iCDN: An NDN-based CDN

Chavoosh Ghasemi
University of Arizona
chghasemi@cs.arizona.edu

Hamed Yousefi
Aryaka Networks
hamed.yousefi@aryaka.com

Beichuan Zhang
University of Arizona
bzhang@cs.arizona.edu

ABSTRACT

Despite the close philosophy between content delivery networks (CDN) and named-data networks (NDN), no solution has realized a large-scale NDN-based CDN yet. In this paper, we void the popular belief that *any* NDN network can be expanded to serve as a CDN and introduce *iCDN*, a scalable, resilient, and high-performance CDN using NDN technology. We evaluate different aspects of iCDN over the Abilene topology against the global NDN testbed solution and show why iCDN is a promising design to build a large-scale NDN-based CDN.

CCS CONCEPTS

• **Networks** → **Network architectures**; **Network design principles**; **Network protocols**.

KEYWORDS

Content Delivery Networks, Named Data Networking, Information Centric Networks

1 INTRODUCTION

A content delivery network (CDN) is a distributed overlay network over the Internet that caches/stores contents as close as possible to consumers to improve content retrieval performance. A CDN understands what content each request asks for and retrieves it from either a *cache* (a network node[1] containing the content) or the *origin* (a.k.a. the content producer). Request forwarding in CDNs requires the nodes to have controlling information about (1) network changes in terms of network delays, bandwidths, failures, and topology, as well as (2) content availability in the network (i.e., what contents each node holds in its cache at any given time). To enable the nodes to learn about the network changes, the majority of CDNs employ a centralized application-layer routing module [15, 25]. This module frequently computes overlay paths by using a massive amount of statistics collected from both overlay and underlay networks. It then populates each node's forwarding information base (FIB) with a new set of overlay paths. Considering the highly dynamic nature of caches, this module cannot include the content availability information in its calculations. Thus, CDNs employ another module to track content availability on each node [3, 15]. Upon receiving a request, the nodes query this module to learn where in the network they can find the solicited content. Both of these centralized modules, however, come with arguable scalability issues [13, 15].

Taking a step back and looking at CDNs, we realize that these two modules are means to cover the limitations of the CDN forwarding plane. If the CDN forwarding plane could adapt to network changes and find in-network caches on its own, none of the aforementioned modules would be necessary. Research in information-centric networks (ICN) [7] has produced a new networking technology, called named-data networking (NDN) [31], with notable capabilities to address CDN forwarding plane's shortcomings. The NDN stateful forwarding plane lets each NDN node be intrinsically capable of caching contents and adapting to network changes on its own. Thus, one may expect that merely connecting a group of NDN nodes to each other builds a CDN where each node can cache, serve, and find contents on its own; but *it does not*.

Unlike the popular belief in the literature [18, 23] (Sec. 2), building a CDN using NDN is not a matter of borrowing *any* NDN network and adding more nodes or new features to it. If so, the current NDN networking solutions (from standard NDN deployment, i.e., the global testbed [4], to research solutions [18]) could be expanded to operate as a CDN, but they cannot. These solutions rely on a routing protocol to disseminate content availability information among the nodes. This raises two major issues. One is that the routing protocol pushes the updates for each content in the network to all nodes, whether they want it or not. Consequently, the nodes need to store these updates in their FIBs to be able to forward future requests. This approach, combined with an ever-increasing number of contents and origins in the network can rapidly lead to FIB explosion. The other issue is that the routing protocol only computes the routes from each node to the origins (excluding in-network caches). This makes utilizing on-path caches completely opportunistic and off-path caches (i.e., any node which is not on the path to the origin but has the content) impossible.

In this paper, we propose and implement *iCDN* ("*i*" for information-centric), a scalable, resilient, and high-performance NDN-based CDN which addresses the aforementioned issues in two steps: (1) *cutting the forwarding plane's dependency on routing information while fully utilizing in-network caches*: iCDN relies on CDNs' full-mesh overlay topology (Sec 3.1) to build a cache hierarchy among the nodes in a decentralized and scalable fashion. This hierarchy enables the forwarding plane to learn about content availability in the network without relying on a routing protocol. Moreover, iCDN employs a new caching policy that fits this cache hierarchy (Sec. 3.2), and (2) *enabling the forwarding plane to make efficient decisions*: iCDN introduces a new forwarding strategy, called *c-strategy* ("*c*" for CDN), with a novel probing mechanism to let the forwarding plane attentively incorporate its local measurements as well as the

[1] In this paper, a *node* refers to any packet forwarder with or without caching capability in the network.

ICN '20, September 29-October 1, 2020, Virtual Event, Canada

ACM ISBN 978-1-4503-8040-9/20/09…$15.00
https://doi.org/10.1145/3405656.3418716

information about the origins and the cache hierarchy to make efficient forwarding decisions (Sec. 3.3).

We compare iCDN with the NDN testbed solution to clarify why the idea of *any* NDN network serving as a CDN is not necessarily held. In addition, we show why achieving a high performance, scalable CDN solution requires an exclusive NDN-based network. Our simulation results show that iCDN outperforms testbed solution in terms of content retrieval delay (47%), origin workload (35%), and resiliency. We also confirm that, unlike iCDN, the testbed solution cannot scale to serve as a CDN (Sec. 4). Finally, we conclude the paper in Sec. 5.

It is worth mentioning that today's operational CDNs are profoundly complex ecosystems composed of multitudinous building blocks with a production-level fitness. The idea of replacing/replicating CDNs using NDN, although might be appealing, is indeed idealistic. That said, different features and components of these networks can be studied for improvement. We narrow the focus of this paper to *content sharing and retrieval*, a common feature across all large general-purpose CDNs. We hope that iCDN builds a concrete foundation towards a complete NDN-based CDN design and performance evaluation.

2 BACKGROUND AND RELATED WORK

2.1 NDN Stateful Forwarding Plane

In NDN, both the requests (i.e., Interests) and the responses (i.e., Data) carry the content name rather than a source/destination address. Contents in NDN are divided into small pieces, called *chunks*, where each chunk can be retrieved by an Interest.

The NDN forwarding plane comes with an in-band measurement, so each node can monitor the content retrieval performance and tune its forwarding decisions while retrieving the content. However, whether to measure or not, what to measure, and how to apply those measurements to the forwarding decisions are determined by *forwarding strategy*. Moreover, the NDN forwarding plane can detect routing loops natively and choose a different interface (i.e., next-hop) if a loop happens. As a result, an NDN node can retrieve a few chunks of a content from different interfaces simultaneously without causing a permanent loop in the network. Next, it evaluates the performance that each interface offers by combining several metrics such as round-trip time and/or the number of timeouts. Accordingly, it might rank the interfaces and decide through which one to fetch the rest of the content's chunks. This process can repeat several times during content retrieval to refine the ranking and adapt to network changes. The node then might remember the interface ranking to properly forward future requests for the same content.

2.2 Related Works

Very few papers in the literature focused on improving CDNs by using ICN's advantages. Authors in [18, 23] highlighted the benefits of employing NDN as a CDN's underlay network. They also discussed how NDN technology changes a CDN's architecture by comparing a CDN-like network with the vanilla NDN. However, the dependency of these solutions on a routing protocol hinders them from expanding into a large-scale CDN (Sec. 4). R-iCDN [22] showed the problem of sub-optimal paths in ISP-operated CDNs and addressed it by using a centralized name-based routing solution. The proposed approach, however, can lead to FIB explosion. Inaba et al. [17] used ICN technology in CDNs to involve consumers in caching the contents and serving them which is not the goal of this paper.

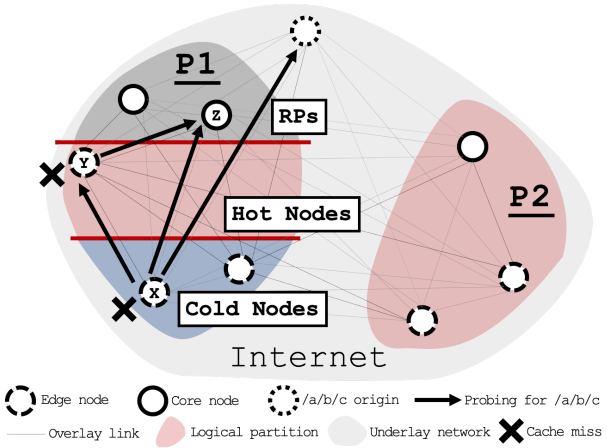

Figure 1: iCDN's logical partitions & cache hierarchy for content/namespace /a/b/c in logical partition $P1$.

There is another line of work focused on content/cache discovery via employing forwarding strategies. We can classify these strategies into three categories: routing-based [14, 20, 30], routing-free [8, 21, 29], and resolution-based [10] forwarding strategies. Routing-based strategies mandate the FIB to contain content availability information based on which they attempt to make efficient forwarding decisions. These strategies, however, can rapidly lead to FIB explosion when the number of contents grows in the network. Routing-free forwarding strategies, on the other hand, require the FIB to contain adjacency (neighborhood) information. These strategies mostly employ a flooding-like mechanism to learn about content availability in the network and can barely scale to operate at a CDN-scale network with millions of contents. Resolution-based strategies hinge on delegating one or more nodes in the network to store a collection of contents. Any node then needs a mechanism to learn/find delegated node(s) for a given content upon receiving a request. A common approach here is to use a hash function to map the name of a solicited content to a specific node. As the main limitation of these strategies, they can only exploit on-path caches. That said, c-strategy combines strengths of routing-based and resolution-based strategies by employing a cache hierarchy — built on top of a DHT — and in-partition announcements (see Sec 3.2) to resolve FIB explosion and fully support on-path and off-path caches in a scalable fashion.

3 DESIGN

In this section, we first detail the architecture and the main building blocks of iCDN and then unfold how c-strategy functions internally.

3.1 Network Architecture

Any CDN is a full-mesh overlay network, meaning that there is a one-hop virtual link between any pair of nodes even if physically they are multiple hops away from each other. Fig. 1 shows such an overlay network, where the topology of the underlay network

(i.e., the Internet) could be totally different. That said, majority of CDNs do not necessarily make all paths available to their nodes; actually, the routing module prunes the full-mesh topology and tightens each node to a small set of paths. However, in iCDN, each node can use any path on the full-mesh topology at any time. In a CDN, a node is referred to as an *edge node* if it is directly connected to a consumer; otherwise, it is a *core node*.

For network management purposes, iCDN *logically* partitions the overlay into relatively small groups of nodes that are topologically close to each other — the nodes in one partition are called *neighbors*. A partition is a logical concept that is defined by the network administrator without limiting the actual access of nodes to other parts of the overlay. For example, in Fig. 1, the logical partition to which node X belongs does not cut node X's direct virtual links to all other nodes beyond its partition. iCDN also employs a centralized module to keep track of all changes to both the overlay topology and the origins. Therefore, a node might query this module to learn about the overlay nodes (e.g., their coordinates and the partition(s) they belong to) and the origins (e.g., the contents they serve).

3.2 In-Network Caches

iCDN comes with a cache hierarchy to exploit on-path and off-path caches without relying on an application-layer routing module.

Each logical partition in iCDN forms a separate cache hierarchy among its nodes for each namespace, defined as a group of contents that share the same name prefix such as `/a/b/c`. The hierarchy is composed of three tiers, each offering a different level of guarantee on the availability of the contents for a namespace in its cache. For example, Fig. 1 shows a cache hierarchy in partition $P1$ for namespace `/a/b/c` where the chance of finding the contents of this namespace on node X is significantly lower than node Z. In the following, we detail how a hierarchy is built and works.

(1) RPs — Every CDN has a mechanism to decide upon which node(s) to offload origins contents, and iCDN is not exempted either. In iCDN, we employ consistent hashing — a well-known solution in the literature [19] — to determine the responsible node(s) for storing individual content. The implementation details of consistent hashing is not part of iCDN's design; thus, iCDN is compatible with any type of consistent hashing solution [10, 27]. The caching node that a content's name is hashed to is called that content's *RP* (rendezvous point). The main benefit of consistent hashing is that each node can find RP(s) of a given content on-the-fly via a light-weighted fast calculation without any application-layer routing information. However, as its main challenge, an RP can be too far from some nodes. This issue, though, is tackled in iCDN by bounding a cache hierarchy to a logical partition such that the maximum topological distance between a given node and an RP would be the partition's diameter. Consistent hashing also handles partition resizing (i.e., adding/removing nodes to/from a partition) without losing the consistency of contents within the partition.

The top tier in the cache hierarchy belongs to RPs that offer the highest content availability guarantee. To populate an RP's cache, we use a *pre-fetching* mechanism; instead of pushing contents of an origin to their corresponding RPs, each RP waits to receive the very first Interest for the content it is responsible for. The RP then populates its cache by retrieving a sufficient number of chunks ahead of the chunk asked by the consumer. This way, (1) consumer's Interests (except the very first ones) can be served directly from the RP's cache, and (2) RP can highly prevent populating its cache with unsolicited chunks. As a result, instead of fetching and caching the entire content, the RP only holds a small portion of the content (e.g., when a consumer asks only for a few seconds of a big video file), leaving more space for other contents in its cache. Pre-fetching implies that RPs know (or are able to learn) the name convention of each content so they can request next chunks of the content ahead of time. Learning a content's name convention is part of a broader topic in NDN/ICN, known as *name discovery* [11, 16, 24, 28]. iCDN's design, however, does not mandate using any specific type of name discovery technique/mechanism.

(2) Hot&Cold Nodes — Although the RPs and origins provide the highest content availability guarantee, in-network caches offer an excellent opportunity for a faster content retrieval. As a reminder, forwarding plane in CDNs *explicitly* find in-network caches by querying a non-scalable centralized module that keeps track of each node's cache. On the other hand, the majority of NDN-based solutions, like NDN testbed, have no knowledge about in-network caches; thus, Interests *implicitly* hit a cache on the path to the origin in an opportunistic manner. On the contrary, iCDN proposes a decentralized, scalable, and *semi-explicit* solution to find in-network caches by introducing the concept of Hot&Cold contents.

The main idea here is to let the nodes inform each other concerning the contents of their caches. However, announcing all cached contents is not practical for it incurs a significant traffic overhead. Moreover, the majority of cached contents are requested infrequently and are extremely volatile, lasting for a very short time in caches. Fortunately, we know that content popularity on today's Internet closely follows a Zipf distribution [9, 12, 26], so the majority of requests on the network are served by a relatively small portion of contents. This allows the nodes to announce names of a very small set of their cached contents to serve the majority of Interests. To determine this small set, each node keeps track of the number of received Interests for each content in a sorted list within every epoch[2]. Using a decay-function over epochs, each node chooses a small set of their most requested contents as *hot* contents at the beginning of each epoch and announces them in its partition. This announcement is a routing announcement populated within a partition, enabling each node to update its FIB pertaining to hot contents. The cached contents that are not considered hot at the beginning of an epoch are known as *cold* contents.

The second tier of cache hierarchy belongs to hot nodes which give a soft guarantee (weaker than RPs) on the availability of the contents and may experience cache misses in some cases, e.g., when they only have a portion of a hot content's chunks. The bottom tier of the cache hierarchy includes all other nodes that are neither an RP nor a hot node for the corresponding content. The only case that the network can benefit from cold nodes is when an Interest opportunistically meets one of them that has the solicited content in its cache. It is worth mentioning that the direction of requests in the hierarchy is always bottom-up; e.g., when a cold node experiences a cache miss, it can send the Interest to any other node in higher tiers (see node X in Fig.1). The cache hierarchy allows the forwarding

[2]The length of an epoch can be adjusted by the network administrator.

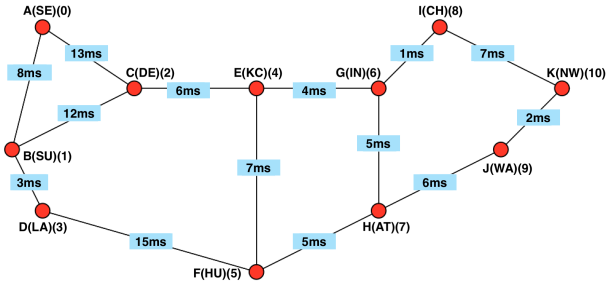

Figure 2: Abilene topology — One-way delays are derived from the geographical distance between servers.

plane to explicitly steer Interests towards the hot nodes, either on or off the path to the origin, and utilize cold nodes opportunistically. That is why we call our approach *semi-explicit*; every node explicitly knows where in its neighborhood to find hot contents while it has no information about cold contents.

Putting everything together, for a given content/namespace, every node can (1) determine the RP(s) by hashing the content's name, (2) find hot nodes in its neighborhood through neighbors' announcements, and (3) identify the cold nodes in its partition. Therefore, every node in a partition can reach exactly the same view of the cache hierarchy for a given content/namespace in a completely decentralized fashion, on-the-fly.

(3) HotCold policy — Depending on a node's location in the hierarchy, the node's cache needs to treat content's chunks, differently. If a node is an RP or hot for a content, then it needs to keep the chunks of that content for a longer time in its cache. Therefore, the nodes need to employ a new caching policy that understands three tiers of the cache hierarchy. This new policy enforces each node to split its cache space into two partitions, hot and cold. If a node receives a chunk belonging to a hot content or the node is an RP for the content, the chunk will be stored in a hot partition; otherwise, it will be stored in a cold partition. Every hot partition has a set of FIFO queues, called HOT queues each of which stores the chunks of a single hot content. A hot partition also has one LRU queue, called RP queue, to store the chunks of contents for which the node is RP.

In a cold partition, there are two LRU queues, COLD and STALE. An origin can tune the freshness of its contents to let network nodes know when to stale the origin's contents in their caches. However, the nodes can reply with stale chunks unless the consumer specifically requests a fresh version. In the entire cache space, whenever a chunk becomes stale, no matter what partition or queue it belongs to, it will immediately move to the STALE queue. If the received chunk belongs neither to a HOT queue nor to the RP queue, it will be inserted in the COLD queue. Cache eviction process follows this order: STALE > COLD > HOT > RP, where > means higher priority to evict a chunk.

3.3 C-STRATEGY

To this point, we have presented iCDN and its structure. Now, to complete the scenario, we must explore how c-strategy utilizes this network to realize an NDN-based CDN. The core of this strategy is a probing mechanism that decides how to forward Interests. To understand how this mechanism functions, we need to answer two questions: (1) *"Which nodes are the best candidates in the cache hierarchy to be probed?"*, and (2) *"How should we rank the probed nodes in order to determine the one with the best content delivery performance?"*

(1) Node selection — Upon cache miss in an edge node, the first step is to choose a set of nodes in the cache hierarchy for probing. This highly depends on the position of the edge node in the hierarchy as nodes on a given tier can only probe the higher tier(s) and origins. If the edge node is a cold or hot node, it includes at least one RP and one origin in its probing. This way, (1) c-strategy allows the RP to populate its cache if it does not have the content, increasing the overall chance of cache hit ratio in the network, and (2) c-strategy never loses the opportunity of directly fetching the content from the origin if it is the best choice. If the edge node is a cold node, it also includes topologically closest hot node in its probing. As shown in Fig. 1, edge node X is a cold node for namespace `/a/b/c`, so it probes not only the origin and an RP, but also node Y as its closest hot node. Finally, if the edge node is an RP for the solicited content and it does not have the solicited chunk, it will include the origin in its probing and populate its cache using the pre-fetching mechanism. Thus, the remaining Interests from the consumer will be served directly from RP's cache.

(2) Nodes ranking — After c-strategy on an edge node determines a set of candidate nodes, it starts probing by forwarding Interests to them. c-strategy then needs a mechanism to rank the nodes based on the packet delivery performance metrics. The node ranking mechanism follows two criteria: (1) having a quick bootstrap phase to rank the candidates and choose the best one based on the delivery performance of a few resolved Interests, and (2) considering the network changes without causing path oscillation during the content retrieval process.

To this end, we use a polynomial weighted moving average formula, including Interest-Data Round Trip Time (RTT), number of timeouts, number of resolved Interests, and estimated bandwidth of the corresponded path to rank the probed nodes. In our implementation, we can successfully provide a ranking list based on only 2 or 3 Interests — meeting our first criterion. Meeting the second one, though, requires c-strategy to consider two cases: (1) when the chosen node's performance degrades in the ranking list, and (2) when another node/path becomes available in the network with a better performance than the chosen node's. During the content retrieval, c-strategy keeps monitoring the performance of the chosen node while probing other nodes to refine its ranking list. Thus, the ranking list continuously captures network changes. However, as mentioned earlier, node ranking should highly avoid path oscillation. We have adequately tuned our ranking formula to give the chosen node higher privilege in the ranking list to prevent any path switching unless the performance of the chosen node be significantly outperformed by another probed node. Note that probing is conducted during actual content retrieval process without incurring any extra delay before the retrieval begins.

4 EVALUATION

In this section, we evaluate the performance of iCDN in terms of network and node traffic load, content retrieval delay, response

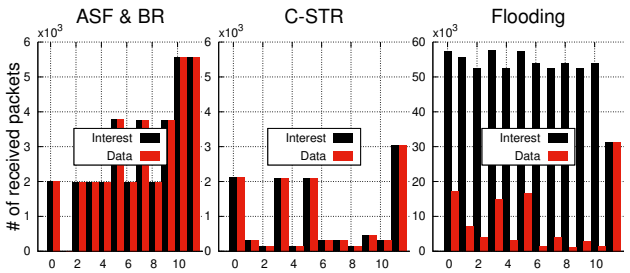

Figure 3: The overall number of received Interests and Data packets on each node for different strategies.

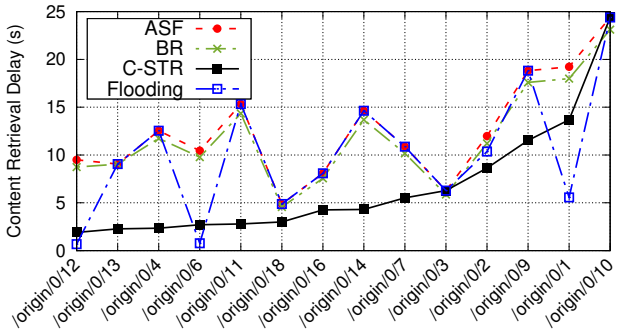

Figure 4: Content retrieval delay of each content on node 0 for different strategies.

time to network changes, and scalability via simulations using the NDN's official network simulator [5].

We compare iCDN with the most common networking solution in NDN — populating every node's FIB through a routing protocol and employing a forwarding strategy to adaptively forward Interests by choosing a next-hop from the FIB at any given time. As this solution is adopted by the NDN testbed, we refer to it as the *testbed solution*. We evaluate three different strategies in the testbed solution: (1) *ASF* [20], the de facto forwarding strategy on the testbed, (2) *Best Route* [1] (or BR for short), the other popular alternative on the testbed which also represents how the majority of existing IP-based CDNs forward packets, and (3) *Flooding* [1], that tries to represent the minimum content retrieval delay by maximizing in-network cache hit ratios. Briefly, as the main difference between ASF and BR when there is more than one available next-hop in the FIB for a given content, ASF periodically evaluates the available next-hops and attempts to choose the one with the best performance, whereas BR sticks to the next-hop with the lowest routing cost defined by the routing protocol or network administrator.

In all scenarios, we use the Abilene overlay network [2] as shown in Fig. 2. The testbed solution sees the overlay network as a multi-hop network and updates the nodes on content availability through a routing protocol. iCDN, however, sees the Abilene network as a full-mesh network and enables the nodes to learn about the content availability through the cache hierarchy. The content popularity (i.e., the request pattern) in the network follows Zipf distribution. LRU is utilized as the caching policy for ASF, BR, and Flooding. Nodes 0, 3, and 5 in Fig. 2 are the edge nodes, and the origin (node 11) is connected to node 10, serving many contents with different sizes. Cache size on each node is randomly picked from 0.1k to 1k Data packets. We employ a vanilla distributed hash table using [6] and enforce a pre-defined name convention for all contents (i.e., `/<content-name>/<version>/<chunk-number>`). For the sake of readability, we present the results from the strategies' point of view, while ASF, BR, and Flooding represent the testbed solution, and c-strategy (C-STR) represents iCDN.

4.1 Network and origin traffic loads

Fig. 3 shows the network traffic load of each approach in terms of number of received Interest and Data packets by each node. ASF and BR send Interests from node 0 hop-by-hop to the the origin over the shortest paths — provided by the routing protocol. However, as the routing protocol does not provide the forwarding plane with in-network caches information, ASF and BR resolve most of the Interests directly from the origin (node 11), causing a high origin workload. Note that no computed routes by the routing protocol pass through node 1; thus, ASF and BR incur no traffic overhead to this node. Anyways, due to the close results of ASF and BR, we show them in a single plot. In contrast, Flooding aggressively increases both the network and origin traffic loads by sending every single Interest to the entire network.

Compared to the others, c-strategy reduces both network and origin workloads for two reasons. First, c-strategy utilizes iCDN's full-mesh topology that connects each consumer to a content holder (RP, hot node, or the origin) with a direct overlay link. This substantially reduces the collective traffic load in the network (see the core nodes in Fig. 3). Second, c-strategy resolves the majority of Interests directly from RP and hot nodes instead the origin, reducing the origin workload by 35% compared to ASF and BR.

4.2 Content retrieval delay

In this section, we measure how fast edge nodes download the solicited contents. Fig. 4 shows the results for node 0, sorted according to c-strategy's retrieval delay. Nodes 3 and 5 perform very close to node 0; however, we do not report their results due to page limitations. As evident from this figure, c-strategy outperforms BR and ASF by 47% on average. This result reinforces the vital role of off-path caches — as ASF and BR can opportunistically utilize on-path caches. We can see that for almost all contents c-strategy could find a closer off-path cache to retrieve the content from. Moreover, comparing Flooding with c-strategy, we noticed an interesting observation. Initially, we expected to see that Flooding outperforms c-strategy in all cases as it tries to fetch the content from the entire network, unlike c-strategy which only uses a set of nodes. However, except for a few contents, c-strategy shows a better performance than Flooding. This is because (1) Flooding overwhelms the network with a huge amount of requests, causing a lot of unnecessary cache evictions on each node, and (2) Flooding treats all contents the same, not giving any priority to keep the popular/hot contents that are more likely to be requested in the future. This figure clearly shows the benefits of employing HotCold policy and the cache hierarchy in the network that allows nodes to populate their caches in a way that contributes to the network's overall cache hit ratio in the future.

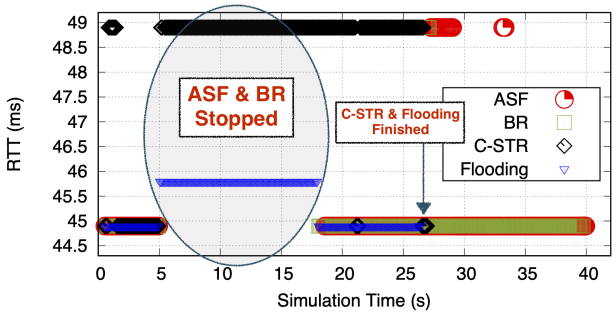

Figure 5: Monitoring Interest-Data RTT of every chunk of a content for different strategies when there is a link failure in the network from 5th to 18th seconds.

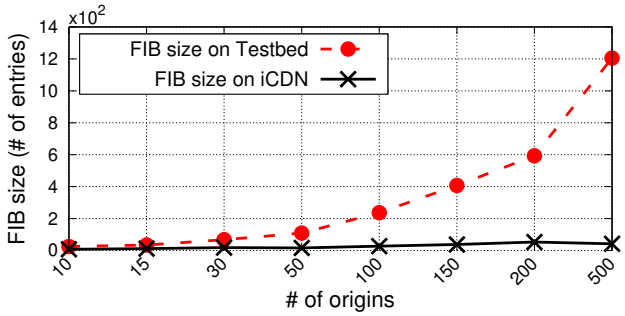

Figure 6: How FIB size of node 0 changes by increasing the number of origins in the network for the testbed and iCDN solutions.

4.3 Response time to network changes

Here, we break an important link in Fig. 2, which is shared by several paths while downloading a single content to see how fast each approach reacts to this change. The link breaks at the 5th second and comes back at the 18th second. The packet delivery completely stops while using ASF and BR during the failure period (see the oval in Fig. 5). This is because these strategies rely on the routing module to detect network changes, compute new paths, and populate each node's FIB with the new paths. This also shows that both ASF and BR are highly bounded by what routing module provides them with. Flooding and c-strategy, on the other hand, experience no interruption and continue packet delivery through another overlay path. However, there are two interesting observations: (1) RTT in Flooding is 3ms better than c-strategy during the failure. This shows that there are some cold nodes in the network that have the solicited chunks providing better performance than RP and hot nodes. Nevertheless, c-strategy is not aware of them, and (2) unlike Flooding (as well as ASF and BR) that immediately jumps back to the old path after link recovery, c-strategy postpones path switching for a few seconds. This comes from our node ranking formula that tries to guard any path oscillation. Moreover, because Flooding and c-strategy keep downloading the content's chunks during the link failure, they could complete the retrieval process almost 12s faster than ASF and BR.

4.4 Scalability

In this section, we study how iCDN and testbed solutions scale by measuring the FIB size on node 0 while increasing the number of origins in the network. In each run, the contents of a given namespace (e.g., `/origin/0`) might be served by more than one origin. Each namespace includes a large number of contents (e.g., `/origin/0/1`, `/origin/0/2`, etc.) with different sizes, and each experiment runs for 35 minutes to provide consumers with enough time to request hundreds of contents. Fig. 6 clearly shows that the testbed solution does not scale due to its strong dependency on the routing protocol. By increasing the number of origins (and accordingly number of contents) in the network, the routing protocol requires FIB on each node to store more and more information even though a given node (here, node 0) might never receive any request for some of the contents in the network. In iCDN, on the other hand, each node's FIB is populated by hot nodes' announcements. Thus, no matter how many contents the network serves, the FIB on each node keeps information of a small group of contents at any given time, which sufficiently helps c-strategy tune its decisions.

5 CONCLUSION

This paper presented iCDN, the first NDN-based CDN that exclusively uses standard NDN mechanisms for content sharing and retrieval. We showed that employing NDN technology in a CDN requires a highly scalable solution that fully utilizes in-network caches and also addresses the FIB explosion issue. We detailed the design and architecture of iCDN and showed how its cache hierarchy provides the forwarding plane with content availability information in a decentralized and scalable fashion. We then explained how our new forwarding strategy, called c-strategy, utilizes content availability information to realize a resilient, scalable, and high-performance forwarding plane for content delivery purposes.

ACKNOWLEDGMENT

We are grateful for the invaluable suggestions made by the Shepherd, Dirk Kutscher, and the comments from anonymous reviewers.

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1629009 and a Huawei grant. Any findings, discussions, and recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsor.

REFERENCES

[1] NFD developer's guide. http://named-data.net/doc/NFD/current/. [Online].
[2] Abilene Core Topology. https://stanford.io/2XAolBw, 2020. [Online].
[3] Fastly CDN. https://docs.fastly.com/en/guides/, 2020. [Online].
[4] NDN Global Testbed. https://named-data.net/ndn-testbed/, 2020. [Online].
[5] ndnSIM. http://ndnsim.net/current/, 2020. [Online].
[6] xxHash. https://github.com/Cyan4973/xxHash, 2020. [Online].
[7] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A survey of information-centric networking. *IEEE Communications Magazine*, 50(7):26–36, July 2012.
[8] M. Badov, A. Seetharam, J. Kurose, V. Firoiu, and S. Nanda. Congestion-aware caching and search in information-centric networks. In *1st ACM Conference on Information-Centric Networking*, pages 37–46, 2014.
[9] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *IEEE Conference on Computer Communications (INFOCOM)*, 1999.
[10] M. D'Ambrosio, C. Dannewitz, H. Karl, and V. Vercellone. MDHT: a hierarchical name resolution service for information-centric networks. In *ACM SIGCOMM workshop on Information-centric networking*, pages 7–12, 2011.
[11] C. Fan, S. Shannigrahi, S. DiBenedetto, C. Olschanowsky, C. Papadopoulos, and H. Newman. Managing scientific data with named data networking. In *5th International Workshop on Network-Aware Data Management*, pages 1–7, 2015.
[12] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker. Less pain, most of the gain: Incrementally deployable ICN. In *SIGCOMM*, pages 147–158, 2013.

[13] A. Flavel, P. Mani, D. Maltz, N. Holt, J. Liu, Y. Chen, and O. Surmachev. Fastroute: A scalable load-aware anycast routing architecture for modern CDNs. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 381–394, 2015.

[14] C. Ghasemi, H. Yousefi, K. G. Shin, and B. Zhang. Muca: New routing for named data networking. In *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 289–297, 2018.

[15] C. Ghasemi, H. Yousefi, and B. Zhang. Far Cry: Will CDNs hear NDN's call? In *7th ACM Conference on Information-Centric Networking*, 2020.

[16] C. Ghasemi, H. Yousefi, and B. Zhang. Internet-scale video streaming over NDN. *IEEE Network Magazine*, 2020.

[17] Y. Inaba, Y. Tanigawa, and H. Tode. Content retrieval method in cooperation with CDN and ICN-based in-network guidance over IP network. In *IEEE 40th Conference on Local Computer Networks (LCN)*, pages 454–457, 2015.

[18] X. Jiang and J. Bi. ncdn: CDN Enhanced with NDN. In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 440–445, 2014.

[19] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *29th annual ACM symposium on Theory of computing*, volume 97, pages 654–663, 1997.

[20] V. Lehman, A. Gawande, B. Zhang, L. Zhang, R. Aldecoa, D. Krioukov, and L. Wang. An experimental investigation of hyperbolic routing with a smart forwarding plane in NDN. In *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2016.

[21] T. Liang and B. Zhang. Enabling off-the-grid communication for existing applications: A case study of email access. In *IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, 2018.

[22] T. Lin, Y. Xu, G. Zhang, Y. Xin, Y. Li, and S. Ci. R-iCDN: An approach supporting flexible content routing for ISP-operated CDN. In *9th ACM Workshop on Mobility in the Evolving Internet Architecture*, MobiArch, pages 61–66, 2014.

[23] G. Ma, Z. Chen, J. Cao, Z. Guo, Y. Jiang, and X. Guo. A tentative comparison on CDN and NDN. In *2014 IEEE international conference on systems, man, and cybernetics (SMC)*, pages 2893–2898, 2014.

[24] S. Mastorakis, P. Gusev, A. Afanasyev, and L. Zhang. Real-time data retrieval in named data networking. In *1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*, pages 61–66, 2018.

[25] E. Nygren, R. K. Sitaraman, and J. Sun. The Akamai network: A platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, 2010.

[26] T. C. Schmidt, S. Wolke, N. Berg, and M. Wahlisch. Let's collect names: How PANINI limits FIB tables in name based routing. In *IFIP Networking*, pages 458–466, 2016.

[27] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on networking*, 11(1):17–32, 2003.

[28] J. Thompson, P. Gusev, and J. Burke. NDN-CNL: A hierarchical namespace api for named data networking. In *6th ACM Conference on Information-Centric Networking*, pages 30–36, 2019.

[29] L. Wang, S. Bayhan, J. Ott, J. Kangasharju, A. Sathiaseelan, and J. Crowcroft. Prodiluvian: Understanding scoped-flooding for content discovery in information-centric networking. In *2nd ACM Conference on Information-Centric Networking*, pages 9–18, 2015.

[30] W. Wong, L. Wang, and J. Kangasharju. Neighborhood search and admission control in cooperative caching networks. In *IEEE Global Communications Conference (GLOBECOM)*, pages 2852–2858, 2012.

[31] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang. Named data networking. *SIGCOMM Comput. Commun. Rev.*, 44(3):66–73, 2014.