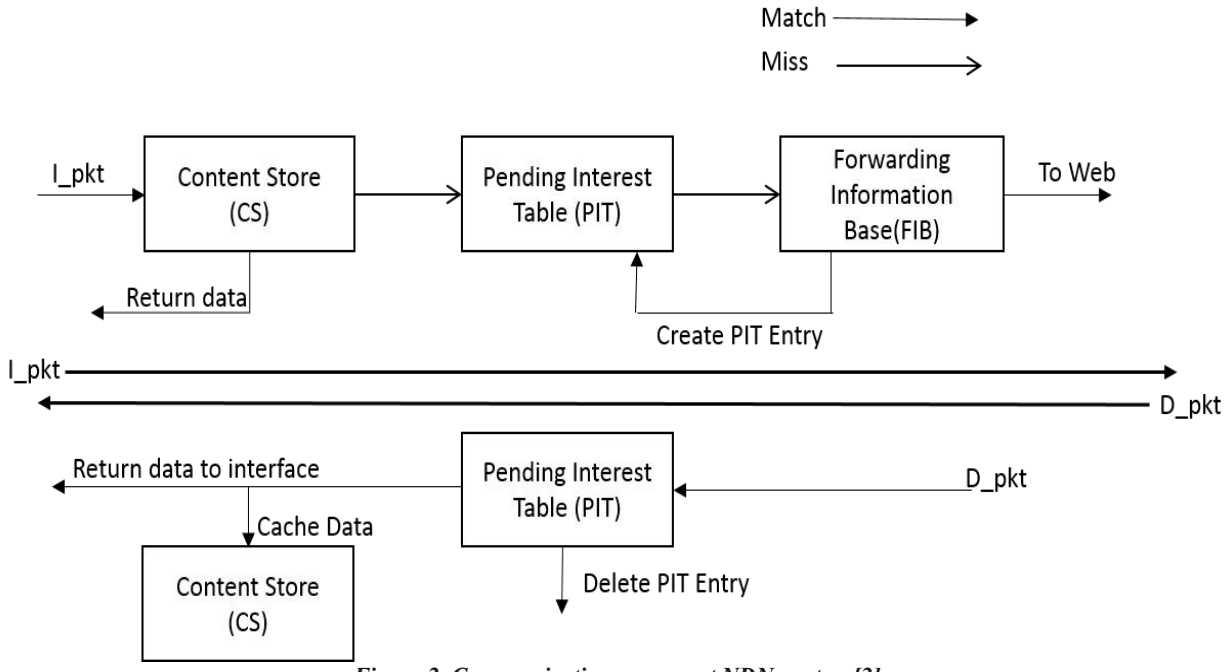

Figure 2. Communication process at NDN router. [2]

NDN's functional characteristics that includes *Routing*, *Caching*, *Forwarding*, *Security* and *Mobility* to support various user requirements in terms of forwarding the particular information in a secure manner taking care of consumer and provider mobility. Below we will discuss major techniques and proposed in NDN for above features in brief:

1. Routing – In NDN, routing mainly works for checking the availability of the routes and setting the topologies and policies. It also updates the forwarding table which in turn will be used by Forwarding plane to forward the information onto optimal routes. Routing algorithms used for internet can also be used for NDN with slight modification like changing the message types (I_pkt/ D_pkt) and adding multipath forwarding. NDN routing table consumes more memory as it stores Content Name (CN). Performance of NDN routing algorithm depends on parameters like *CPU Utilization, PIT Count, memory consumption, network utilization* and *time to completion*. Major routing protocols used in NDN are mainly intra-domain routing protocols like *OSPF, 2-layered* and *link state routing protocols*. [3, 4]

2. Caching – In NDN, every router caches the copy of every D_pkt passed through it, also called in-network caching. It caches the data in order to fulfil the future requests for the same content in minimal time and reducing the provider's overhead by fulfilling the multiple requests for that data form in-network copies stored at routers. Cache performance depends on four parameters: *hit ratio, content retrieval delay, average no of hops traversed* and *dissemination speed*. [3, 4]
Cache decision policy depends on two schemes given as:

- *Cache Placement*: It decides on whether a D_pkt to be cached or not. It again depends on: *Content popularity, Cache partitioning* and *Selfishness*.
- *Cache Replacement*: It decides on which D_pkt has to be replaced when a new D_pkt has to be cached. It depends on *Content popularity* and *Content prioritization*.

3. Forwarding: Forwarding plane in NDN takes care of forwarding the I_pkt and D_pkt to optimal routes based on the performance and status of various routes. Forwarding can be classified as: *Scalable Forwarding* and *Forwarding Strategies*. [3, 4]

- *Scalable forwarding*: It supports stateful and intelligent forwarding. NDN forwards data based on content names of variable length and need to read/update the FIB for different I_pkt's and D_pkt's. Main challenge in Scalable forwarding is to provide fast CN lookup on low memory cost. It mainly works on 2 data structures PIT and FIB, data updates and retrievals happen on PIT and FIB to forward the packets to desired consumer or provider.
- *Forwarding Strategies*: It mainly deals with the decision making on how to efficiently use various forwarding options and choose best interface to forward the I_pkt. Deciding factors for a strategy are: *Working path*, *Content based selection* and *efficient interface probing*. Different forwarding strategies are *Adaptive forwarding, Congestion control, Blind forwarding* and *Aware forwarding*.

4. Security: Data security is most important challenge of any information centric network. NDN ensures the security at packet level through content encryption. It supports both symmetric and asymmetric key encryption. Producer encrypts the data using consumer's public key which can only be decrypted using private key (available with consumer only). Access control is needed to distinguish the legal and illegal users. Trust management and Privacy of the content are other features that NDN provides. [4, 7]

5. Mobility: In internet environment, every device must have to attain a particular IP address before sending or receiving the data. So any mobile device cannot send or receive the data until it acquires a new IP address. While NDN supports CNs instead of the IP addresses, that enables the mobile users to access data without the need of acquiring IP addresses repeatedly. Special feature of NDN is that two devices can communicate without the need of internet if they are in range of each other physically. Mobility techniques can be classified into: Provider's mobility (Mapping-based & Locator free) and Hand-off management. [4, 6]

Applications of NDN [8] are spread across wide variety of domains like Sensor Networks, Tactical Networks, Educational Services (like setting up conference rooms, ad-hoc communication during meetings or lectures etc.), Commercial applications (like electronic payments, file hosting services, data collection form mobile devices etc.), Multimedia applications (like video conferencing, tele-teaching, tele-marketing etc.), Vehicular Communication systems, enterprise networking, instant messaging, gaming and entertainment etc.

As we said earlier that NDN uses three data structures CS, PIT and FIB for storing the various information related to the content packets at routers. As the number of requests increases, content packets increase which will lead to memory cost and higher complexity of content search query. One of the major challenge in NDN is to provide efficient storage and search mechanism to manage the ever growing data in NDN network. So as a result, *Probabilistic Data Structures* (PDS) [9] rise up as suitable candidate to solve above challenge. PDS's provide time and memory efficient storage and search queries to handle the large data sets like CS, PIT & FIB in above scenario. BF is one PDS that supports membership queries and best suitable to store and retrieve the content packets efficiently in NDN data structures.

B. Bloom Filters

A Bloom filter [10] is space efficient PDS that is used to check the presence of an item in a data set. E.g. checking the availability of a new username is a set membership problem, where data set is all the registered user names. Only price we have to pay for the efficiency of BF is the probabilistic nature of BFs, there might be some False Positive results. False positive means that sometimes it will say that the element is member of the set but in reality it's not.

Working of a BF:
Let S be a unique set of n elements, S = {X1, X2, …., Xn}. As depicted in Fig 3, BF is an m bit array initially all the elements set to 0. When an element Xi arrives then it is mapped onto the BF using k hash functions {H1, H2, …., Hk} calculated against Xi such as {H1(Xi), H2(Xi)…, Hk(Xi)} bits will be set to 1 in the BF.

ISSN 2348 – 8034
Impact Factor- 5.070

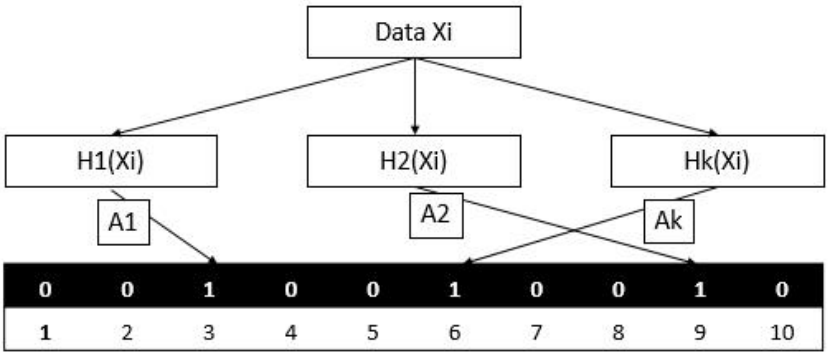

Figure 3. Insertion of an element in BF.

To check the membership of any member Yi, we first calculate the k hash function against Yi i.e. {H1(Yi), H2(Yi)…, Hk(Yi)}. All these bits then checked in BF, if any single bit is set to zero then we can say that Yi is definitely not present in the set. If all the bits for Yi are set to 1 then we say that Yi might be present in the set as these bits might have been set to 1 by some other elements, which is termed as False Positivity in BF i.e. BF can identify that some element is present in the set while it is not actually there. False positives in PDS are acceptable with some predefined threshold. Some of the BF like Counting BF (CBF) that supports deletion also induces False negative results i.e. it identifies that element is not present while it actually present in the set.

Space Efficiency of BF:
All the methods such as hash map, simple array or linked list when used for storing the large list of items for the purpose of set membership, they require to store the item itself which is not very memory efficient way.
BFs do not store the data item at all, they only use a bit array using hash function allowing hash collision, which makes BFs a memory efficient way for storing large set of items for purpose of set membership.

Different kinds of BF:
Many variants of the BFs have been proposed based on application areas, which can be classified as:

- Basic BFs with fixed length. They have constant false positive rate and mainly suitable for static data sets. Deletion operation is not allowed as deletion of single member might cause the deletion of another as well by setting bit set by another to 0. E.g. Distance sensitive BFs [12], Weighted BF [13] etc.
- SomeBFs use counter instead of a bit at every position of array. With insertion of every element, we increment the counter. Deletion is supported as we can simply decrement the specific bit for deleted element. It increases the memory overhead in order to store the counter at each array position. E.g. variable-increment CBF [17], L-CBF [18] etc.
- Some BFs are designed to be adaptive to handle the dynamic data sets, it keeps adding new BFs if the incoming data is increasing. Main drawback is that membership query becomes complex as the size grows. Initial size of filter must be chosen carefully as smaller will lead to more computational overhead while larger will tend to memory wastage. E.g. Dynamic BF [14], Scalable BF [15].
- To handle real time network applications, BFs tend to reside in main memory to provide faster search. So in order to deal with limited main memory, old data needs to be deleted from BF in order to accommodate new data. E.g. Stable BF, A^2 buffering [16] etc.

C. Applications of BF:
Initially, BF was used to search the words in a dictionary. With the time and advancement in technology application of BFs start spreading to wide range of domains like networking (like web caching, security enhancement, routing and forwarding, network monitoring and content delivering etc.), databases (classification, querying and searching, content synchronization, duplicate detection, privacy preservation etc.), spatial applications to represent areas with diverse properties, navigation scenarios in mobile computing, accounting, security, load balancing, monitoring etc.

Big IT giants like Quora, Facebook, Oracle, Bitly, Apache HBase etc. also use BFs in various organizational application like Quora is using BFs filter the story recommendations based on user interest. [11]

BF has many applications in various areas of NDN like Caching, Forwarding, Security etc. Some examples being Mapping BF is used for PIT lookups in Scalable forwarding, NameFilter (2 staged BF) for longest fast name look up in FIB, Prefix BF used for GPU based FIB lookups etc. Detailed analysis of various BFs used in NDN will be done in next section.

II. APPLICATIONS OF BF IN NDN

In this survey, we try to concentrate on the various types of BFs used in different application areas of NDN in detail. In this section, we will mention BFs used in NDN Routing, Forwarding, Caching and Security.

A. Routing:

1. BF based Routing(BFR): BFR [19] is a BF based routing mechanism introduced to overcome the challenges introduced by exiting techniques like Flooding, Shortest path routing. Flooding wastes the channel bandwidth by forwarding the I_pkt to all the interfaces while Shortest path routing demands prior knowledge of topology and locations inducing extra overheads. In BFR, servers represent their contents using Simple BFs that allows the compact representation of the sets. These BFs containing content information are sent to various routers and consumers using a special I_pkt called Content Advertisement Interest Packet(CAI), so no router will return any D_pkt for such I_pkt. NDN takes care of deleting the duplicate CAI packets. Each routers stores CAI packets in PIT table, and when any content request comes it checks which CAI BF has this information and then adds that interface (from which that CAI is received) to its FIB as next interface. BFR uses multicast of the I_pkt to all the interfaces mentioned in its FIB to cover all the paths between consumers and origins to overcome the challenges in case of topology changes.

As a result, BFR provides the lesser communication cost and average round-trip delay as compared to flooding and shortest path routing. It has also overcome the challenge of robustness to topology changes as in shortest path routing.

2. COBRA (Content-drivenBF based Intra domain routing algorithm): COBRA [20] is also a BFR protocol that uses Stable BFs(SBF) to store the information about the content names, forwarding interfaces etc. at every NDN node. Cobra provides characteristics like Simplicity (less computational overhead), Efficiency (fully distributed) and Autonomy (No IP based routing). Stable BF is a CBF. In Cobra when a new D_pkt arrives at a node, we insert it into SBF by decrementing P random cells by 1 and k cells that match hash functions to max values, this increment and decrement on every insertion makes the 0's and 1's in a correct proportion. Also it provides lesser false positive rate in limited memory space. Traditional FIB is replaced with the SBFs. Cobra works in four phases: 1) Content driven construction of FIB: it stores the traces of every retrieved D_pkt in all the intermediate nodes. 2) Interface ranking: all the outgoing interfaces are assigned ranking in order to reduce bandwidth wastage in case of flooding. 3) Retransmission handling: stale information is replaced with the newer information available through increment and decrement of counters. 4) Link failure and recovery handling: it provides handling in case of any link failure and recovers from such event through link recovery.

As a result, Cobra has reduced the network overhead by a large extent as compared to flooding based routing and providing the same hit distance as of Dijkstra's algorithm.

3. Pull Based BFR: Pull based BFR [21] is an improved version of the BFR (also called Push based BFR) discussed earlier. In Push based BFR servers advertise all their content to the routers using Content Advertisement Interest (CAI) messages which will result in wastage of network bandwidth as well as router's and consumer's memory to store such large info on their FIB. To avoid these problems, In Pull based BFR, Servers advertise only the demanded objects rather than advertising the full content. Clients generate Content Advertisement Request (CAR) messages which encapsulates the BF containing the names of demanded content, these CAR messages then forwarded to

routers and servers. On the route from client to server, if any router receives 2 CAR messages then it combines both CAR messages into one CAR by performing a union of the BF's contained in respective CAR's. Resulting CAR will contain the demanded objects from various clients and when it reaches at the server, server first takes out the BF and checks if it has the requested content, then it creates one BF containing the advertising information of the requested content, encapsulates that BF in a CAI message and sends back to the path from where it is received. On the way back to client, all the routers will store CAI on their respective FIB's and client also stores CAI on its FIB for future use. So this way CAI messages are of smaller size as compared to original BFR and consumes less bandwidth and memory.

As a result, it provides us many advantages like 1) less bandwidth of network, 2) less memory requirements on clients, 3) BF based aggregation of CAR messages and 4) better average round trip delay, as compared to earlier routing protocols.

B. Forwarding:

1. Mapping BF: Mapping BF(MBF) [22] is used to enhance the PIT implementation called MaPIT, under Scalable forwarding in NDN. MBF is an enhanced BF that supports querying and mapping of set elements in memory and reduces the on-chip memory consumption. It has 2 components, Index Table and Packet Store. Index table is stored on main memory and holds the offset address of all the packets stored on Packet store that resides in the secondary memory. Index table has 2 data structures: Simple BF and a Mapping Array(MA) both being bit arrays of finite length, where BF is to check the availability of the elements and MA is used to give offset address of the particular packet in Packet store.

MaPIT uses MBF as explained above and a CBF to support the addition or deletion of the packets from packet store for incoming/ outgoing Interest/Data packets. If an *I_pkt* has arrived MaPIT, it is first checked in BF, if not exists then it is forwarded to FIB and entry will be added to BF and CBF. If I_pkt is present in BF, then its entry will be updated and it is forwarded to FIB. Similarly, for incoming D_pkt, BF is queried to check the existence and if it is not present then it is blocked. If an entry for D_pkt is present in BF then data will be forwarded to all the interfaces stored in packet entry in Packet store, and references will be deleted from BF and CBF.

As a result, MaPIT has reduced on chip memory consumption to 2.097MB and false positivity of 1% for almost 2 million records.

2. NameFilter (2-staged BF): NameFilter [23] is a fast name lookup method used in Forwarding process that uses 2 BFs (One memory access BF and Merged BF) to provide *high memory efficiency* and *fast querying capacity*. This filter works on the concept of bloom-filter based matching of the content names. First stage in NameFilter, maps the content names to respective BFs based upon the length and which in turn is used to find the longest prefix match of a name. Second stage divides the names based on their next port, and destination port for a particular longest prefix match is returned by querying the second stage BFs. One memory access BFs (Introduced by Yan Qian et al. [7], which combines the results of k hash function into a single word) are used at first stage to reduce memory access time of the query from k (number of hash functions) to 1. Merged BF is used at second stage that merges the N number of BFs that are bound to N destination ports to form a single bit string by combining all values at 0^{th} position of all the BFs resulting in a single BF with bit string stored at each positon instead of single bits, which reduces the memory access time by a factor of N.

As a result, it provides 37 million searches per second in name prefix table of 10M entries at a very low memory cost of 234.27MB. NameFilter outperforms the Character Trie (12 times speedup and 77% memory savings), Bloom hash and lookup using simple BFs.

3. Adaptive Prefix BF: Name Lookup Adaptive Prefix BF (NLAPB) [24] is name lookup engine for longest prefix mapping in NDN FIB. It uses a combination of 2 data structures one is BF and second is Trie. Motivation for making this engine comes mainly from 2 challenges, first is false positive results in BFs engines and memory insufficiency of Trie (Tree based data structures) based engines. So NLAPB uses the combination of both, it divides

the name into 2 parts, B-Prefix and T-Suffix. B-Prefix is of fixed length and matched using CBFs while T-Suffix is of variable length (Smaller than its name) and is processed using Trie. When any prefix is to be searched it is first divided into 2 parts B-Prefix and T-Suffix. At First Stage B-Prefix is searched in CBF to check if any match exists. Second stage is to process the remaining variable path using Trie accessible through the location stored in the hash table associated to CBF. After a longest prefix is matched the request is forwarded to desired next hop. Multiple names can have same B-Prefix, that means CBF will have lesser entries and that will reduce the false positive rate of CBF. T-Suffix of shorter length can be easily processed using Trie, resulting in reduced memory utilization.
As a result, NLAPB is designed to address the issues raised by BFs and trie. It has successfully come out as a solution by reducing the main limitation of both the data structures and producing an optimal name lookup engine.

4. A new BF based architecture for FIB Lookup using 2-phase BF: It is a new FIB lookup architecture [25] that works on the concept of Longest Prefix Trie (LPT) which is an efficient longest prefix matching data structure, but due to its large memory requirements, it has to reside on off-chip memory, so access time is 10-12 times slower than on chip memory accesses. So this architecture aims to use space efficient data structures i.e. BFs to implement the LPT. As BF's require lesser memory space, they can reside on chip which will decrease the access times and in turn increase the FIB lookup speed. In this, they used 2-phase BF strategy i.e. 2 BFs, level BF(l-BF) is used to store the level information of prefixes and port BF(p-BF) is used to store the outgoing face information for respective Longest Prefix Match (LPM). When a new request comes it is first searched in the l-BF, all the prefixes in the request are matched one by one till l-BF returns negative result (means it is last matched level). Once we get the last level then p-BF is searched for the outgoing face of that particular LPM which in turn returns if any such information exists. So this way almost all the request for FIB lookup can be addressed through on-chip memory accesses by avoiding the access to off-chip hash table. Off-chip hash table comes into picture only when p-BF returns the indeterminable result, in that case off-chip hash table is used to obtain the outgoing face information.

As a result, it shows that 99.99% of the requests are satisfied by querying only on chip BFs in turn drastically improving the FIB lookup speed for incoming requests.

C. Caching:

1. Cache Sharing using BFs: In NDN every router has its own cache to store the frequently requested contents to reduce the content delivery time, but as the content increases, cache replacement replaces old content leading to lesser content diversity. So in this proposal, BFs are used to share the cache among various routers in order to increase the content diversity. Every router maintains a BF containing cached content information, which is shared among all the neighboring routers in form of a Summary Packet (new packet containing the summary BF). So when a router is aware of the cached content of its neighboring router then it will not cache the content stored in the cache of its neighboring router. For example, suppose a content C1 reached router R1, then R1 will check the cache summaries of all its neighboring routers R2, R3 etc. & if C1 is present in R2 then C1 will not be cached in R1 as it can be retrieved from R2 when needed. A router with n faces maintains (n+1) BFs, i.e. one for every face and one to store its own cache. As the size increases storing (n+1) BFs on a router will also leads to more memory consumption, so to reduce this problem some existing BFs (that supports association queries) can be used to represent the summary store, some examples are Stable BF, Shifting BF etc. Use of stable BFs drastically reduced the query time as well as memory consumption for $n \geq 3$.

As a result, with sharing of cache summaries, cache hit ratio has been increased as well as the content diversity (as a router does not need to store a content which can be retrieved from its neighbor). [26]

2. Line-speed and accurate on-line popularity monitoring using BF: Cache replacement policies based on content popularity needs some efficient ways to collect the content popularity in order to replace the least popular contents with the more popular one's. So in this proposal, author proposed a BF based popularity monitoring algorithm that provides popular content with high speed and low memory consumption at the router. This scheme uses k BFs each representing the range of popularities i.e. BF_1[1 - N], BF_2[N+1 - 2N] and so on. All the contents passed through the routers are assigned to BF's based on their popularity. Example, if C has come for first time, its popularity p = 0, then its popularity will be increased by 1 and inserted to BF_1. If p = N, then p will be set to N+1

and C will be inserted to BF_2, it can also reside on BF_1. So at any point in time, K^{th} BF, BF_k will have the most popular contents and that are used for cache replacement policy. Two schemes are proposed to obtain content popularity, fixed window and sliding window. In fixed window, we update BF for a defined time slot say $[t_0 – t_1]$, at t_0 all the BFs are set to 0 and at t_1, most popular contents taken from BF_k and saved for further use. This scheme has one limitation that if content is accessed at the junction of t_0& t_1, then its popularity will be divided. This issue can be addressed in sliding window scheme, in which we divide the time line into slots smaller than the window, this window slowly moves forward by leaving the slot at tail and covering the slot at front. At the end of one slot the content popularity is combined into a single table called absorbed table, summing up the popularity of repeated content. This scheme will present more timely content popularity.

As a result, this method provides the results at a high speed of 16.74 Gbps at very less memory of 32mb. [27]

D. Security:

1. SP-NDN: SP-NDN [28] is a Security and Privacy preserving NDN architecture that uses traditional BFs to provide security and privacy of I_pkt over the network. Content names along with secret keys are stored in the compressed form into a BF to securely transmit it over the network. When any I_pkt is arrived at the router then it is queried within the BF to check whether a matching D_pkt exists or not, if matched then D_pkt is sent back to consumer on the same path on which I_pkt is received. Else the D_pkt is flooded forwarded to all neighboring routers. If no matching D_pkt is matched till end node, then I_pkt is discarded. To further enhance the security SP-NDN uses the Content Dependent Key Management Tree (CDKT), that stores all the content and transmission keys in the form of tree to reduce the transmission overhead. Leaves containing the tails of prefixes, a user can request content from multiple leaves, which in turn can be addressed by a single parent which was not the case in traditional tree. SP-NDN uses the multicast encryption to ensure the privacy and security of the secret keys shared between the client and servers over the network. Further in SP-NDN, they used multiple BFs to reduce the probability of false positive results.

As a result, SP-NDN provides a BF based secure and privacy preserving architecture with efficient key management and minimal false positive rates.

Table 1. Summary of existing applications of BF in different domains of the NDN

NDN Area	Title of paper	Bloom Filter Used	Features
Routing	BF based Routing	Simple BFs	• Reduces the communication cost and round trip delay. • Overcomes the challenges of topology robustness.
	COBRA (Content-driven BF based Intra domain routing algorithm)	Stable BFs	• Reduced the network overhead by a large extent. • Provides the same hit distance as of Dijkstra's algorithm
	Pull Based BFR	Simple BFs	• Consumed lesser bandwidth of network • Less memory requirements on client • BF based aggregation of CAR messages. • Better average round trip delay.
Forwarding	MaPIT	Mapping BF	• Reduced chip memory consumption to 2.097 MB • False positivity of 1% for almost 2 million records.
	NameFilter(2-staged BF)	Memory Access	• Provides 37 million searches per

ISSN 2348 – 8034
Impact Factor- 5.070

		BF and Merged BF	second in name prefix table of 10M entries. • Provides above result at a low memory cost of 234.27 MB. • It outperformed Character Trie, Bloom hash and lookup using simple BFs.
	Name Lookup Adaptive Prefix BF (NLAPB)	Adaptive Prefix BF	• NLAPB addresses the issues raised by BFs and trie by combining the best of both. • Reduces the false positivity raised by BFs • Reduces memory insufficiency raised by Trie.
	A new BF based architecture for FIB Lookup using 2-phase BF	2-phase BF	• Works on the principle of LPT, most efficient longest prefix matching data structure. • It uses the on chip BF's to store the prefix information. • 99.99 % requests are satisfied using on-chip BFs, which improves the FIB lookup drastically.
Caching	Cache Sharing using BFs	Stable BFs, Shifting BFs etc.	• It stores the cache information in BFs and share it with nearby routers (n). • Router don't cache data that is cached on its neighbor. • Stable BF drastically reduces query time and memory consumption for n≥3. • It improves the cache hit and content diversity by sharing cache summaries.
	Line-speed and accurate on-line popularity monitoring using BF:	Simple BFs	• BF based algorithm to monitor the popularity of content and further used for replacement. • It provides results at speed of 16.74Gbps at very low memory consumption of 32mb.
Security	Security and Privacy preserving NDN architecture	Traditional BFs	• It uses BFs to store the compressed secret keys and securely transmit over network. • It reduces the false positivity by using multiple BFs. • It uses CDKT for key management.

III. CONCLUSION

In this paper, we have covered NDN and BF. We have explored NDN, BF and applications of BFs in various NDN domains and tried to provide a comprehensive review on that. BFs have been applied on different NDN areas like routing, forwarding, caching and security. This survey can be used to find the various areas where BFs have been applied and further can be optimized using other BFs.

There are many open challenges in areas like routing, forwarding, caching, security etc. which can be explored further and BFs can be applied for providing efficient results. As an extension to this paper, NDN can be explored further to find more re-search gaps which can be addressed with the use of BFs in an efficient way.

ISSN 2348 – 8034
Impact Factor- 5.070

REFERENCES

1. *History of Computers and Computing, Internet, Maturing of the Net, TCPIP, www.history-computer.com/Internet/Maturing/TCPIP.html*
2. *L. Zhang, D. Estrin, J. Burke, V. Jacobson, J.D. Thornton, D.K. Smetters, B. Zhang, G. Tsudik, K.C. Claffy, D. Krioukov, D. Massey, C. Papadopoulos, T. Abdelzaher, L.Wang, P. Crowley, E. Yeh, Named Data Networking (NDN) Project, 2010, http://named-data.net/project/annualprogress-summaries/ (2010)*
3. *Z.A. Jaffri, Z. Ahmad, M. Tahir, "Named Data Networking (NDN), new approach to future Internet architecture design: A survey", Int. J. Inf. Commun. Technol. (IJ-ICT) 2(3), pp. 155–165 (2013)*
4. *Divya Saxenaa, Vaskar Raychoudhurya, Neeraj Surib, Christian Beckerc, Jiannong Caod, "Named Data Networking: A survey", Computer Science Review (Science Direct), Volume 19, pp. 15-55 (2016)*
5. *V. Jacobson, D.K. Smetters, D. James Thornton, P.F. Micheal, B.H. Nicolas, B.L. Rebeeca, "Networking named content", In: Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies, DOI: 10.1145/1658939.1658941, pp. 1-12 (2009)*
6. *Yu Zhang, Alexander Afanasyev, Jeff Burke, Lixia Zhang, "A survey of mobility support in Named Data Networking", In: 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), DOI: 10.1109/INFCOMW.2016.7562050, San Francisco, CA, USA (2016)*
7. *Zhiyi Zhang, Yingdi Yu, Haitao Zhang, Eric Newberry, Spyridon Mastorakis, Yanbiao Li, Alexander Afanasyev, Lixia Zhang, "An Overview of Security Support in Named Data Networking", IEEE Communications Magazine, Volume: 56, pp. 62 - 68 (2018)*
8. *Kaoutar Ahed, Maria Benamar, Rajae El Ouazzani, "New classification of Named Data Netwoking Applications", In: 2018 International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT), DOI: 10.1109/MoWNet.2018.8428857, Tangier, Morocco (2018)*
9. *Introduction to Probabilistic Data Structures - DZone Big Data, https://dzone.com/articles/introduction-probabilistic-0 (2015)*
10. *B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," In: Communications of the ACM, vol. 13, pp. 422-426 (1970)*
11. *Lailong Luo, Deke Guo, Richard T.B. Ma, Ori Rottenstreich, Xueshan Luo, "Optimizing Bloom Filter: Challenges, Solutions, and Comparisons", arXiv:1804.04777v1 [cs.DS] (2018)*
12. *A. Kirsch and M. Mitzenmacher, "Distance-sensitive bloom filters", In: ALENEX, vol. 6. SIAM, 2006, pp. 41–50 (2006)*
13. *J. Bruck, J. Gao, and A. Jiang, "Weighted bloom filter", In: 2006 IEEE International Symposium on Information Theory, DOI: 10.1109/ISIT.2006.261978, Seattle, WA, USA (2006)*
14. *D. Guo, J. Wu, H. Chen, Y. Yuan, and X. Luo, "The dynamic bloom filters", IN: IEEE Transactions on Knowledge and Data Engineering, vol. 22, pp. 120–133 (2010)*
15. *P. S. Almeida, C. Baquero, N. Preguiça, and D. Hutchison, "Scalable bloom filters" Information Processing Letters, vol. 101, pp. 255–261 (2007)*
16. *M. Yoon, "Aging Bloom Filter with Two Active Buffers for Dynamic Sets," In: IEEE Transactions on Knowledge & Data Engineering, vol. 22, pp. 134-138 (2009)*
17. *Yossi Kanizo, Isaac Keslassy, "The variable-increment counting bloom filter", In: IEEE/ACM Transactions on Networking (TON), Volume 22 Issue 4, pp. 1092-1105 (2014)*
18. *Elham Safi, Andreas Moshovos, Andreas Veneris, "L-CBF: A Low-Power, Fast Counting Bloom Filter Architecture", In: IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume: 16, pp. 628-638 (2008)*
19. *Ali Marandi, Torsten Braun, Kave Salamatiany, Nikolaos Thomos, "BFR: a Bloom Filter-based Routing Approach for Information-Centric Networks", arXiv:1702.00340v1 [cs.NI] (2017)*
20. *Michele Tortelli, Luigi Alfredo Grieco, Gennaro Boggia, Kostas Pentikousis, "COBRA: Lean intra-domain routing in NDN", In: 2014 IEEE 11th Consumer Communications and Networking Conference (CCNC), DOI: 10.1109/CCNC.2014.6994403, Las Vegas, NV, USA (2014)*
21. *Ali Marandi, Torsten Braun, Kave Salamatiany, Nikolaos Thomos, "Pull-based Bloom Filter-based Routing for Information-Centric Networks", arXiv:1809.10948v1 [cs.NI] (2018)*

22. *Zhuo Li, Kaihua Liu, Member, IEEE, Yang Zhao, Yongtao Ma, "MaPIT: An Enhanced Pending Interest Table for NDN with Mapping Bloom Filter", IEEE COMMUNICATIONS LETTERS, VOL. 18, pp. 1915-1918 (2014)*
23. *Yi Wang, Tian Pan, Zhian Mi, Huichen Dai, Xiaoyu Guo, Ting Zhang, Bin Liu, "NameFilter: Achieving fast name lookup with low memory cost via applying two-stage Bloom filters", In: 2013 Proceedings IEEE INFOCOM, DOI: 10.1109/INFCOM.2013.6566742, Turin, Italy (2013)*
24. *Wei Quan, Changqiao Xu, Jianfeng Guan, Hongke Zhang, Luigi Alfredo Grieco, "Scalable Name Lookup with Adaptive Prefix Bloom Filter for Named Data Networking", IEEE COMMUNICATIONS LETTERS, VOL. 18, pp. 102-105 (2014)*
25. *Hayoung Byun, Hyesook Lim, "A New Bloom Filter Architecture for FIB Lookup in Named Data Networking", Appl. Sci. 2019, 9, 329; doi:10.3390/app9020329 (2019)*
26. *Ju Hyoung Mun, Hyesook Lim, "Cache sharing using bloom filters in named data networking", Journal of Network and Computer Applications 90 (2017), 74–82 (2017)*
27. *Huichen Dai, Yi Wang, Hao Wu, Jianyuan Lu, Bin Liu, "Towards Line-Speed and Accurate On-line Popularity Monitoring on NDN Routers", In: 2014 IEEE 22nd International Symposium of Quality of Service (IWQoS), DOI: 10.1109/IWQoS.2014.6914318, pp. 178-187 (2014)*
28. *Emmanuel A. Massawe, Suguo Du, Haojin Zhu, "A Scalable and Privacy-Preserving Named Data Networking Architecture based on Bloom Filters", In: 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops, DOI 10.1109/ICDCSW.2013.32, pp. 22-26 (2013).*