Enhancing Compact Routing in CCN with Prefix Embedding and Topology-Aware Hashing

Stefanie Roos
TU Dresden
Dresden, Germany
stefanie.roos@tu-dresden.de

Liang Wang
University of Helsinki
Helsinki, Finland
Liang.Wang@helsinki.fi

Thorsten Strufe
TU Dresden
Dresden, Germany
thorsten.strufe@tu-dresden.de

Jussi Kangasharju
University of Helsinki
Helsinki, Finland
Jussi.Kangasharju@helsinki.fi

ABSTRACT

Information-centric networks are a new paradigm for addressing and accessing content on the Internet, with Content-Centric Networking (CCN) being one of the more popular candidate solutions. CCN de-couples content from the location it is hosted and allows for mobility of the node requesting the content. However, CCN's ability to handle the mobility of the content source are limited and so far little research has focused on how both endpoints would be able to be mobile. We focus on mobility of the content source, using network embeddings as a tool. Network embeddings have already been proposed for content addressing and mobility management in prior work. In this paper, we first show that previously designed embeddings lead to a highly unbalanced storage and traffic load: More than 90 % of all stored references are mapped to one node, which is involved in more than 95% of all queries. We propose a modified embedding, *Prefix-S embedding*, and a topology-aware key assignment, which enable a uniform distribution of the storage load. The maximum traffic per node is also considerably reduced from more than 95% to 35%.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Network topology

Keywords

provider mobility; embedding; content-centric networking

1. INTRODUCTION

Information-Centric Networking (ICN) has been proposed as a solution for content delivery in the Internet, especially in terms of addressing and accessing content. In this context, challenges such as naming, caching, and routing, are of core interest and require efficient solutions. As ICN decouples content name (and how it is accessed) from the location it is stored, it would stand to reason that mobility, both for the content provider as well as consumer, would be relatively easy to achieve. However, as pointed out in [12], content provider mobility still remains one of the biggest research challenges in almost all ICN proposals [1, 7, 10, 11].

Compared to the web, ICN offers improved mobility support as existing proposals typically offer natural consumer mobility, since the system architectures behind the proposals are essentially receiver-driven. However, provider mobility, which usually requires expensive name operations (e.g., propagating updates by flooding or maintaining large databases) in the network, is extremely difficult to implement in an efficient and scalable way.

Thus, an ICN network is expected to either implement name resolution or content-based routing in order to discover and deliver content. The choice on these two approaches is a trade-off between efficiency and accuracy. Name resolution guarantees content discovery but has to maintain two databases (identifier-to-locator mapping and reachability information) at a logically centralized point and suffers from query overheads. Content-based routing, on the other hand, is more robust and efficient since it bypasses the query step but only promises probabilistic content discovery (with tunable probability) and suffers from high traffic overheads. However, as [13] showed, this tradeoff can be mediated by using a flat naming scheme with greedy routing.

In technical terms, [13] uses SHA-1 for content addressing and hyperbolic embedding technique to implement a distributed hash table (DHT) underlay in the network, further transforming every content router into a rendezvous point. Note that there are multiple ways of addressing content and hyperbolic embedding with greedy routing is merely one possible realization of compact routing.[1] Though [13] showed that their solution is superior to existing alternatives in terms of handling mobility, it fails to take into account the complexity of the embedding. As an extension to the work in [13], in this paper we focus on looking for a su-

MobiArch'14, September 11, 2014, Maui, Hawaii, USA.

http://dx.doi.org/10.1145/2645892.2645900.

[1]Compact routing refers to routing with minimal local complexity in terms of both storage and computation.

perior substitute to hyperbolic embedding, and study how content addressing should take advantage of the underlying structure of the metric space in a routing scheme. We show that a poor choice of embedding can cause a highly skewed name distribution in routing's metric space, which further may lead to severe congestion.

Specifically, our contributions in this paper are as follows:

1. We study several embedding schemes and propose Prefix-S embedding as a superior substitute to hyperbolic embedding. Prefix-S embedding provides guaranteed delivery at lower computational costs.

2. We show that content addressing with naive hashing may cause a highly unbalanced load distribution, mapping most content on one node. We propose a topology-aware hashing which takes network topological properties into account to achieve a uniformly distributed storage load.

3. We evaluate our solution thoroughly with realistic settings and show it outperforms the existing ones and achieves good system performance in terms of low overheads, high scalability and well-balanced load.

The rest of the paper is organized as follows: Section 2 describes our system model and Section 3 presents the Prefix-S and topology-aware hashing algorithms. Section 4 evaluates the proposed algorithms with different simulation settings and Section 5 discusses the related work. Finally, Section 6 concludes the paper.

2. SYSTEM MODEL

In our model, we consider an information-centric network using a flat naming scheme. While our evaluation focuses on a CCN-like network, our use of hashing of names means that the results and model apply to any ICN proposal. In the rest of the paper, we follow CCN's terminology. The network topology can be represented as a graph $G = (V, E)$ where node set V corresponds to the content routers and edge set E corresponds to the links among the connected routers. Each router v_i has a unique ID $id(v_i)$ allocated from a well-defined metric space $< M, d >$ with distance function d. The assignment $id : V \to M$ is called an embedding. In general, we want id to enable greedy routing, i.e., a path between any two nodes in V can be found by always choosing the neighbor of the currently contacted node, whose ID minimizes the distance to the target ID. Note that the distance of nodes refers to the distance of their IDs, rather than their topological distance.

Each content piece c_j is uniquely identified with a name drawn from a name space C. We denote the addressing function which maps a content name into M as $g : C \to M$. A content c_j has a designated host v_i such that $v_i = \arg\min_{\forall v_i \in V} d(id(v_i), g(c_j))$. In other words, c_j is designated to the router with the closest ID v_i in the metric space M.

Our system operation model is based on the work in [13]. However, since our concern is how embedding and hashing can influence the ID distribution in the metric space, which further impact the system performance. We simplify their model and only focus on the two aforementioned functions id and g instead of delving into practical protocol designs.

When user requests content c_i, the content name is hashed into M by calling $g(c_i)$ and embedded into the Interest packet. The Interest traverses the network with greedy routing. Each router only maintains a small table of its neighbors' coordinates. In order to forward an Interest, the router extracts destination coordinate (content ID) from its header, then it calculates the distance between the destination and each of its neighbors. The Interest is forwarded to the neighbor whose ID is closest to the destination.

The operation model defined in [13] requires that the content owner continuously registers himself at the host router so that the coming Interests can trace back. However, in our simplified model, we do not distinguish between content and content owner, and simply assume the Interest will be satisfied after arriving at its host router.

3. ALGORITHMS

In this section, we first review existing embeddings used for comparison in Section 4, before detailing our approach.

3.1 Existing Embeddings

Greedy embeddings are in general created by embedding a spanning tree into a suitable metric space. All nodes enumerate their children. The root node of the spanning tree is assigned an ID, and then each child computes its ID based on the ID of its parent and the index given by the parent.

Kleinberg's Embedding

Kleinberg's embedding into hyperbolic space enables embedding an arbitrary finite graph into two-dimensional hyperbolic space. The maximal degree m in the spanning tree needs to be known beforehand. The spanning tree is embedded into the Poincaré Disk by choosing the ID of the root node as the center of the ideal m-gon whose corners are m-th roots of unity, i.e. the complex numbers $e^{2\pi ij/m}$ for $j = 0, \ldots, m-1$. The ID of a child is obtained by applying a suitable isometric transformation, parametrized by the ID of the parent and the index of the child, of the above m-gon and choosing the center as the child's ID. Greedy routing, which always selects the closest neighbor to a desired target ID, is guaranteed to succeed [9]. Kleinberg did not treat the issue of creating suitable routing keys for content addressing. The straight-forward solution is to obtain a two-dimensional coordinate $(r(f), \phi(f))$ of the content f in polar coordinates. More precisely, let h be a hash function with a z bit image. Then $r(f)$ and $\phi(f)$ can be chosen as $r(f) = \frac{h(f)}{2^z}$ and $\phi(f) = 2\pi\frac{h(f+1)}{2^z}$.

Prefix Embedding

Prefix Embedding is an adaption of the PIE embedding [5] for unweighted graphs that has been considered for content addressing in [6]. The idea is to assign IDs using a custom metric space such that the distance between node IDs is identical to their hop distance in the spanning tree. The root is given the empty vector. A child's ID is the ID of the parent and an additional coordinate equal to the index of the child. The distance between two IDs s and t is then given as the sum of the their lengths minus twice the common prefix length $cpl(s, t)$, so

$$d(s, t) = |s| + |t| - 2cpl(s, t). \quad (1)$$

Two solutions for content addressing have been proposed in [6]. We use virtual tree construction, which is more flexible and was indicated to have slightly better load balancing properties. Here, the coordinates are bit sequences. For any node with c children, consider the maximally balanced binary tree with c leaves. The node can hence map each child to one leaf in the binary tree. The ID of the child is then the ID of the parent concatenated with bit sequence corresponding to the position of the leaf in the binary tree. Content addressing is done by hashing the content f with a z-bit hash function. The responsible node is then the node with the longest common prefix with $h(f)$. However, greedy routing need to be slightly modified because some internal nodes of the virtual binary tree do not exists. The distances of nodes represent the distances in the virtual tree rather than in the actual spanning tree, so that a node can appear closer to its sibling than its parent. Thus, if a node encounters a query for key and its ID is not a prefix of the key but it does not have any closer neighbor, the query has to be forwarded to the parent. The modified greedy routing algorithm is guaranteed to find the closest node [6].

3.2 Prefix-S Embedding

One problem with the above version of the Prefix embedding is that content is only stored at nodes with at most one child. We hence propose Prefix-S Embedding, which stores content at all nodes. Here, each node u is given two IDs, the routing ID $ID_R(u)$ and the storage ID $ID_S(u)$, a prefix of all keys stored at u. The embedding works very similarly to the virtual tree variant of Prefix Embedding. The root node gets assigned the empty vector as routing ID. When enumerating its children, an internal node u adds an additional virtual child u'. Routing IDs are assigned to the children by concatenating $ID_R(u)$ and the bit sequence of the corresponding leaf in the maximally balanced binary tree as described above. The storage ID of u is then the routing ID of the virtual node u', $ID_S(u) = ID_R(u')$. The pseudocode of Prefix-S Embedding is given in Algorithm 1, with symbol || indicating concatenation operation.

Algorithm 1 PrefixSEmbedding()

```
1: {Given: Graph G=(V,E) with spanning tree T}
2: {children(v): children of node v in T, ||: concatenation}
3: Assign ID_R(r) = ()
4: Queue q = {r}
5: while q is not empty do
6:    u = remove head of q
7:    if |children(u)| > 0 then
8:       Create balanced binary tree B of size |children(u)| + 1
9:       Create mapping map: children(u) ∪ {u} → leaves(B)
10:      for v ∈ children(u) do
11:         ID_R(v) = ID_R(u)||map(v)
12:         add v to q
13:      end for
14:      ID_S(u) = ID_R(u)||map(u)
15:   else
16:      ID_S(u) = ID_R(u)
17:   end if
18: end while
```

As for Prefix Embedding, greedy routing with the modification of forwarding to a parent if the current node's storage ID is closest to the key but not a prefix is guaranteed to work. Since the storage ID always corresponds to a leaf node in the virtual tree, a node is responsible for a key if and only if its storage ID is a prefix of key (under the assumption that keys are longer than node IDs). If a node is not responsible for a key, the responsible node can be found by forwarding to the closest neighbor if said neighbor is closer, or by contacting the parent. The forwarding decision is described in Algorithm 2.

Algorithm 2 nextHopS(BitSequence key, Node u)

```
1: {Given: Graph G=(V,E), assignments ID_R, ID_S, spanning tree T}
2: {parent(v): parent of node v in T}
3: { N(v): neighbors of node v in G}
4: if ID_S(u) is a prefix of key then
5:    routing terminated
6: end if
7: next = argmin{d(ID_R(v), key) : v ∈ N(u)}
8: if d(ID_S(u), key) < d(ID_R(next), key) then
9:    return next
10: else
11:    return parent(u)
12: end if
```

3.3 Topology-Aware Hashing

For the above embedding, leaves on the same level of the virtual tree are responsible for the same fraction of IDs. Hence, for arbitrary unbalanced trees, the storage load is expected to be unbalanced. In the following, we discuss how to create topology-aware keys, which achieve a uniform distribution over of keys over nodes for arbitrary topologies. The principal idea is that the probability that a key with prefix x has prefix $x||0$ is approximately equal to the ratio of the leaves in the left subtree rooted at node x in the virtual tree and all leaves in the subtree rooted at x. We thus compute the key of a piece of content f iteratively, as detailed in Algorithm 3. The key is a bit sequence $b_1 b_2 ... b_z$, where z is larger than the depth of the spanning tree. Assume for all possible prefixes $d_i = b_1 ... b_i$, the number of nodes v for which d_i is a prefix of $ID(v)$ is known. In case of Prefix-S Embedding, we consider the storage ID $ID_S(v)$. A hash function h is chosen with image space 2^z for some $z \in \mathbb{N}$. The values $h_i = h(f \oplus i)$ need to be known for $i = 0 \dots depth(T)$ to locate the responsible node. The $(i+1)$-th digit b_{i+1} of the file ID d is computed on basis of d_i. The recursion anchor is given by the empty string d_0. Then we have

$$b_{i+1} = \begin{cases} 0, \frac{h_{i+1}}{2^z} \leq \frac{|\{v \in V : cpl(ID(v), d_i||0) = i+1\}|}{|\{v \in V : cpl(ID(v), d) = i\}|} \\ 1, otherwise \end{cases} \quad (2)$$

with $cpl(s,t)$ denoting the common prefix length. The process continues until the set $|\{v \in V : cpl(ID(v), d) = i\}|$ is empty, so that the responsible node is uniquely identified by the key d_i. In order to distinguish different pieces of content stored at the same node and have keys of identical length, we concatenate d_i with the last $z - i$ bits of $h(f)$.

4. EVALUATION

We evaluated the approaches proposed in Section 3 using 9 different topologies of autonomous systems (AS).

4.1 Metrics and Set-up

We considered three metrics for our evaluation:

i) the fraction of stored items each node is responsible for,

Algorithm 3 ComputeKey(BitSequence f)

```
1: {Given: Graph G=(V,E), assignment ID}
2: {cpl: common prefix length, ||: concatenation}
3: {s[i...j]: bits i to j of s}
4: i = 0
5: key =''
6: while length(key) < z do
7:    if |{v ∈ V : cpl(ID(v), key) = i}| = 0 then
8:       key = key||h(f)[i + 1, ..., z]
9:    else
10:      hash = h(f ⊕ i)/2^z
11:      if hash ≤ |{v∈V:cpl(ID(v),key||0)=i+1}| / |{v∈V:cpl(ID(v),key)=i}| then
12:         key = key||0
13:      else
14:         key = key||1
15:      end if
16:   end if
17: end while
18: return key
```

ii) the traffic distribution, i.e., fraction of queries processed by each node,

iii) the number of hops needed to discover the destination of the query using greedy routing.

We evaluate the correlation between i) and ii) to see if a high storage load might be partly compensated by experiencing less traffic and vice versa. Ranking the nodes by i) and ii) provided an overview of how storage and traffic is balanced between the nodes. The evaluation was conducted as follows: We first created a spanning tree of the graph executing a breadth-first search starting a random node. Then we generated a set of $k = 10,000$ random ASCII character strings of length 20, which we used to represent the queried content. Afterwards, we computed the embedding for all considered embedding algorithms. For each embedding and each applicable key generation scheme, we then created the keys from the character strings and executed a query for each key from a random start node. Hence, a total of 5 combinations of embedding and key generation were evaluated : The Kleinberg embedding KB with hashing into the unit disk as well as Prefix P_H/P_{TA} and Prefix-S Embedding PS_H/PS_{TA} using both straight-forward hashing (H) and topology-aware (TA) keys. The relation between the storage and the traffic load on nodes is measured by the Pearson correlation coefficient. Results were averaged over 20 runs.

The hop count iii) does not necessarily correspond to the stretch, which is defined as the average ratio of the length of the routing path to the shortest path for all pairs of nodes. Since iii) considers queries, it is lower than the stretch if nodes that are easily discovered, e.g., the root, receive a disproportional high number of queries. The paths actually traversed during routing are more important for the working system, so that we choose this metric rather than the stretch.

4.2 Expectations

We expect the Kleinberg embedding to produce a very unbalanced load distribution. The root node is responsible for the majority of the unit disk when considering Euclidean space (which the hashing does) regardless of the structure of the tree. Similarly, the tree for Prefix and Prefix-S Embedding is bound to have leaves at very different levels, leading to a high storage load on those on a high level when using straight-forward hashing. The maximum storage load is likely to be even higher for Prefix-S Embedding because the virtual nodes corresponding to the storage ID of the internal nodes on the top levels are always leaves. However, for topology-aware keys the load is supposed to be uniformly distributed over all nodes (Prefix-S Embedding) or all leaf nodes (Prefix Embedding). When considering the traffic a node has to process rather than the storage load, we expect nodes on the higher levels to experience a higher load. It is not required that messages between nodes in different branches pass their common ancestor in the tree, since greedy routing also uses non-tree edges. Nevertheless, tree edges are more likely to be used, so that we expect an unbalanced traffic distribution for all spanning-tree based embeddings. Thus, there should also be a high positive correlation between traffic and storage load for the Kleinberg and Prefix-S embedding with standard hashing. Both allocate the majority of the queries and the traffic to the higher levels of the tree. When using topology-aware keys, the traffic should be uncorrelated to the uniformly distributed load for Prefix-S embedding. Prefix embedding allocates all files on leave nodes. These are rarely intermediate nodes for queries, however they frequently are destinations, so that the sign of the correlation is not immediately clear.

Previous work on greedy embeddings showed that they exhibit similarly short routes and a low stretch [2,6,9]. Potentially, the average routing length is slightly lower for Kleinberg's embedding due to high fraction of queries addressed to the root node, which is fast to route to using tree edges.

4.3 Results

We first consider the maximum load per node and the total traffic, for the 9 considered ASs. Afterwards, we analyze the distribution of the load for one exemplary AS, AS1239. In order to show the general applicability of our results, Table 1 summarizes the maximal storage and traffic load for the 9 sample ASs. In addition, the average routing length is given in order to estimate delays and the overall traffic.

Maximum Load

For the Kleinberg embedding, the storage load is always above 90%, and the fraction of traffic the most loaded node has to process is above 95%. For Prefix and Prefix-S embedding with hashing, the highest storage load and traffic are between 20% to 40%, and 50% to 85%, respectively. The maximum load is slightly higher for Prefix-S because internal nodes on high levels receive a large fraction of queries. For topology-aware keys the maximum storage load is always less than twice the average load for Prefix-S embedding, the actual load depending on the size of the AS. For Prefix embedding, the maximum load is slightly higher, because only a subset of the nodes participate in storing. The maximum traffic is drastically reduced by topology-aware keys as well, being at most 35% and 65%. Here, Prefix-S embedding has no overall advantage over Prefix embedding.

Routing Length

We also analyzed if load balancing increased the overall traffic, i.e., the number of hops needed to resolve a query. Table 1 indicates that indeed the topology-aware keys exhibit a slightly longer routing length than with Kleinberg's embedding, but the difference is mostly around half a hop in average, and at most 0.76 hops (comparing Kleinberg KB and Prefix-S PS_{TA} for AS3967). Note that the difference

was not only due to an increased stretch, since both the standard hashing and the topology-aware keys use the same topology. Rather, the reason seems to be the location of the nodes that are responsible for the queries. Prefix embedding, storing all content at leaves, in general showed the longest routes, whereas Kleinberg embedding, storing most of the content at the root, is potentially the least costly. Due to the tree structured, the shortest path to the route is found, but non-optimal routes are common for leave nodes.

We now focus on a single AS 1239 for further analysis, but emphasize that the results applied equally to the other ASes as well, but they have been excluded due to space limitations.

Storage Load Distribution

The distribution of the storage load is displayed in Figure 1a, using a cumulative distribution function (cdf) to show the fraction of files the k nodes with the highest load are responsible for. The curve shows a very steep initial increase for Kleinberg's embedding as well as for the two Prefix embeddings with a standard content addressing. By introducing topology-aware keys, the storage load is balanced uniformly, so that the increase is close to linear. The curve for Prefix embedding with topology-aware keys has a steeper slope because internal nodes with more than one child do not receive any load, whereas for Prefix-S Embedding, the load is uniformly distributed between all nodes.

Traffic Distribution

For the fraction of queries a node has to forward, i.e. the traffic per node, topology-aware keys also lessen the imbalance, but cannot abolish it (see Figure 1b). The nodes with the highest load are involved in more 35 % of all queries, which is however considerably less than for hyperbolic embeddings, for which the root node is involved in more than 98 % of the queries.

Correlation

As can be expected from these results, the correlation coefficients of the storage and the traffic load are high for Kleinberg (0.704) and Prefix-S embedding (0.629), whereas there is no notable correlation for Prefix-S embedding with topology-aware keys (0.011). For Prefix Embedding is correlation coefficient is clearly positive (0.316) for straightforward hashing and clearly negative (-0.348) for topology-aware keys. The result can be explained since leaves nodes at a high level are frequent destinations of queries as well as storing a large number of items for the standard addressing scheme, leading to a positive correlation. For topology-aware keys, items are still stored only on leaf nodes, but uniformly distributed between them. Hence none of them has a disproportionally large amount of traffic, which is reserved for the internal nodes without storage responsibility, leading to a negative correlation.

4.4 Discussion

We have shown that the poor load balance of hyperbolic embeddings can be improved. Topology-aware keys achieve a uniform storage load and reduce the traffic at congested nodes at the price of a marginally increased overall traffic. The above results indicate that Prefix-S Embedding is the best choice when combined with topology-aware keys, since it offers a uniform storage distribution over all nodes and the lowest maximum traffic. However, Prefix Embedding offers a negative correlation between storage and traffic, so that congested nodes only have to forward queries rather than answer them. Depending on the actual scenario, in particular storage and time constraints, Prefix embedding can be a better choice.

5. RELATED WORK

CCN inherently supports receiver mobility due to its receiver-driven model. To retrieve a content, the user needs to construct an Interest packet with the content name. The Interest is sent to the network and routed in a hop-by-hop manner by CCN routers to the data source unless it can be satisfied by the en-route caches. Hence the lost packet can be easily recovered by retransmitting the previous Interest. However, data source mobility still remains as a challenging open question though several solutions were proposed in the prior work [4, 8, 13], such as Sender-driven message, Rendezvous point, Indirection point and Interest Forwarding. Compared to others, [13] shows greedy routing is a promising candidate with superior performance in both handoff delay and average latency.

Greedy routing is one realization for the compact routing, and the core of a successful routing scheme heavily relies on finding a so-called *greedy embedding*. The greedy embedding preserves the property that given any destination y not directly connected to a node x, x can always find a neighbor of him who is closer to y than himself. With greedy routing scheme, a node only needs local information from directly connected neighbors in order to forward a message. Such properties make greedy routing an ideal solution in the situation where global knowlege is expensive to obtain if not possible like ad-hoc or large-scale networks. Several greedy embedding schemes with different properties were proposed in the previous work [3, 5, 6, 9].

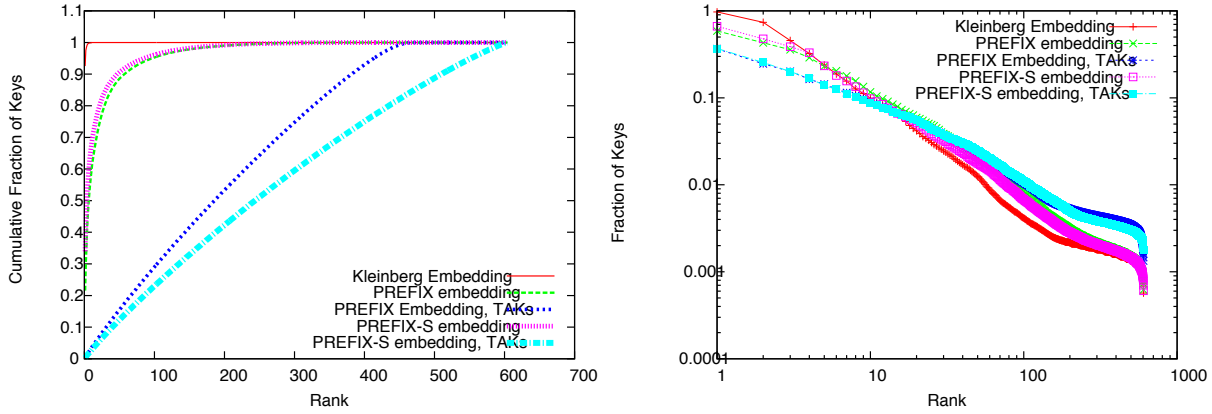
Despite of showing the possibility of solving mobility issue with greedy routing, three important facts are overlooked by [13]. First, hyperbolic embedding is a relatively expensive operation. Other embeddings that guarantee the same 100% delivery but with less overheads exist. Second, the content addressing in [13] uses simple *Sha-1*. Due to the non-uniform division in hyperbolic space, the distribution of the generated content IDs is highly skewed. The node in the Poincaré Disk center is responsible for most of the content, which causes severe congest problem. Third, the redundant links (or shortcuts) between the nodes were not well explored, results in potential unbalance traffic since most of the traffic may go through the root. We extended the work in [13] by solving the aforementioned issues. Based on our knowledge, we are the first one extensively exploring the relation between greedy embedding and topology-ware content addressing in the context of CCN.

6. CONCLUSION

In this paper, we studied the relation between the graph embedding and content addressing in the context of CCN. We showed that naive combination of the both is not only a costly solution but also causes highly skewed ID distribution in routing's metric space, further leading to storage load and congestion issues. To get resolve these issues, we proposed Prefex-S embedding to reduce the overhead and distribute the load among all nodes, and topology-aware keys

AS	Maximum Storage Load					Maximum Traffic					Average Routing Length				
	KB	P_H	P_{TA}	PS_H	PS_{TA}	KB	P_H	P_{TA}	PS_H	PS_{TA}	KB	P_H	P_{TA}	PS_H	PS_{TA}
AS1221	0.952	0.410	0.081	0.401	0.011	0.970	0.804	0.515	0.829	0.648	5.54	5.59	5.35	5.51	4.88
AS1239	0.925	0.216	0.004	0.340	0.003	0.972	0.588	0.367	0.665	0.369	4.94	5.41	5.00	5.39	5.28
AS1755	0.948	0.280	0.016	0.336	0.008	0.962	0.640	0.486	0.719	0.498	5.25	5.52	5.93	5.31	5.57
AS2914	0.950	0.270	0.003	0.289	0.002	0.972	0.699	0.348	0.763	0.350	5.90	6.22	6.36	6.01	6.20
AS3257	0.950	0.337	0.010	0.375	0.006	0.975	0.813	0.562	0.841	0.574	5.30	5.81	6.03	5.45	5.84
AS3356	0.950	0.185	0.004	0.228	0.003	0.973	0.507	0.351	0.582	0.361	3.84	4.38	4.26	4.23	4.18
AS3967	0.943	0.270	0.010	0.301	0.007	0.963	0.567	0.434	0.627	0.428	5.05	5.39	5.98	5.11	5.81
AS6461	0.922	0.351	0.117	0.389	0.052	0.953	0.638	0.575	0.698	0.567	3.34	3.55	3.57	3.42	3.41
AS7018	0.927	0.238	0.004	0.286	0.003	0.973	0.597	0.401	0.648	0.379	5.60	5.68	6.27	5.50	6.23

Table 1: Maximum load and routing length for various AS topologies with the following embedding/content addressing schemes: KB-Kleinberg Embedding, P_H-Prefix Embedding with standard hashing,P_{TA}: Prefix Embedding with topology-aware keys, PS_H-Prefix-S Embedding with standard hashing,PS_{TA}: Prefix-S Embedding with topology-aware keys



(a) Storage (b) Traffic

Figure 1: Load Distribution: a) CDF of storage by rank, b) fraction of traffic ranked

to distribute the load uniformly. We evaluated our solution thoroughly and showed it outperforms others in realistic settings.

7. ACKNOWLEDGEMENTS

This work was partially supported by the Academy of Finland in the EINS project (grant no. 275938).

8. REFERENCES

[1] C. Dannewitz, J. Golic, B. Ohlman, B. Ahlgren. Secure Naming for a Network of Information. *IEEE Global Internet Symposium*, pages 1–6, March 2010.

[2] A. Cvetkovski and M. Crovella. Hyperbolic embedding and routing for dynamic graphs. In *INFOCOM 2009, IEEE*, pages 1647–1655. IEEE, 2009.

[3] A. Cvetkovski and M. Crovella. Hyperbolic embedding and routing for dynamic graphs. In *INFOCOM 2009, IEEE*, pages 1647 –1655, april 2009.

[4] F. Hermans, E. C.-H. Ngai, and P. Gunningberg. Mobile sources in an information-centric network with hierarchical names : An indirection approach. In *Proc. 7th Swedish National Computer Networking Workshop*, 2011.

[5] J. Herzen, C. Westphal, and P. Thiran. Scalable routing easy as pie: A practical isometric embedding protocol. In *ICNP*, 2011.

[6] A. Hofer, S. Roos, and T. Strufe. Greedy embedding, routing and content addressing for darknets. In *NetSys*, 2013.

[7] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *ACM Conext*, 2009.

[8] D.-h. Kim, J.-h. Kim, Y.-s. Kim, H.-s. Yoon, and I. Yeom. Mobility support in content centric networks. In *SIGCOMM ICN workshop*, 2012.

[9] R. Kleinberg. Geographic routing using hyperbolic space. In *INFOCOM*, 2007.

[10] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. *SIGCOMM Comput. Commun. Rev.*, 37(4):181–192, 2007.

[11] Publish/Subscribe Internet Routing Paradigm. Conceptual architecture of psirp including subcomponent descriptions. Deliverable d2.2, PSIRP project. , August 2008.

[12] G. Tyson, N. Sastry, I. Rimac, R. Cuevas, and A. Mauthe. A survey of mobility in information-centric networks: Challenges and research directions. In *Mobihoc NOM Workshop*, 2012.

[13] L. Wang, O. Waltari, and J. Kangasharju. Mobiccn: Mobility support with greedy routing in content-centric networks. In *IEEE Globecom*, 2013.