Analysis of Greedy Hyperbolic Routing in Named Data Networking

Sean Cannan, Robert Simon
Computer Science Department
George Mason University
Fairfax, VA, USA
scannan, simon@gmu.edu

***Abstract*—Hyperbolic routing (HR) techniques are known to provide powerful and efficient packet delivery capabilities for Named Data Networking (NDN) architectures. HR is often implemented using various greedy forwarding techniques. However, the underlying implementation choices for HR are not well understood and not always well defined. This paper addresses this issue by developing a taxonomy for greedy HR and then evaluating the performance of different greedy strategies. The evaluation was conducted using a number of different methods for caching and approaches for interest collapsing. Among the results is that the standard method for returning data to a consumer – simple reverse path forwarding – is often suboptimal in terms of performance. To address this issue, this paper proposes and evaluates a more aggressive producer-to-consumer packet return mechanism. The results benefit future work using HR in NDN.**

I. Introduction

Information Centric Networking (ICN) is an emerging paradigm for routing, moving away from methods like the Internet Protocol (IP) and instead towards a system that is centered around content producers and consumers [1]. Named Data Networking (NDN) is one such specific implementation of ICNs [2]. NDN, in contrast to IP, is a data-centric model, meaning that the data itself is the focus, and connections are not end-to-end. This works by making consumers send out interest packets with the name of the data they wish to receive and producers to satisfy those requests with data packets.

One key advantage of NDN is that it is synergistic with Hyperbolic Routing (HR), which utilizes hyperbolic coordinates to do greedy forwarding [3]. Greedy forwarding works at each hop by, starting at the consumer, having a router send an interest packet to its neighboring router that is closest to the destination, the producer, in an attempt to get the data packet back as quickly as possible. Additionally, HR reduces the size of a forwarding table, as in an HR paradigm a packet only needs to know the location of the starting node and the destination node [1]. While there has been much work on HR, many greedy algorithms do not specify *how* they should be implemented, or what the performance costs and tradeoffs are for different implementations. As an example, one of the tenets of NDN is to follow the reverse of the taken path in order to save on overhead. However, for algorithms that allow cycles, this paper will show that following the exact reverse of the taken path when a cycle occurs is actually less efficient. Additionally, NDN can support interest collapsing as a method for reducing the number of interest packets in the system. If the exact reverse of the taken path is taken and interest collapsing has occurred, the packet that is meant to satisfy both the original interest and the collapsed interest would only satisfy the original interest, as the collapsed interest may not have the exact reverse path of the original interest. Making certain assumptions can lead to varying performance results, so all interpretations must be thoroughly analyzed and compared.

This paper provides a detailed analysis of how routing and packet return strategies impact system performance for HR-enabled NDN systems. This paper first constructs a HR greedy routing taxonomy. The taxonomy defines three broad categories, including basic forwarding strategy, the amount of required state information, and the return path policy. This enabled the identification of 12 variants of greedy HR. This paper then implements a simulator that is configurable by HR method, caching policy, and interest collapsing. This paper quantifies performance using several network topologies and a number of metrics, including latency, packet delivery ratio, and path stretch. Among the findings is that under many scenarios, ALL, a new packet return policy this paper defines, generally outperforms the standard packet return policy of simple reverse path forwarding. The taxonomy, simulation approach, and enhanced understanding of the impact of path return mechanisms will assist future work in deploying HR in NDN.

II. Background and Related Work

A. Named Data Networking

In NDN architecture [1], [2], each node in the network is required to have two data structures: a Pending Information Table (PIT) and a Forwarding Information Base (FIB). The PIT is responsible for logging unsatisfied requests. Whenever a request reaches a node, its incoming interface is recorded in the PIT. Additionally, its outgoing interface can be optionally recorded [4]. If the same piece of data has been requested (ie. the name is already in the PIT), only the incoming interface is recorded. The FIB, in conjunction with a Forwarding Strategy (FS), is what is responsible for choosing the outgoing interface where the interest packet will be forwarded to. This works similar to conventional forwarding models in IP, but instead of using IP prefixes, data name prefixes are used. Additionally,

979-8-3315-2096-0/25/$31.00 ©2025 IEEE

a node might have a Content Store (CS). This acts as an in-node cache, as data packets can be cached here based on a caching strategy and cache eviction policy. Since in an NDN architecture, data packets do not rely on the context of the connection, they can be cached for future use.

This architecture has many benefits, such as natural flow balance [5], removing the need for NAT traversals and address management [1], and built-in multicasting [2]. NDN has inherent flow balance by having data packets follow the reverse of the path an interest packet took, which means that each interest message has exactly one content object message [5]. This helps with congestion and control problems, as a node can simply limit the amount of items in a PIT in order to prevent incoming traffic. The need for NAT traversals and address management is also removed, as NDN does not have addressable names, instead relying on fetching the name of the data itself. Furthermore, inbuilt multicasting allows for less overhead regarding routing.

B. Hyperbolic Routing and Greedy Forwarding

Hyperbolic routing works very well with NDN as it provides stable hyperbolic coordinates, a large level of scalability, and nodes do not need to maintain a full FIB [6]. In hyperbolic space, a node needs to know its hyperbolic coordinates and the hyperbolic coordinates of its neighbors. This work uses the transformation $\theta = Atan(Lat/Long)$ and $r = \sqrt{(Lat^2 + Long^2)}$ to simulate hyperbolic coordinates based upon the topology of the network. Additionally, the hyperbolic distance x between two points at polar coordinates (θ, r) and (θ', r') is defined by the hyperbolic law of cosines [3]. This contrasts with non-hyperbolic routing, where a node needs to know its links to its neighbors, and the distances to its neighbors are measured by link weights.

Much of the work pertaining to hyperbolic routing in NDN is not focused on greedy algorithms, or at least they do not analyze the greedy algorithms on their own. Reference [6] discusses utilizing Adaptive Smoothed RTT-based Forwarding (ASF) in conjunction with hyperbolic routing. While greedy algorithms can be used in conjunction with ASF (weighting RTT and distance), greedy algorithms are not analyzed in this paper. References [7] and [8] discuss using Distributed Hash Tables (DHTs) in order to do hyperbolic routing. While these papers do suggest greedily forwarding to the correct DHT, the underlying effectiveness of greedy algorithms are not measured. References [3] and [9] discuss greedy algorithms (specifically greedy forwarding, modified greedy forwarding, and in [9] outlining gravity-pressure greedy forwarding) in-depth, however, these algorithms only appear promising when the topologies are strongly clustered or tree-like (therefore making use of hyperbolic geometry). These papers only mention that greedy forwarding and modified greedy forwarding algorithms are poor choices (0.17% and 0.21% success rate, respectively [3]) for random graphs. Furthermore, gravity-pressure greedy forwarding itself is not well defined for NDN, as it allows for cycles which conflict with NDN's interest collapsing.

III. Algorithms

To explain the taxonomy, each algorithm is classified along three dimensions: the base algorithm, the presence of dual-state tracking, and the return policy. These algorithms can be easily identified based on their name, and the naming convention is used to fully describe the algorithm.

A. Base Algorithm

Each technique has a base algorithm: either non-gravity-pressure, consisting of base greedy forwarding (GF) or modified greedy forwarding (mGF), or gravity-pressure, consisting of gravity-pressure greedy forwarding (GPGF) or gravity-pressure modified greedy forwarding (GPmGF).

Greedy Forwarding (GF): Calculates the hyperbolic distance between each of the current node's neighbors and the destination and forwards the packet to the node closest to the destination. The packet is dropped if the node closest to the destination is the current node (this signals a local minimum). This algorithm is further defined in [3].

Modified Greedy Forwarding (mGF): Calculates the hyperbolic distance between each of the current node's neighbors and the destination and forwards the packet to the node closest to the destination. However, the current node is not considered when looking at neighbors. The packet is dropped if it sends a packet to the same neighbor twice (a cycle signals a local minimum). This algorithm is further defined in [3].

Gravity-Pressure GF (GPGF): This algorithm works the same way as GF, except the packet is not dropped at a local minimum. Instead, when a local minimum is detected, calculate the current node's distance to the destination, store it in the packet, then enter pressure mode. While in pressure mode, instead of abiding by GF, a node instead forwards the packet to the neighbor it has seen the least. In the event it has seen multiple nodes the same least amount of times, forward the packet to the neighbor the node has seen the least that is closest to the destination. In order to determine the neighbor a packet has seen the least, it must track all the nodes it has visited in pressure mode, along with the number of times it has seen each node. At each node, check the current distance to the destination versus the distance recorded when pressure mode was first entered. If the current distance is smaller, leave pressure mode and resume GF. When the packet leaves pressure mode, reset the local minimum value to allow for interest collapsing. This algorithm is unique because cycles are allowed, meaning packets are never dropped if a path exists from source to destination. This algorithm is further defined in [9] and [10].

Gravity-Pressure GF (GPmGF) This algorithm works the same way as GPGF, except instead of using GF, mGF is used. Additionally, the local minimum is not reset like GPGF, instead keeping constant track of the absolute minimum.

B. Dual-State Tracking (DST)

Algorithms with DST in their name have packets track nodes visited in gravity AND pressure mode as opposed to

only in pressure mode. [1] Since the tracking is only relevant to GPGF or GPmGF, GF and mGF can never have a DST in the name. If there is no DST in the name, the packet track nodes visited only in pressure mode.

C. Return Policy

Each of the algorithms has one of two different return policies, dictating how the data packet is returned to the consumer. The options are either First-In-Last-Out or ALL.

First-In-Last-Out (FILO): FILO works by having a node forward the data packet to the most recent PIT entry associated with that packet's original consumer on the return trip. The benefit to this is the packet exactly follows the reverse of the path taken. The drawback to this algorithm is the lack of support for interest collapsing as it would not be following the exact reverse of the path taken. As an example, if consumer A is requesting content object X from producer C, and consumer B is requesting content object X from producer C, if A collapses into B therefore leading B to satisfy A's request, the packet is not following the exact reverse of the path taken.

There are two additional technicalities: the wrong PIT entry could be satisfied, and the oldest PIT entry could go unsatisfied forever. As an example, assume a line topology with three nodes: A, B, and C. Assume A is the consumer, and it is requesting data from producer C. A sends the packet (called A_1) to B, who logs it in its PIT, and sends the packet to C. Before C can generate a data packet, assume A again sends the same request to B (now called A_2). B would log this new packet in its PIT, and forward it to C. C then sends back A_1 as a data packet to B. Since the return policy is FILO, B would have A_1 satisfy A_2's PIT entry, which is technically incorrect. However, since A is still receiving the same data, and the path traversed was the same, there is no difference. Now assume A sends A_n packets before A_1's PIT entry could be satisfied, meaning that PIT entry could be stuck forever. However, since A is still receiving the data, this too does not matter.

Take Figure 1 as an example. The consumer (node 1) generates an interest that is fulfilled by the producer (node 7). 1 forwards the interest to 2 via red line 1, 2 forwards the interest to 3 via red line 2, and so on. Following the red lines, the interest packet reaches the producer (node 7). The producer generates a data packet and sends the data packet to the consumer by following the reverse of the taken path. 7 forwards the data to 6 via blue line 1, 6 forwards the data to node 2, and so on.

ALL: ALL works by having a node forward the packet to each matching PIT entry. The benefit to this is that the consumer will receive the packet sooner if cycles occur. The drawback to this algorithm is there are extraneous hops (a hop that does not contribute to the data packet getting back to the consumer on the return trip). Take Figure 2 as an example. The consumer (node 1) generates an interest that is fulfilled by the producer (node 7). 1 forwards the interest to 2 via red line 1, 2 forwards the interest to node 3 via red line 2, and so on. Following the red lines, the interest packet reaches the producer (node 7). The producer generates a data packet and sends the data packet to the consumer by satisfying all entries in the PIT entries along the way. This means that 7 forwards the data to node 6 via blue line 1 and then 6 forwards the data to 2. Once the data packet reaches 2, since there are two PIT entries, one instance of the data packet is sent to 5 via blue line 3a, and a copy of that data packet is sent to 1 via line 3b in parallel. The 3b line satisfies the request in fewer total hops than FILO, and the 3a line cleans up any lingering PIT entries.

[1]In literature "gravity" and "greedy" are sometimes used interchangeably

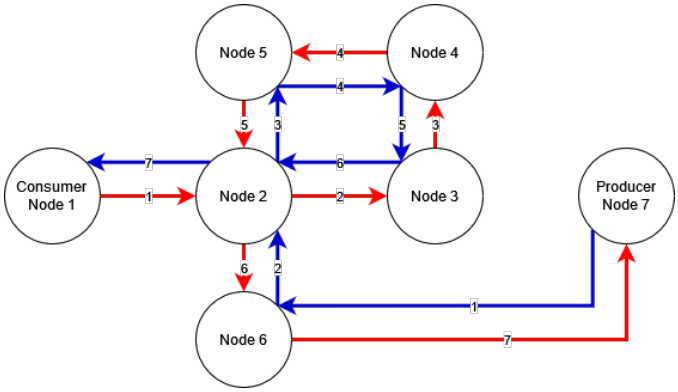

Fig. 1. FILO return policy. The red line shows the path from the consumer to the producer and the blue line shows the path taken from the producer to the consumer.

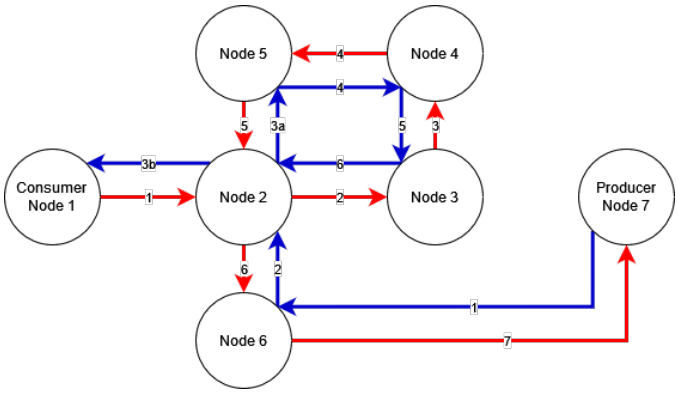

Fig. 2. ALL return policy. The red line shows the path from the consumer to the producer and the blue line shows the path taken from the producer to the consumer.

D. Forwarding Algorithms

With the three components explained, the algorithms can be defined based on all possible combinations of the components, which are listed in Table I. It should be noted that in Table I there are only 12 listed algorithms, as opposed to 16, because GF and mGF algorithms do not use DST in their name.

TABLE I
SUMMARY OF ALGORITHMS TESTED

Name	Base	Dual-State?	Return
FILO GF	GF	No	FILO
ALL GF	GF	No	ALL
FILO mGF	mGF	No	FILO
ALL mGF	mGF	No	ALL
FILO GPGF	GPGF	No	FILO
ALL GPGF	GPGF	No	ALL
FILO GPmGF	GPmGF	No	FILO
ALL GPmGF	GPmGF	No	ALL
DST FILO GPGF	GPGF	Yes	FILO
DST ALL GPGF	GPGF	Yes	ALL
DST FILO GPmGF	GPmGF	Yes	FILO
DST ALL GPmGF	GPmGF	Yes	ALL

IV. EXPERIMENTAL APPROACH

This section first explains how the simulator was designed, and then details the types of metrics it produces.

A. Simulator Design

First, the driver, the main control loop, reads in a given topology file that lists each node with each node's latitude, longitude, and neighbors. Second, the driver calculates the hyperbolic coordinates of each node, based on the latitude and longitude. Third, the driver sets up a number of "workers" (nodes in the topology) equal to the number of nodes in the topology, where each worker will listen on its own IP and port and be assigned its neighbors and hyperbolic coordinates from the previous step. Fourth, the driver iteratively selects each node pairing as an experiment. For each node pairing, each forwarding algorithm will be used to send a packet from the source node to the destination node. More specifically, the driver creates a packet with the destination set as the destination node and hands it to the source node. The source node uses the forwarding algorithm to send the packet to the destination node, and the driver is informed when the packet is successfully delivered or dropped. Results will then be aggregated and averaged across all node pairing, all nodes are shut down, and the metrics are output.

For caching, there are four different configurable options. First, multiple caching strategies (Leave Copy Everywhere (LCE), Random Caching (RAND), and No Caching (NONE)). Second, multiple cache eviction policies (First-In-First-Out (FIFO), Least Recently Used (LRU), and Least Frequently Used (LFU)). It should be noted that all caches obey normal update and TTL rules. Third, different cache sizes. 0 means no caching, .5 means a cache can hold 50% of every possible value, and 1 means the cache can hold 100% of every possible value, also meaning it has infinite size. These values are known since each node only satisfies one name meaning that the maximum number of caches that can be used is equal to the number of nodes in the topology. And fourth, packet TTL measured in seconds.

B. Metrics

In order to fully analyze the effects of different algorithms, caching, and interest collapsing, the following metrics are recorded:

- **Latency**: Measures RTT in seconds. Most of the latency here will be caused by a mixture of processing delay, and path differences.
- **Packet Success**: Measures the average percentage of packets successfully delivered. This value should always be 100% for GPGF and GPmGF algorithms since they are always successfully delivered if a path exists.
- **Path Stretch**: Measures the number of hops taken in simulation divided by the number of hops taken according to Dijkstra's Algorithm. It should be noted that this value can be less than 1, which means that there was a cache hit so the actual path was less hops than Dijkstra's Algorithm.
- **Packet Size**: Measures the size of the packet in bytes. Note that each packet has a 4-byte field for the following: its previous node, its starting node, its destination node, and to indicate if this is an interest or data packet. Each packet has a 25/26-byte name, an 8-byte field recording the start time of when the packet was sent, and an 8-byte field recording the TTL. Recall that GPGF and GPmGF algorithms at minimum require a 4-byte value signaling whether the packet is in pressure or gravity mode, and an 8-byte value signaling the distance to the destination from the local minimum when pressure mode was entered. GPGF and GPmGF also add a 4-byte value signaling the node ID and a 4-byte value signaling how many times the packet has visited this node for each node that is tracked.
- **Cache Hit Ratio**: Measures the number of cache hits as a percentage of the number of experiments.
- **Hops Collapsed**: Measures the number of hops saved by interest collapsing as a percentage of total hops.
- **Number Collapsed**: Measures the number of times interest collapsing was used as a percentage of the number of experiments.

V. RESULTS AND ANALYSIS

All of the results and analysis were conducted using each of the four topologies: GARR, a 61 node Italian network [11], GEANT, a 40 node European network [11], WIDE, a 30 node Japanese network [11], along with a 36 node NDN testbed [12].

A baseline measurement that allows interest collapsing and caching using LCE, FIFO, .5 cache size, and a 5 second TTL was recorded. Then, in order to do testing, the four configurable options for caching (caching strategy, cache eviction policy, cache size, and TTL) are treated as independent variables and were measured. Those experiments are as follows:

- Caching Strat (LCE vs RAND vs NONE)
- Cache Eviction (FIFO vs LRU vs LFU)
- Cache Size (0% vs 50% vs 100%)
- Packet TTL (0 sec vs 1 sec vs 5 sec vs Infinite seconds)

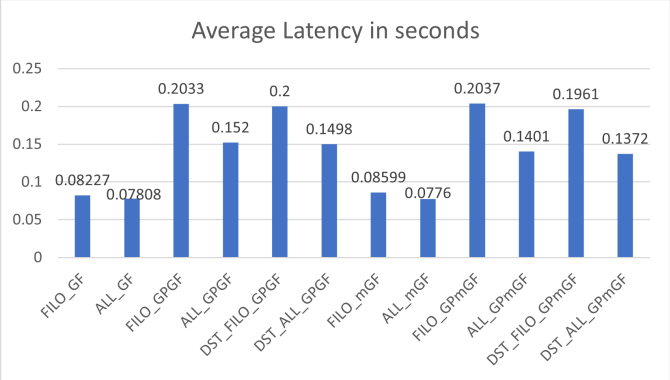

Fig. 3. Average latency measured in seconds for each algorithm.

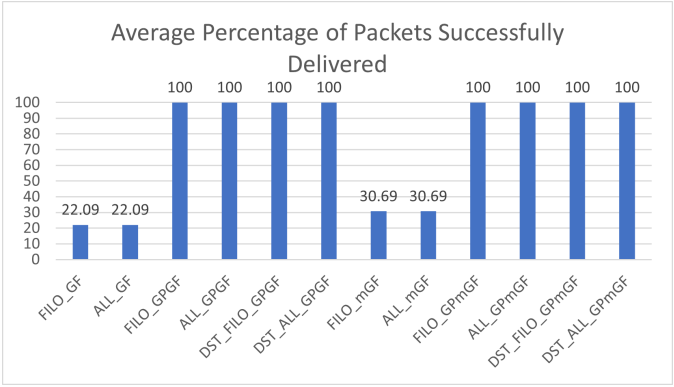

Fig. 4. Average percentage of packets successfully delivered for each algorithm.

Finally, the baseline experiment was rerun using the best values from the independent tests.

A. Baseline Results

Figures 3-9 consist of 7 different graphs, corresponding to 7 different metrics, showing the comparison of each of the different algorithms to one another for the baseline that has interest collapsing and caching using LCE, FIFO, .5 cache size, and a 5 second TTL.

Looking at the base algorithm, the figures show that non-gravity-pressure algorithms had a low latency, lower success ratio, and lower packet size compared to gravity-pressure ones. Additionally, non-gravity-pressure algorithms have a lower path stretch compared to gravity-pressure ones, however that is because they do not have 100% success when delivering a packet. The packets that do get delivered are predominantly due to cache hits, meaning the path stretch is less than 1. Furthermore, non-gravity-pressure have a lower cache hit ratio compared to gravity-pressure ones, because the packets can be dropped before a cache hit can occur.

The dual-state tracking results show that these algorithms had a similar latency, similar success ratio, and higher packet size compared to non dual-state tracking algorithms. Dual-state tracking algorithms had a slightly lower path stretch and cache hit ratio.

Looking at the return policy, the results show that FILO algorithms had a higher latency, similar success ratio, and lower packet size, compared to their ALL counterparts. FILO algorithms have a higher path stretch than their ALL counterparts because FILO algorithms cannot do interest collapsing. Furthermore, FILO algorithms have slightly higher cache hit ratio compared to their ALL counterparts. This can be attributed to the fact that ALL algorithms can interest collapse, meaning packets can be collapsed before they have a chance to get a cache hit.

B. Caching and Extended Results

For caching strategy, the experiments showed that LCE and RAND had similar cache hit ratios, so LCE was selected as optimal. Since the cache can hold one copy of every node's data, LCE means the cache can hold each node's data and never have to be evicted. Since evictions will never happen (aside from updating old copies), and since testing showed all cache eviction policies had similar cache hit ratios, FIFO was chosen as optimal for the cache eviction policy. A cache size of 1 was chosen as optimal, since testing showed a larger cache size resulted in a higher percentage of cache hits and the aforementioned synergistic properties with LCE. Finally, since testing showed that the longer the TTL, the higher the percentage of cache hits, an infinitely long TTL was chosen.

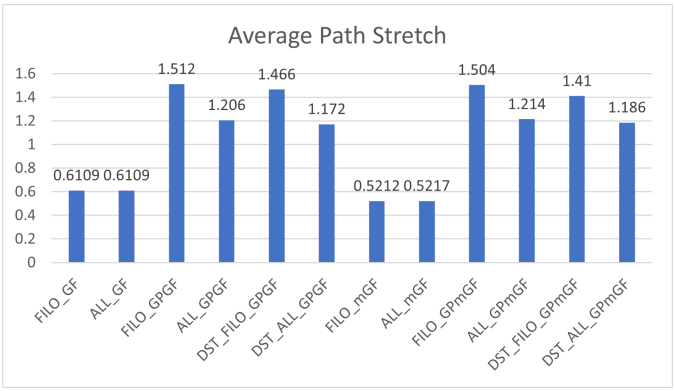

Fig. 5. Average path stretch for each algorithm.

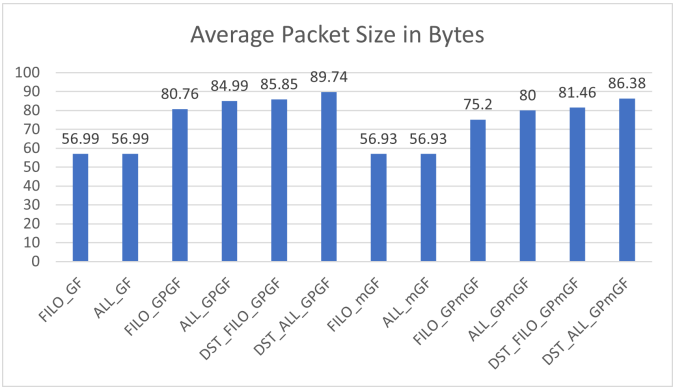

Fig. 6. Average packet size in bytes for each algorithm.

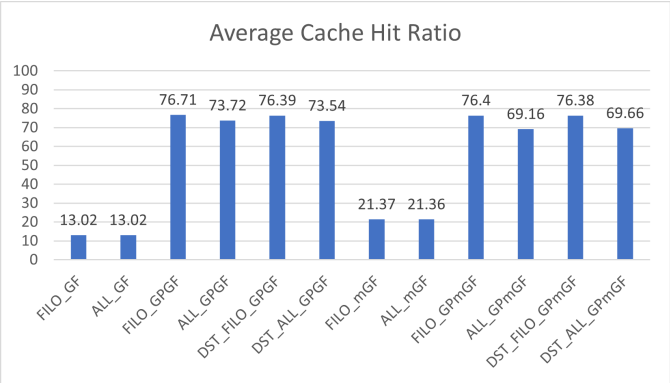

Fig. 7. Average cache hit ratio for each algorithm. Each algorithm used LCE as its caching strategy, FIFO as its cache eviction policy, a TTL of 5 seconds, and a cache size of 50%.

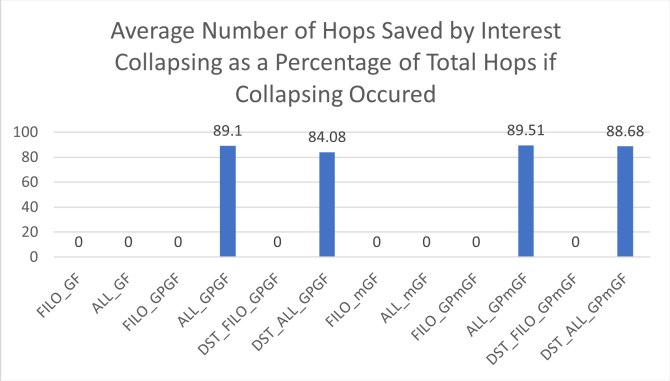

Fig. 8. Average number of hops saved by interest collapsing as a percentage of total hops if collapsing occurred for each algorithm.

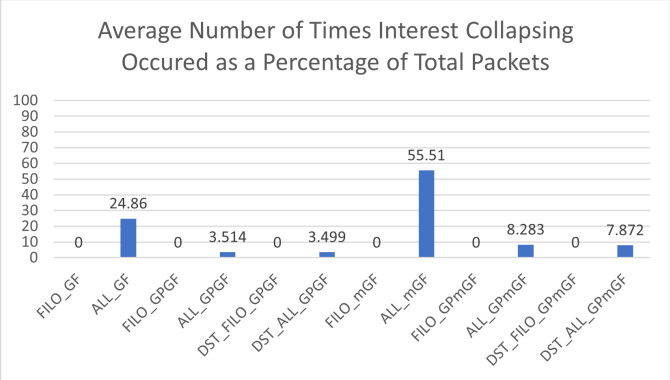

Fig. 9. Average number of times interest collapsing occurred as a percentage of total packets for each algorithm.

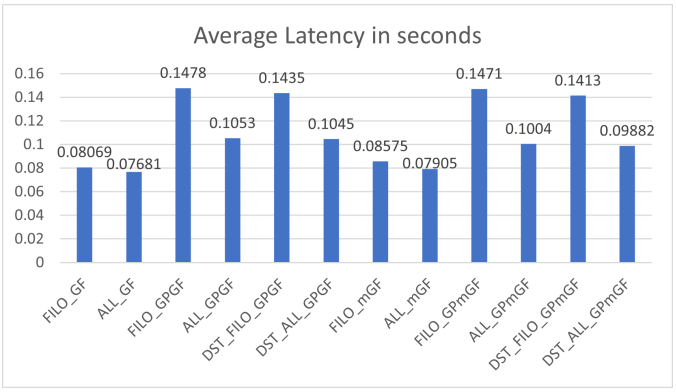

Fig. 10. Average latency measured in seconds for best results.

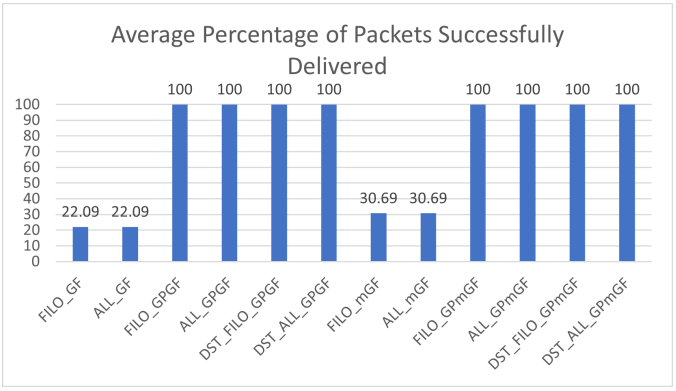

Fig. 11. Average percentage of packets successfully delivered for best results.

Figures 10-16 are the results of a comparison between the best results based on the optimal cache settings from the previous section. The results show that even with the optimal caching parameters, the trendlines for each algorithm remain the same. This means that the same conclusions drawn from the baseline algorithms remain true here.

VI. Conclusions

This paper provided a basic taxonomy for understanding greedy and gravity-pressure forwarding strategies in HR. One

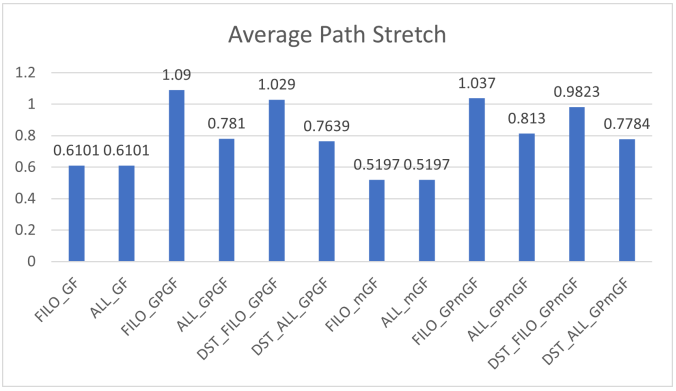

Fig. 12. Average path stretch for best results.

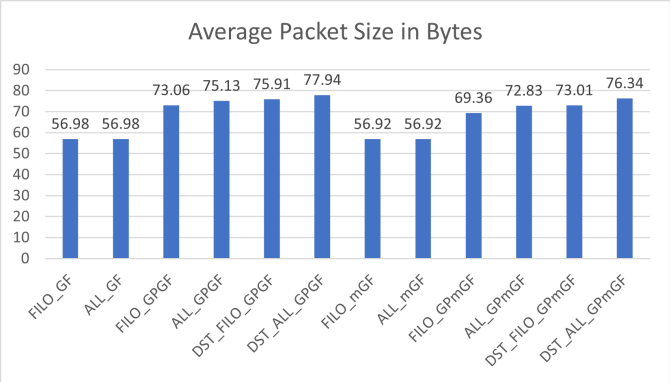

Fig. 13. Average packet size in bytes for best results.

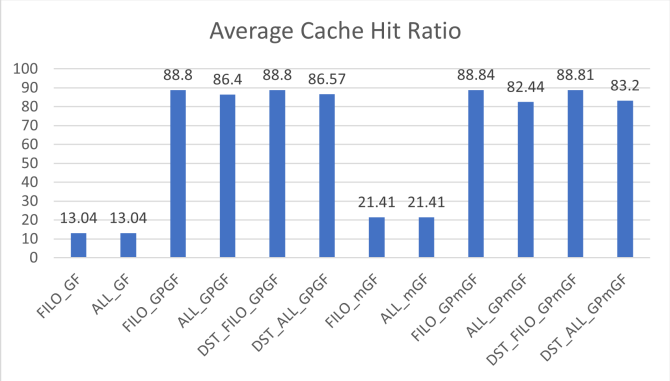

Fig. 14. Average cache hit ratio for best results, using LCE as its caching strategy, FIFO as its cache eviction policy, an infinite TTL, and a cache size of 100%.

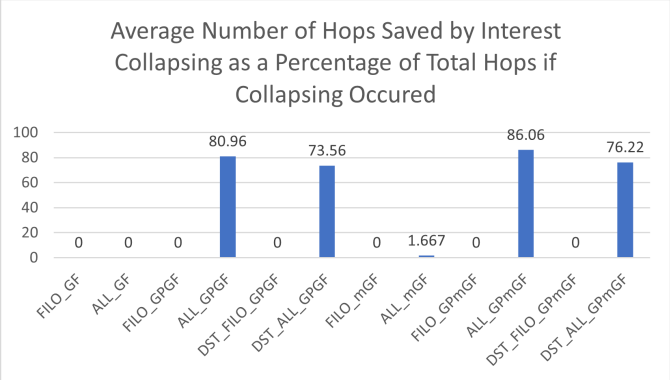

Fig. 15. Average number of hops saved by interest collapsing as a percentage of total hops if collapsing occurred for best results.

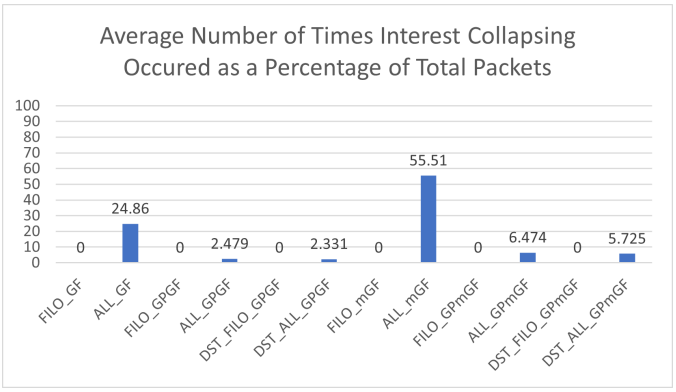

Fig. 16. Average number of times interest collapsing occurred as a percentage of total packets for best results.

of the primary results is to show that ALL algorithms outperform FILO, despite the fact that FILO is often the suggested approach. If latency is the most important metric, then ALL non-gravity-pressure algorithms perform best. The caveat is that the odds of a packet being successfully delivered corresponds with how connected the topology is or how quickly an interest can be satisfied. A more connected graph and a smaller hop total means a packet is less likely to cycle, which causes non-gravity-pressure algorithms to drop the packet. This means that caching can be critical to packet success, as it can reduce the path stretch of a packet. Finally, if packet delivery rates or path strength is the most important metric, ALL gravity-pressure algorithms perform best.

References

[1] L. Zhang *et al.*, "Named data networking," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[2] A. Afanasyev *et al.*, "A brief introduction to named data networking," in *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, Oct 2018, pp. 1–6.

[3] D. Krioukov *et al.*, "Hyperbolic geometry of complex networks," *Phys. Rev. E*, vol. 82, p. 036106, Sep 2010.

[4] B. Wissingh *et al.*, "Information-centric networking (icn): Content-centric networking (ccnx) and named data networking (ndn) terminology," 2020. [Online]. Available: https://www.rfc-editor.org/info/rfc8793

[5] M. Mosko, I. Solis, and C. A. Wood, "Content-centric networking - architectural overview," *CoRR*, vol. abs/1706.07165, 2017.

[6] V. Lehman *et al.*, "An experimental investigation of hyperbolic routing with a smart forwarding plane in ndn," in *2016 IEEE/ACM 24th International Symposium on Quality of Service*, 2016, pp. 1–10.

[7] R. Kleinberg, "Geographic routing using hyperbolic space," in *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*. IEEE, 2007, pp. 1902–1909.

[8] T. Tiendrebeogo, D. Ahmat, and D. Magoni, "Reliable and scalable distributed hash tables harnessing hyperbolic coordinates," in *2012 5th International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, 2012, pp. 1–6.

[9] F. Papadopoulos *et al.*, "Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces," in *2010 Proceedings IEEE INFOCOM*. IEEE, pp. 1–9.

[10] A. Cvetkovski and M. Crovella, "Hyperbolic embedding and routing for dynamic graphs," in *IEEE INFOCOM 2009*, 2009, pp. 1647–1655.

[11] L. Saino, C. Cocora, and G. Pavlou, "A toolchain for simplifying network simulation setup," in *6th International ICST Conference on Simulation Tools*. ICST, 2013, p. 82–91.

[12] N. Project, "Ndn testbed," 2022. [Online]. Available: http://ndndemo.arl.wustl.edu/