ATHENS UNIVERSITY OF ECONOMICS AND BUISINESS

M.Sc. in Computer Science

Evaluation of multicast efficiency improving techniques in CCN networks

Anastasios Litsas-Alexis

Supervisor: George Xylomenos

Athens, July

Abstract

Recently, content centric networking (CCN) has become very popular as an alternative way to route data without using IP. In CCN every request for data is stored in every router through which it travels and the answer to that request follows its traces until it reaches the client that issued it. CCN also comes with some major drawbacks like the overwhelming need for memory in routers. Various techniques have been proposed to solve this problem, with a lot of weaknesses themselves which have also become an object of research. The purpose of this study is to evaluate enhancements to a method that consists of not storing the request in every router but storing it every D hops (D>1) and using Bloom filters to describe the direction the request came from, so as to reduce memory consumption in routers. Specifically, we evaluate the method of permuting the Bloom filters to avoid routing loops and flow duplication and the method in which the number of bits that get set during the double hashing procedure to construct the initial Bloom filters depends on the degree of the current node, so as to reduce the transmissions of redundant packets. We implemented a simulation of a CCN network together with the previously described techniques and we discovered that by using the right combination of them one can achieve a major improvement in the system's efficiency.

Acknowledgements

I would like to thank professor George Xylomenos for supervising my work and professor George Polyzos for his evaluation of my thesis. I am grateful to Christos Tsilopoulos for his support and contribution to the development of our software and Yannis Thomas for his help with the interpretation of the statistical results of our reasearch.

Contents

1 Introduction

IP has been used extensively throughout the last decades as a simple way to route packets in networks ranging from small LANs to the whole Internet. But IP has some major problems. An IP address has a geographical meaning. In order to get a piece of data one wants, he has to know where to get it from, meaning in a way that he has to know an approximate geographical location even if he doesn't refer to it explicitly, since host names and IP addresses are for the most part geographically organized. Another issue with IP is the fact that IP routers are completely unaware of the content they transmit. They don't process payloads except for the addresses they need to process in order to route a packet on the layer of abstraction they have to route it. In other words they only know what to do with the addresses. This can sometimes lead to very inefficient behaviour. If thousands of users request a certain piece of data and their requests get routed through the same router (something that can happen very often in situations where the data is stored in another country or continent and there are not many available routes to get it) then thousands of copies of the same piece of data will be returned through the same router and at many times there will be thousands of these exact copies stored inside the same router. This is obviously extremely inefficient.

In [1], the authors propose a whole other way of routing data that can be implemented on top of or instead of IP. They propose content centric networking (CCN), a way to separate location from content and also to make routers aware of the content they distribute so that unnecessary storing and transmitting can be almost eliminated. Every piece of data in the system is referred to by a unique name, this way every part of the system can know exactly which piece of data it handles at any time.

In CCN there are two types of packets. Interest packets and Data packets.

Interest packets are issued by clients in order to request a piece of data by its name and owners of data reply to these requests by sending Data packets containing the data requested. In CCN all packets include a name, a unique identifier that identifies the piece of data requested.There are three basic data structures every CCN router implements, described in the following sections.

1.1 The pending interest table (PIT)

The PIT is a data structure that stores pending requests that have been forwarded previously towards some owner of data. When the Data packet containing the requested data returns, it will be forwarded back towards the client that requested it using information from the PIT describing the interface of the router from which the request came. The interface can be something like a network adapter with only one point to point connection in terms of hardware. In CCN there is no global programming of any route. Every router is just concerned with its point to point connections and just has to decide towards what neighboring node a packet must be transmitted. No routing state other than that is used.

The name of the requested object as well as the interface through which it has to be transmitted to return to the client is stored inside the PIT as soon as the Interest packet containing the request arrives at the router. If a request for a piece of data arrives at a router where the PIT doesn't already contain such a request then it is stored in the PIT and an Interest packet is transmitted through the interface dictated by the FIB. If a request for a piece of data arrive at a router where the PIT already contains such a request then no Interest packet is transmitted, but the already existing PIT entry gets updated with the new interface through which the Data packet has to be forwarded when it arrives so the Data packet will not only be transmitted

towards the first client that asked for it but also towards the second and any other possible client. The very fact that an answer is transmitted to any possible client makes this procedure qualify as multicast.

1.2 The forwarding information base (FIB)

The FIB is a data structure containing information regarding the interface through which every possible request for data has to be transmitted. Like a routing table that contains information about every piece of data available in the system and the correct direction (interface) to reach it. Whenever an Interest packet arrives at a router, the FIB is checked to find out what the next hop of the packet has to be in order to get closer to the data owner. Every FIB in the system has to know all the data names published in the system.

1.3 The content store (CS)

The content store is a kind of cache memory every router is equipped with. When an Interest packet arrives at a router there is a chance for a hit in this kind of cache memory if the requested piece of data was previously routed through that node. According to [1]: "The Content Store is the same as the buffer memory of an IP router but has a different replacement policy. Since each IP packet belongs to a single point-to-point conversation, it has no further value after being forwarded downstream. Thus IP 'forgets' about a packet and recycles its buffer immediately on forwarding comple-

tion (MRU replacement). CCN packets are idempotent, self-identifying and self-authenticating so each packet is potentially useful to many consumers (e.g.,many hosts reading the same newspaper or watching the same YouTube video). To maximize the probability of sharing, which minimizes upstream bandwidth demand and downstream latency, CCN remembers arriving Data packets as long as possible (LRU or LFU replacement)."

1.4 Scaling issues with CCN

CCN has some major scalability problems. One of the biggest problems with CCN is the scaling of the PIT sizes. Router memory is very limited making it nearly impossible to fit huge PITs inside it. When a client issues an Interest packet requesting some data then the PITs of every CCN router between the client and the owner of that data will store that request. As soon as the clients become many, requesting many pieces of data each, the PITs become very big for the average router memory.

Another problem is the information inside the FIB. FIBs have to know about every object published in the system. This means that every router in the system has to contain information for the direction through which every data owner can be reached. That makes FIBs also very big to fit in the memory.

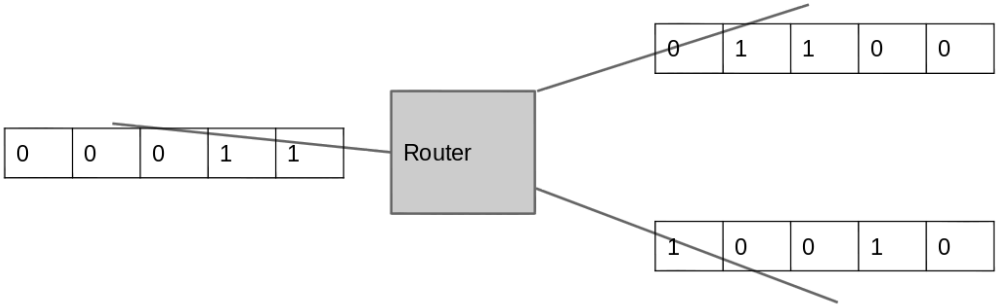
As authors in [2] note: "The number of Interests that must be stored in the PIT in order to fully utilize the network depends on the link capacity, the size of Data packets and the average Round-Trip Time (RTT) which defines the lifetime of Interests inside the PIT. A rough estimation for the required number of PIT entries per link is $bandwidth \times RTT/data\_packet\_size$. For

example, to fully utilize a 40 Gbps link with 1000-byte Data packets and an average RTT of 80 ms, the PIT must contain 400K entries; this must be multiplied by the number of links hosted by the router."

2 Reducing forwarding state

2.1 Tracking every D hops

Authors in [2] propose a solution (semi stateless forwarding) that reduces the average PIT size significantly. They propose tracking Interests not at every hop but every D hops (D is a small integer), meaning that a PIT is used every D routers instead of every router's PIT being used. In between the routers of which the PITs are used, Interest packets gather information about the route and encode it into a Bloom filter they carry. According to [2]: "Instead of tracking an Interest at either all or none of the routers, we store forwarding information at some routers. An Interest is tracked at every d hops, where d is a predefined system parameter. If a data path is N hops long, an Interest is on average tracked at N/d routers and each router tracks on average 1/d of the forwarded Interests. At intermediate hops, Interests collect reverse path information, which is stored at routers tracking that particular Interest. Data are later forwarded via Bloom filter-based stateless forwarding [9] between the routers tracking the corresponding Interest. Our solution reduces the forwarding state maintained at routers, while preserving the advantages of name-based forwarding. Specifically, native multicast and host anonymity are preserved by the adoption of Bloom filter-based forwarding, while routers can still discard unwanted traffic and support adaptive forwarding for the



A router with three interfaces and their randomly selected five bit LIDs.

fraction of Interests that they are tracking. In addition, our approach does not radically change CCN, as it only modifies Interest and Data forwarding, leaving the control plane (routing information exchange and bootstrapping) intact."

This can be achieved by assigning link IDs (LIDs) to every interface in the system [2]. This way the LIDs, which are be binary arrays, can be added in the Bloom filters by ORing them with it so that the Bloom filter will be a description of the set of the interfaces that would have to be used in order for a Data packet to return to the right place. Ideally every LID in the system would be unique, but this could only be achieved if there was a global network state stored somewhere for centralized control to be possible. Since one of CCN's main advantages is the lack of need for centralized control LIDs have to be chosen randomly with a very low probability of every LID not being unique. Double hashing can be used to choose the LIDs [2].

2.2 Double hashing

Double hashing is a way to describe something using an array of bits. We can use double hashing to construct random bit arrays (LIDs) for every network interface of every router. This way no centralized control needs to be present in the system because every router chooses its own LIDs. Using two hash functions ($h(x)$, $f(x)$) we hash an element relative to what we want to describe. In our case it's a network adapter (in our implementation every network interface is supported by a separate network adapter) so we can use it's MAC address. Let L be the length of the array. Given a MAC address x and a parameter k, we set k of the bits in the bit array. The indices of these bits will be:

$$h(x) + j^2 * f(x) mod L \quad for\ every\ j,\ 0 \leq j < k$$

The probability of these LIDs not being unique will be very low since every network adapter has a unique MAC address and in our implementation we use 128 bit LIDs (same length as the Bloom filters we use).

2.3 Packet routing

As authors in [2] say: "During Interest propagation, instead of updating the PIT at each router, the Interest is tracked at every d hops, where d is a predefined system parameter, e.g. d = 3 or d = 4. We call d the Forwarding State Reduction Factor. Intermediate routers add reverse path information inside Interests. When a router tracks an Interest, instead of storing the Interest's incoming interface, the router stores the reverse path (or tree) gathered by the Interest. During Data forwarding, routers that tracked a particular Interest place the source-route for the downstream path

(tree) in the Data packet and push it towards the next stateful router(s). Between stateful points, packets are forwarded according to the in-packet source-route."

As the packet travels it adds the LID of every interface it came from to its Bloom filter by ORing them. Whenever it's time to use a PIT, the Bloom filter of the Interest packet is stored inside it. A hop counter in every packet can be used to count the hops and decide when it's time for a PIT to be used. As noted in [2]: "HC is incremented at each hop and when HC = d, routers store the Interest in their PIT. Routers always perform an initial PIT lookup in order to suppress multicast Interests regardless of the HC value, as in Probabilistic tracking. The initial value for the HC is set by the issuing host but instead of setting it to 0, the initial HC is randomly selected in the range [0, d − 1] so as to distribute forwarding state to all routers. If the initial HC was always set to 0, routers with distance d − 1 from hosts would be kept stateless. In the example of Figure 1, if d = 3, all Interests issued by hosts would be tracked by R4, while R1 to R3 would have an empty PIT. PIT state would thus be unevenly distributed: R4 would be a bottleneck point and R1 to R3 would not participate in (say) adaptive forwarding at all. In contrast, with a randomly selected HC, there is a 1/d probability for each on-path router to track the Interest."

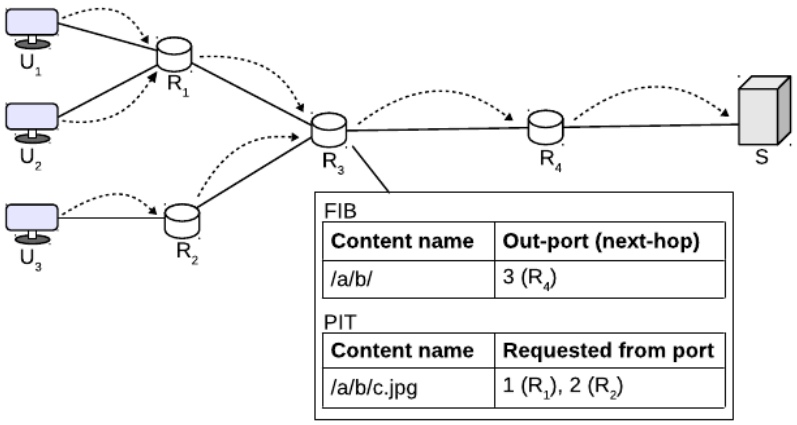


Fig. 1. Basic CCN operation. Arrows show the propagation of Interests towards content source S. The contents of the FIB and PIT at router R_3 are shown. Data follow the reverse path (tree), based on the PIT of each router. [2]

As the Data packet travels from the data owner towards the client that issued the request, it starts by taking an initial Bloom filter from the first PIT it runs into (which is the PIT of the router serving the data owning application) and uses it to travel to the next. At every hop, the interface through which the packet must be transmitted is decided by checking which of the LIDs of the available interfaces are found to have been added in the filter. This is done by performing an AND between every LID and the Bloom filter in the packet. If the result is the LID then we transmit through the interface that LID belongs to.

```
if ( LID  &  BF  ==  LID ) transmit;
```

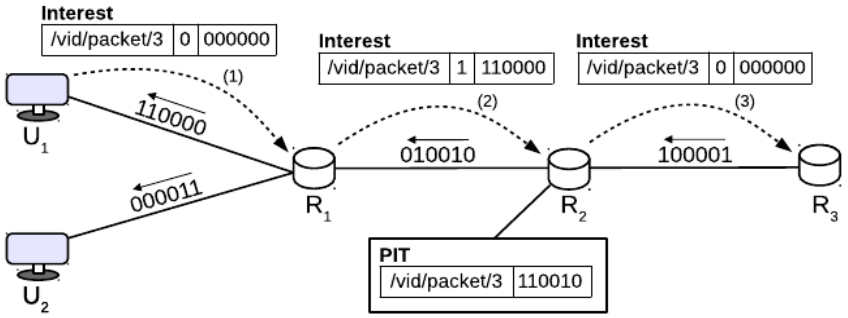
This way the Data packet returns to everyone that issued a request. Of course the false positives are a side effect. When the Data packet arrives

at a router where it is supposed to use the PIT, it replaces its Bloom filter with the one from the PIT and continues. According to [2]: "To integrate Bloom filter-based forwarding in CCN, we extend Interest and Data packets to carry an iBF in their headers. Interest packets accumulate the iBF for the traversed (reverse) path and Data packets carry the iBF for the delivery path (or tree). Specifically, upon receiving an Interest, routers update the Interest's traversed path by adding (ORing) the outgoing LID of the packet's incoming link. If a router decides to store an Interest in its PIT, it also stores the iBF and then resets the iBF in the Interest before further forwarding it. When the respective Data packet arrives, the router acts as a relay point by inserting the stored iBF in the Data packet and then forwarding it based on the iBF. Figure 4(a) shows an example where d = 2, U1 and U2 are two multicast users and the network uses Hop Counter-based tracking. At some point, U1 requests the packet /vid/packet/3 with initial HC = 0. U1 creates the Interest with an empty iBF (i.e. all bits are set to 0) and transmits the packet. R1 receives the Interest, increases the HC and adds the LID for the reverse direction, i.e. LID_{R1} →U1, to the Interest iBF (step 1). R1 forwards the Interest to R2. Node R2 increases the HC to 2 and adds LID_{R2} →R1 to the Interest iBF (now containing the path R2 → R1 → U1). Since HC = 2 (= d), R2 stores the Interest along with the iBF in its PIT (step 2). R2 then resets the Interest's HC and iBF and forwards the Interest (step 3). This continues until the Interest reaches the data source.

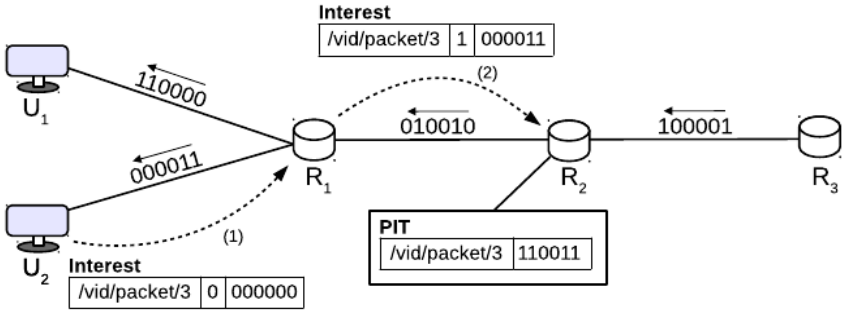
During Interest forwarding, if a router finds a matching PIT entry, it adds the Interest's iBF to the iBF already stored in the PIT. The resulting iBF is the union of the already stored and the additional path links, which form a multicast tree. This is shown in Figure 4(b). U2 transmits an Interest for the same content as U1 did, with initial HC = 0. The request arrives

at R1 which updates the Interest's HC and iBF (step 1) and forwards the Interest to R2. At that point, R2 updates the Interest's iBF, adds it to the iBF already stored in the PIT and suppresses the Interest (step 2). The PIT entry at R2 now contains the iBF for the multicast tree R2 → R1 → U1, U2.

Upon the arrival of a Data packet, a router checks its PIT and if a matching entry exists, it replaces the Data iBF with the stored iBF and further forwards the packet. If no entry exists, the router forwards the Data packet according to its iBF. If no LID matches the Data packet's iBF, the router drops the packet. Finishing the example of Figure 4, when the Data packet /vid/packet/3 arrives at R2, the router replaces the Data iBF with the one stored in the PIT. iBF now contains LID_{R2} →R1, LID_{R1} →U1 and LID_{R1} →U2 therefore the packet is delivered to R1 which then duplicates the Data packet to U1 and U2 . It is important to note that even though the iBF is stored at a non-branching router (R2), Bloom filter-based forwarding ensures that the Data packets are only duplicated at branching nodes (R1)."



(a) Interest from U_1: R_1 updates the Interest iBF. R_2 tracks the Interest in its PIT with the iBF for $R_2 \rightarrow R_1 \rightarrow U_1$. R_2 resets the Interest iBF to 0 and further forwards the Interest.



(b) Interest from U_2: R_1 updates the Interest iBF. R_2 adds the iBF to its *existing* PIT entry and suppresses the Interest. The stored iBF contains the multicast tree to U_1 and U_2.

Fig. 4. Interest propagation using iBFs to track reverse paths. Only right-to-left LIDs are shown.

[2]

By using the PITs only every D hops we share the PIT records between many routers instead of storing every record on every router. This way the average PIT size is reduced. The downside of this is that as a result of using Bloom filters, false positives appear in our system and many unneeded Data packets are constructed just because of that. More Interest packets are also transmitted, since Interest packets do not always merge at the first

opportunity.

3 Problems with Bloom filter based routing

Flow duplication consists of Data packets that arrive to their destination using various different routes. Routing loops are loops that a Data packet might follow. When this happens, a copy of the Data packet is sent to the legitimate receivers every time the packet goes through the loop [3]. Flow duplication and routing loops can be caused by false positives occurring in operations with Bloom filters [3]. False positives can easily occur during operation with Bloom filters meaning that an element that might not have been inserted in the set described by the Bloom filter, might seem to have been added just because some other element or elements have been added that happened to set the same bits. Authors in [3] note that "The probability of a false positive is $fpr = \varrho^k$ where ϱ is the fill factor of the filter (i.e., the percentage of bits set to 1) and k is the number of bits set for each link added to the filter. Previous discussions of the effects of false positives in such in-packet Bloom filter-based forwarding applications has revolved around two issues: (1) the extra bandwidth they consume, and (2) potential forwarding loops they may cause." and that "Bloom filter false positives can be controlled by adjusting the size of the filter, or by limiting the number of links stored in it. The latter is effectively a limit on the size of the multicast tree (or the length of the forwarding path for unicast). In order to optimize the Bloom filter for the largest trees, the limit can be set so that $\varrho \preceq 50\%$. In that case, the false positive rate will be $fpr = 2^{-k}$, and the maximum size of the path or multicast tree that can be encoded in the Bloom filter is approximately $n = (m/k) \cdot ln2$."

To deal with the false positives caused by the use of Bloom filters in the system, the authors in [3] propose two techniques among others. They pro-

pose varying the k parameter of the double hashing procedure implemented to choose the LIDs of every interface and the permutation of Bloom filters at every hop. We will explain these techniques in the following subsections.

4 Varying k

In this technique, every node chooses its own value for the k parameter of the double hashing it will execute to chose the LIDs of its interfaces. Specifically, it chooses:

$$k = \lceil \log_2(d-1) \rceil + r$$

where d is the degree of the node and r is a small integer. This technique guarantees that $fpr < \frac{\varrho_{max}^r}{d-1}$ (ϱ_{max} is a global variable of the system). This technique according to [3] is supposed to prevent false positives at the high degree core parts of the network and also optimizes the usage of the capacity of Bloom filters so there is no unnecessary safety margin and the Bloom filter can be used to encode larger multicast trees.

5 Bloom filter permutations

Permuting the bits of the Bloom filter can also be used to fight flow duplication and routing loops. We rearrange the bits in the Bloom filters so that even if two packets are exact copies, their filters will be different if their

history is different. Every router has a randomly selected permutation that characterizes it. Every time an interest packet arrives at a node, its Bloom filter is updated with the LID of the interface from which it came and then it's permuted using the router's unique permutation before it is transmitted. Every time a data packet arrives at a router its Bloom filter is permuted and then it is forwarded through the interfaces that are found to have been added to its Bloom filter.

If two copies of the same packet arrive at a node, only one has followed the route intended by the routing algorithm. To discern the right packet from the one that is a result of a false positive, we would like to look at the history of the packet but that of course isn't possible. Using the bit permutation technique of the Bloom filters, the packet can accumulate information about the route it has traversed and if it has followed the intended path it will carry a Bloom filter permuted the right way. If it it's coming from the right path (the path the Interest packets follow to reach the data owner) then it will have been subjected to the exact same permutations its corresponding Interest packet had been subjected to as it traveled towards the owner. This way its Bloom filter will be in exactly the right condition when it arrives, meaning it will be capable of continuing and it will not be dropped when the router's LIDs get examined. If the packet hasn't followed the right path then the permutations performed on the corresponding Interest packet's Bloom filter will not have been mutually exclusive with the ones the Data packet's Bloom filter has been subjected to, so the Bloom filter will be useless, incapable of allowing even one more hop (unless another false positive appears at the time). There is no way to know a packet's exact history, but if the history is wrong the packet is rendered useless and is dropped.

```
Input: LinkMasks of the node;
  Permutations of the node;
  iBF in the packet header;
let π = Permutations[ingress link]
set iBF in packet to π(iBF);
foreach outgoing link l do
  let mask = LinkMasks[ingress link, l]
  if iBF & mask == mask then
    Forward packet on the outgoing link l
  end
end
```

Fig. 4. Pseudocode for packet forwarding [3]

6 Implementation

We implemented a content centric network simulator and a content centric network simulator with tracking every d hops [2], both based on ns3[12]. We also implemented the varying k and the permutations techniques proposed in [3]. For double hashing we use implementations of sha-1 and md5 hash functions ([8] and [9]). In our implementation we don't use Content stores[1], all FIBs are filled before the experiment by traversing the graph of the topology using BFS starting from the data owner and filling every router's FIB with entries referring to the direction to which the data owner can be reached using the minimum number of hops possible. We make an entry for every piece of data we want the clients to request. When the experiment starts we choose the clients and the data owner from the graph's access nodes using the modern version of Fisher-Yates shuffle algorithm [10] introduced by Richard Durstenfeld in [11]. (See Evaluation section for more details).

Every CCN router in our simulations counts every Interest and every Data packet it encounters incrementing a global counter which is common for the whole system; this way we keep track of the total number of Interest and Data packets that appeared. Using the discrete event simulation capabilities of ns3[12] we schedule the number of PIT entries of every router in the system to be counted at a time when all the Interests have reached the data owner but no answers have been sent yet (answers would have consumed the PIT entries on their way). This way we can count every PIT entry. In order to be sure that no answers have been sent yet at the time we scheduled the counting of the PIT entries to take place, we have scheduled the data owner to reply with a specific delay.

The simulators we implemented are capable of simulating any of the basic procedures content centric networking includes. They offer the capability

of instantiating as many nodes one needs as long as ns3[12] can support that many on the specific machine being used. Our CCN simulator can be run with the initial seed of its random generator and the number of interest requests to be transmitted as arguments. Our semi-stateless forwarding simulator expects as arguments the initial seed of its random generator, the number of interest requests to be transmitted, the length of the Bloom filters (in bytes), the k parameter of double hashing and whether varying k [3] and Bloom filter permutations [3] should be used. If varying k is used then k is useless.

The number of experiments our simulators execute is 20 for every possible number of clients and for every d (in the case of semi stateless forwarding) and during each one of them they program every client node to issue the number of requests that has been given to the application as an argument. Each one of these requests refers to a different data object, so the number of data objects present in the system is also programmed accordingly. Our simulators count the PIT size of every router during each experiment as well as the total number of Data and Interest packets that appeared and write all the statistical data in text files which we afterwards parse to calculate the properties we are interested in. We conducted our experiments using 128 bit Bloom filters, k=4 (when varying k was not used) for our semi stateless forwarding simulator and 100 interest requests for both our simulators. As our random number generator we use ns3's[12] UniformRandomVariable.

PIT sizes

Authors in [4] identified the d-left open-addressed hash-table data structure as a good solution for the implementation of a PIT so we calculated the PIT sizes accordingly. We assume a 32 bit machine [2].

CCN PIT element size:

32 bit pointer to name	32 bit pointer to next element in bucket	16 bit net interface [2]	20 or 56 byte name [5]

30 or 66 bytes total

PIT element size with BFs:

32 bit pointer to name	32 bit pointer to next element in bucket	128 bit Bloom filter	16 bit destination interface [2]	16 bit application	20 or 56 byte name [5]

48 or 84 bytes total

7 Baseline Evaluation

To perform our evaluation we used topologies from Internet Topology Zoo[6] and Rocketfuel[7] as in [2]. More specifically we used the following topologies: GEANT and UniNet from [6] and 6461, 1221, 1755, 3967 from [7].

Topology	Number of nodes	Density
GEANT	39	0.153846
UniNet	74	0.0747871
1221	104	0.0563854
6461	138	0.0787052
1755	87	0.0860732
3967	79	0.0954236

We used GEANT for our main experiments and the others in our effort to discover if the benefit that varying k gives depends on the topology's density.

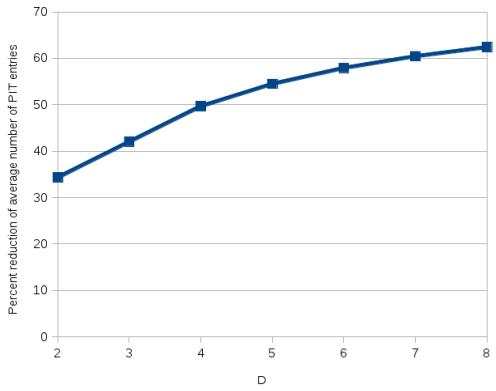
We evaluated our implementation of the system from [2] in order to discover to what extent it reduces PIT sizes compared to the system in [1]. We performed experiments for every D from 2 to 8, using 128 bit Bloom filters and double hashing with k=4. In every experiment we use random access nodes as clients and one access node as the data owner. These are chosen randomly at the beginning of every experiment. We perform 20 experiments for every possible number of clients with randomly selected participants every time. The results that follow are the averages of all 20 experiments of all different client group sizes. Every comparison that is done, is done between groups of experiments that were run with exactly same seeds so exactly the same nodes were chosen as clients and as owner and exactly the same pseudo-

random numbers were used every time for every algorithm in our system that needs to use pseudorandom numbers. The system proposed in [2] achieved the results in the following subsections (using the GEANT [6] topology), which are a performance baseline against which possible improvements can be evaluated.

7.1 Percent reduction of average number of PIT entries

D	percent reduction of PIT entries avg
2	34.3506208912
3	42.0540540541
4	49.6997808619
5	54.495982469
6	57.9364499635
7	60.4528853178
8	62.4200146092

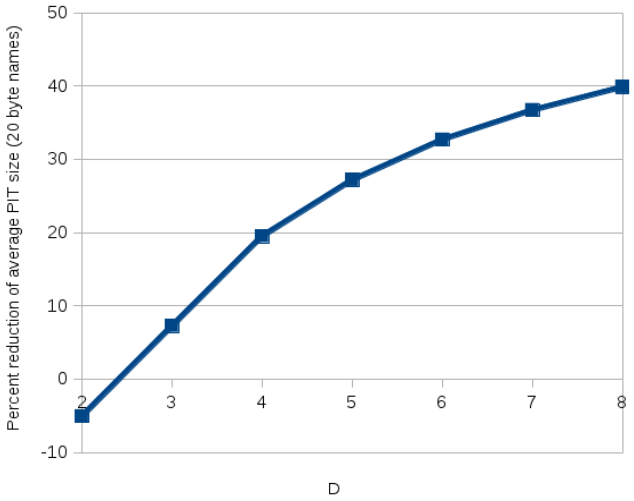
This plot displays the percent reduction of the average number of PIT entries that is achieved for every value of d we experimented with. 40 percent means that average PIT size went 40 percent down compared to the simple CCN system. It's obvious that for bigger d's the PIT entries become shared among more routers so average PIT size becomes smaller. However, the PIT entries in the modified system are larger, we need to examine the actual size savings achieved.



7.2 Percent reduction of average PIT size with 20 byte object names

D	percent reduction of PIT size avg (20 byte names)
2	-5.0390065741
3	7.2864864865
4	19.5196493791
5	27.1935719503
6	32.6983199416
7	36.7246165084
8	39.8720233747

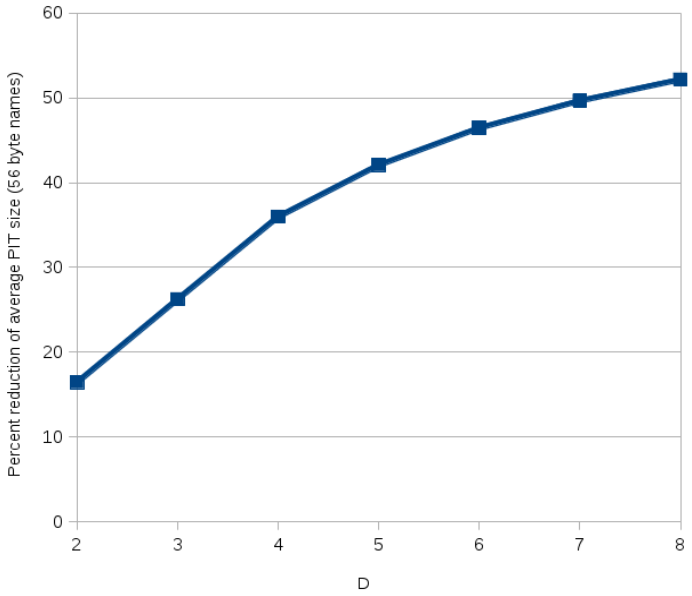
This plot displays the percent reduction of the average PIT size that is achieved for every value of d we experimented with compared to a simple CCN system. When the names are only 20 bytes long, the situation is even worse for some D. This happens because the overhead of a big Bloom filter in the PITs is not worth it when we have tiny names.



7.3 Percent reduction of average PIT size with 56 byte object names

D	percent reduction of PIT size avg (56 byte names)
2	16.4462447706
3	26.2506142506
4	35.9815392788
5	42.0857958696
6	46.4645726808
7	49.6673085862
8	52.1709276844

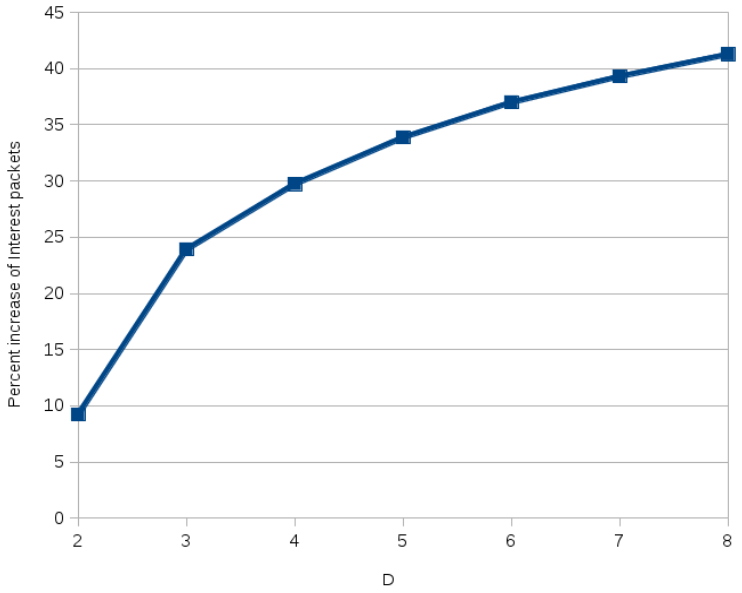
This plot displays the percent reduction of the average PIT size that is achieved for every value of d we experimented with compared to a simple CCN system. With larger names, we always have an improvement.



7.4 Percent increase of Interest packets

D	percent increase of Interest packets
2	9.2077484559
3	23.9096013476
4	29.7265581134
5	33.8708590679
6	36.9966311061
7	39.3048848961
8	41.2717574396

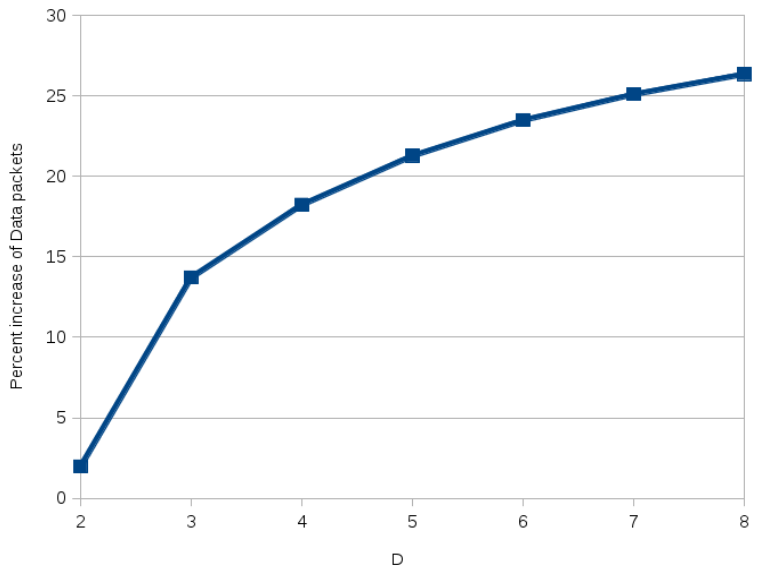
This plot shows the percent increase of Interest packets that appeared in the system compared to a simple CCN system. These packets are due to the fact that Interest packets for the same content are not always merged at the first opportunity.



7.5 Percent increase of data packets

D	percent increase of data packets
2	1.9696799551
3	13.718697361
4	18.2369455362
5	21.2863559798
6	23.4969118473
7	25.1117349803
8	26.3627175744

This plot shows the percent increase of Data packets that appeared in the system compared to a simple CCN system. These are due to the false positives in the Bloom filters, and their reduction is the main goal of the techniques studied below.



8 Evaluation of false positive reduction techniques

We evaluated our implementation of the techniques from [3] in order to discover to what extent they reduce the false positives in Bloom filter operations. We perform experiments for every D from 2 to 8, using 128 bit Bloom filters and double hashing with k=4. In every experiment we use random access nodes as clients and one access node as the data owner. These are chosen randomly at the beginning of every experiment. We perform 20 experiments for every possible number of clients with randomly selected participants every time. The results that follow are the averages of all 20 experiments of all different client group sizes. Every comparison that is done, is done between groups of experiments that were run with exactly same seeds so exactly the same nodes were chosen as clients and as owner and exactly the same pseudorandom numbers were used every time for every algorithm in our system that needs to use pseudorandom numbers.

The tables in the following subsections show the percent of additional Data packets that appeared in the network using just the system from [2] (none column), the percent of additional Data packets using a certain technique (or both), and the difference between them. The experiment of which the results are described below is conducted using exactly the same seeds for the random number generator of ns3[12] as in the previous experiment to evaluate the system from [2]. This way we know that the clients and the owner during every experiment are exactly the same as before allowing us to make an accurate comparison between to find out how the false positives were affected by the techniques from [3] tested here.

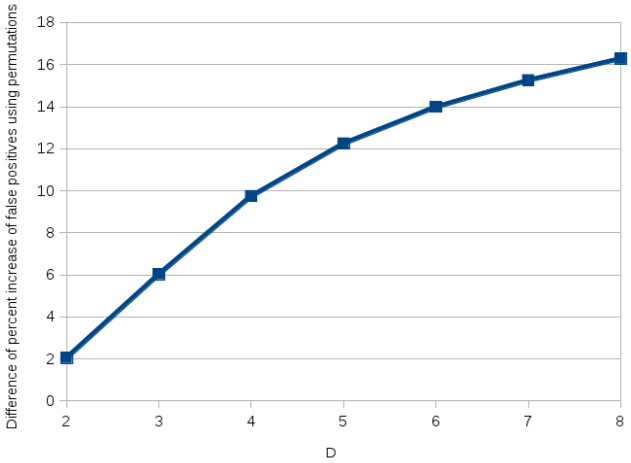
PIT sizes and extra interest packets are not an issue here since they

are not affected by these techniques. The previous results show how much the average PIT size was reduced by the proposition in [2] and how much more false positives appeared as a result of it while the following results show the next possible step towards improvement, meaning that now that we established that the average PIT size is reduced by [2], we measure how the techniques in [3] can affect the false positives caused by [2]. The increase of interest packet that was caused as a result of tracking interests every D hops cannot be reversed by these techniques, that is, the extra interest packets are necessary. The techniques proposed in [3] achieved the results shown in the following subsections (using the GEANT [6] topology).

8.1 Bloom filter permutations

D	none	permutation	difference
2	1.9696799551	-0.0830993824	2.0527793374
3	13.718697361	7.6855699046	6.0331274565
4	18.2369455362	8.4912970241	9.7456485121
5	21.2863559798	9.0381807973	12.2481751825
6	23.4969118473	9.5075800112	13.9893318361
7	25.1117349803	9.8534531162	15.2582818641
8	26.3627175744	10.0780460415	16.2846715329

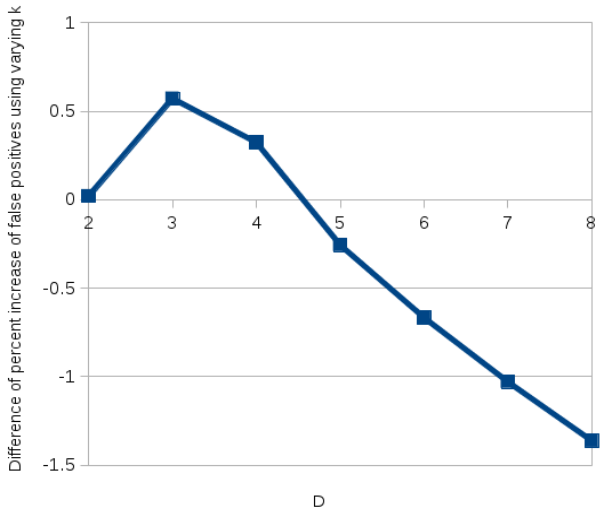
When interests are tracked every d hops, as mentioned before, false positives occur. A measure of how many false positives have occurred is how much percent did the Data packets in the system increase. This plot shows the difference between how much they increased (compared to a simple CCN system) in a system like [2] and how much they increased when the permutation technique was used. Every positive percentage means that false positives were reduced.



8.2 Varying k

D	none	var k	difference
2	1.9696799551	1.9489051095	0.0207748456
3	13.718697361	13.1471083661	0.5715889949
4	18.2369455362	17.9135317238	0.3234138124
5	21.2863559798	21.542953397	-0.2565974172
6	23.4969118473	24.1633913532	-0.6664795059
7	25.1117349803	26.1409320606	-1.0291970803
8	26.3627175744	27.7248736665	-1.3621560921

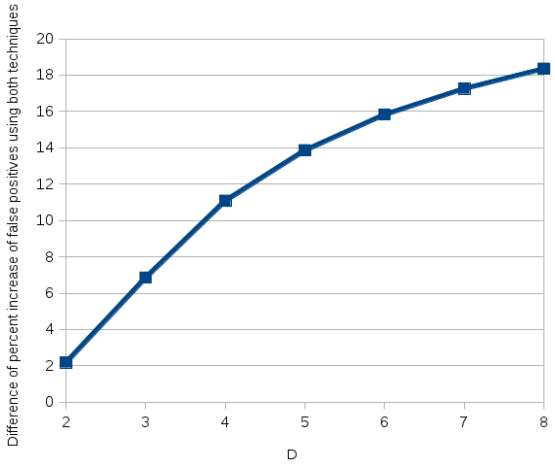
This plot shows the difference between how much they increased (compared to a simple CCN system) in a system like [2] and how much they increased when the varying k technique was used. Every positive percentage means that false positives were reduced. Note that in some cases this technique led to worse results, that is, more redundant data packets.



8.3 Results using both techniques

D	none	combination	difference
2	1.9696799551	-0.2172936553	2.1869736103
3	13.718697361	6.8624368332	6.8562605278
4	18.2369455362	7.1516002246	11.0853453116
5	21.2863559798	7.4256035935	13.8607523863
6	23.4969118473	7.6681639528	15.8287478945
7	25.1117349803	7.8596294217	17.2521055586
8	26.3627175744	8.0101066816	18.3526108928

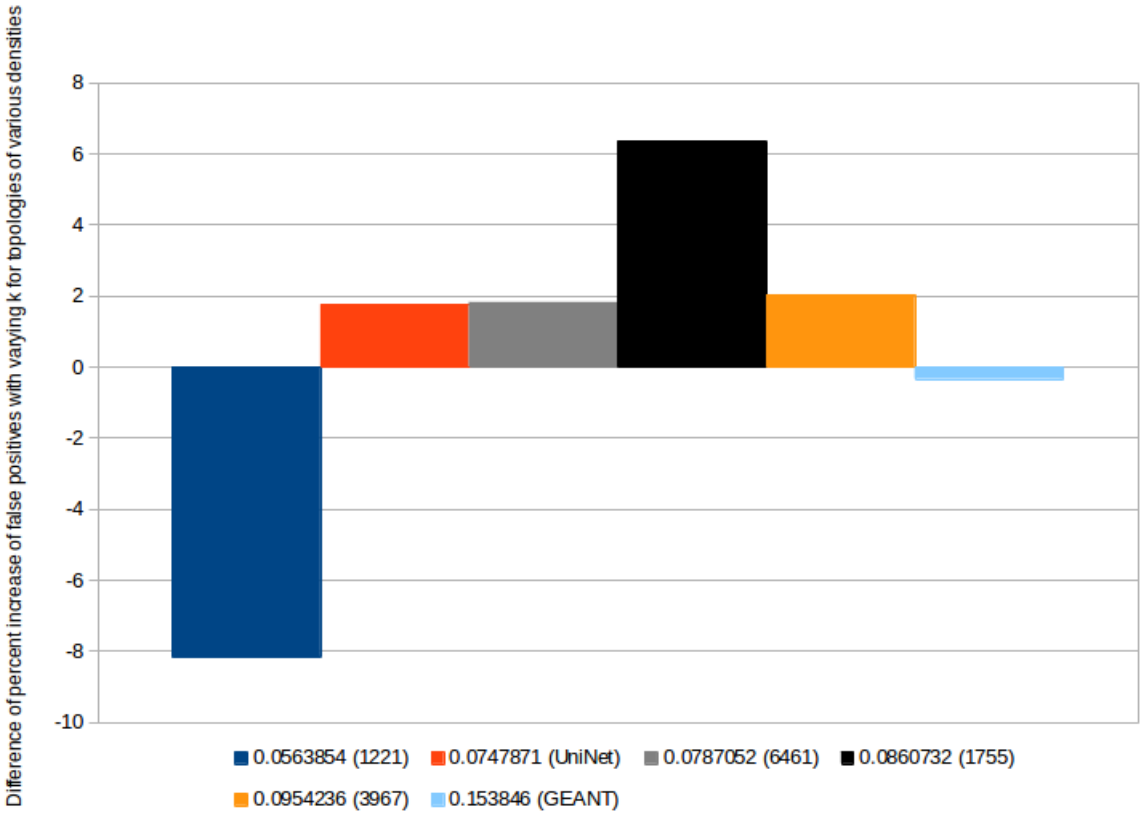
his plot shows the difference between how much they increased (compared to a simple CCN system) in a system like [2] and how much they increased when the permutation and the varying k technique were both used. Every positive percentage means that false positives were reduced. In this case we have always a positive improvement, and the combination of both techniques outperforms each technique in isolation.



8.4 Further analysis of varying k technique

The technique of varying k showed improvements only for $2 \leq D \leq 4$. For $D > 4$ it actually increased the false positives. We tried it with six different topologies in order to check whether or not the density of the topology plays a role in the effectiveness of the varying k technique. Density is defined as:

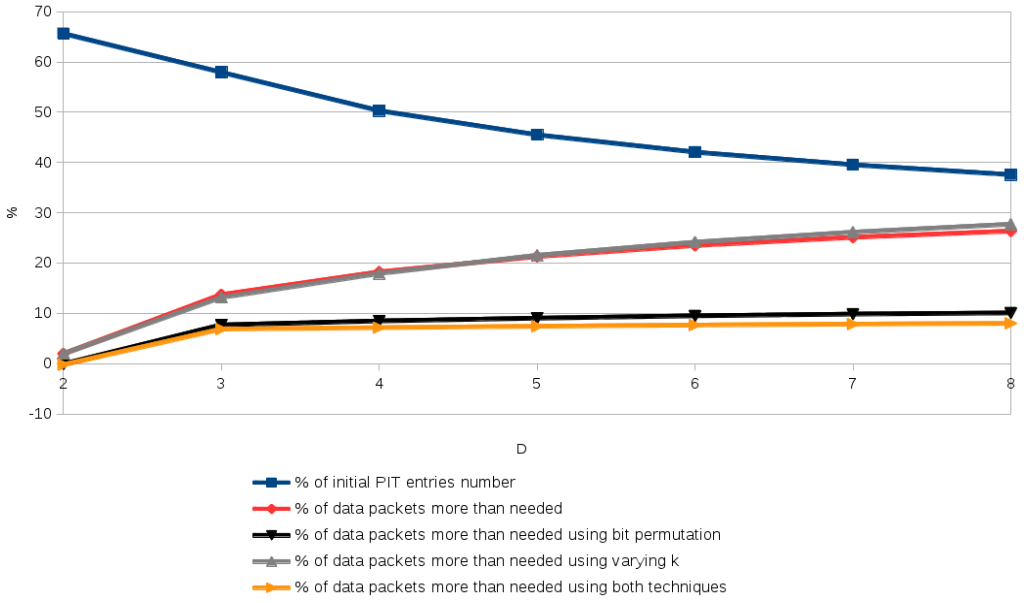
$$D = \frac{2|E|}{|V|(|V| - 1)}$$



Density seems to play a role since two topologies that have almost identical density have almost identical benefit from varying k. A topology with density 0.08 benefits the most from the varying k technique.

9 Conclusions

The techniques proposed in [2] reduce the average PIT size significantly. As d increases, the average PIT size is further reduced but more false positives occur as a result of the fact that the maximum number of LIDs that might be encoded inside a Bloom filter increases so the fill factor of many Bloom filters increases and false positives are more probable. Interest packets increase too but that is normal. The results of the evaluation of the techniques proposed in [3] show that for d=8 and using Bloom filter permutations and varying k we have the most efficient content centric network, as shown in the figure above. If varying the k parameter alone will help depends on the density of the topology. We also found that there is some kind of synergy between varying k and bit permutation since for some Ds for which varying k only harms the efficiency, the combination of varying k and bit permutations is more efficient than bit permutations alone. The size of the object name plays an important role in the PIT. If the name is too small then it's not worth the overhead of a big Bloom filter being stored along with it for small Ds.



10 References

[**1**] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, Rebecca L. Braynard, "Networking Named Content", in Proc. of the ACM CoNext 2009.

[**2**] Christos Tsilopoulos, George Xylomenos and Yannis Thomas, "Reducing Forwarding State in Content-Centric Networks with Semi-Stateless Forwarding", in Proc. of the IEEE INFOCOM 2013.

[**3**] Sarela M., Rothenberg C.E., Aura T., Zahemszky A., Nikander P., Ott J., "Forwarding Anomalies in Bloom Filter-based Multicast", in Proc. of the IEEE INFOCOM 2011.

[**4**] Matteo Varvello, Diego Perino, Leonardo Linguaglossa, "On the Design and Implementation of a wire-speed Pending Interest Table", in Proc. of the IEEE INFOCOM NOMEN Workshop 2013

[**5**] H. Dai, B. Liu, Y. Chen, and Y. Wang, "On pending interest table in named data networking," in Proc. of the ACM/IEEE ANCS, 2012, pp.211–222.

[**6**] http://www.topology-zoo.org

[**7**] http://research.cs.washington.edu/networking/rocketfuel/

[**8**] Original C Code by Steve Reid (steve@edmweb.com), Small changes to fit into bglibs by Bruce Guenter (bruce@untroubled.org), Translation to simpler C++ Code by Volker Grabsch (vog@notjusthosting.com)

[**9**] converted to C++ class by Frank Thilo (thilo@unix-ag.org) for bzflag (http://www.bzflag.org) based on: md5.h and md5.c reference implementation of RFC 1321 Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

[**10**] Fisher, Ronald A., and Frank Yates. "Statistical Tables for Biological, Agricultural and Medical Research, Oliver and Boyd, London, 1938." Note: the sixth edition is available on the web, but gives a different shuffling algo-

rithm: 26-27.

[**11**] Durstenfeld, Richard. "Algorithm 235: random permutation." Communications of the ACM 7.7 (1964): 420.

[**12**] http://www.nsnam.org/

Index