QuickR: A Performance-aware Routing Strategy for Wireless Mobile Information-centric Networks

Adita Kulkarni, Anand Seetharam
Department of Computer Science, SUNY Binghamton, USA
akulka17@binghamton.edu, aseethar@binghamton.edu

Abstract—In traditional mobile networks that do not support in-network caching, node mobility can cause a large number of requests to get dropped due to unavailability of paths from users to the origin servers (content custodians). In comparison, in information-centric networks (ICN) that cache content at storage-enabled network nodes in addition to the content custodians, a significantly higher percentage of requests can be served by effectively leveraging in-network caching even when direct paths are unavailable. In this paper, we propose QuickR, a routing algorithm that augments shortest path routing with features such as multi-path routing, content search, random walk, request caching and path recomputation to provide superior performance in wireless mobile ICN. We propose two versions of QuickR—i) a low-overhead version 1 that includes multi-path routing, neighbor search, random walk and request caching, and ii) version 2 that includes route recomputation along with all the other features of version 1. To demonstrate its robustness, we evaluate the performance of QuickR on diverse mobile networks—synthetic mobility models (i.e., Grid, Random waypoint), pedestrian mobility traces (i.e., Stockholm pedestrian trace) and real-world vehicular mobility traces (i.e., Rome taxi cab, Seattle bus). We also test QuickR’s performance on real-world request streams from YouTube and Wikipedia, and on multiple cache eviction and insertion policies. Our experiments demonstrate that QuickR outperforms shortest path and multi-path routing in terms of percentage of requests served by a factor of 3.5x on average.

I. INTRODUCTION

Routing in diverse mobile networks (e.g., MANETs, VANETs, DTN) has been widely investigated in the last decade in order to support a number of varied applications (e.g., wildlife monitoring, localization, tactical operations). Alongside, information-centric networks (ICN) that aim to improve user performance (e.g., delay, throughput) by serving requests for content from en route caches instead of the content custodians (origin servers) have gained popularity. This presents the opportunity to explore wireless mobile ICN and design routing strategies tailored to such networks. Revisiting routing in mobile ICN is important because effective content search strategies can improve performance by enabling a significant portion of requests to be served from in-network caches even if direct paths to content custodians are unavailable.

Although it is tempting to design a new routing algorithm for mobile ICN from first principles, before embarking on this endeavor it is important to identify the lacking in prior work, and determine how to effectively leverage existing research. Existing research in wireless mobile networks has been mainly conducted in traditional networks that do not support in-network caching [1]–[4]. Most prior work has focused on designing routing strategies for targeted environments (e.g., mesh, MANET, DTN) [5], with few papers concerning themselves with designing protocols that adapt seamlessly to changing network characteristics [3]. In-network content caching adds a new dimension to the routing problem as a significant portion of requests can be served from in-network caches even if direct paths to content custodians are unavailable.

Therefore, in this paper, instead of designing a routing algorithm from scratch, we study prior literature to identify the key features of a potent routing algorithm in wireless mobile caching environments. We identify multi-path routing and route recomputation to fix broken paths as key strategies in mobile networks. To effectively leverage caches, the key strategy is to search the caches of neighboring nodes to find if the requested content is available nearby, when the path to the custodian is unavailable. We identify two minimum overhead approaches for locating content—i) searching a node’s immediate neighbors, and ii) a simple stateless random walk strategy that probes deeper into the network. If the request still cannot be served, the user could cache the request for a certain interval of time during which time it could periodically search for content in the caches of its neighboring nodes or check if new paths to custodians are available.

Having identified the key features, we propose Quick Routing (QuickR), a routing strategy that effectively combines these features and executes on top of a network’s underlying routing algorithm to provide superior performance. We propose two versions of QuickR—i) a low-overhead version 1 that includes multi-path routing, search, random walk and request caching to demonstrate the gains obtainable by network nodes making decisions that do not require coordination or route recomputation, and ii) version 2 that includes route recomputation along with all the other features of version 1 to study the potential gains of route recomputation.

To investigate the benefits of QuickR, we conduct exhaustive experiments in diverse mobile environments. We consider structured mobility in a grid network, unstructured random waypoint mobility, synthetic Stockholm pedestrian mobility traces and real-world vehicular mobility traces (i.e., Rome taxi cab trace, Seattle bus trace). We conduct experiments on real-world request streams obtained from YouTube and Wikipedia. We also investigate the impact of cache insertion and eviction policies on the performance of QuickR. Our experiments demonstrate that both versions of QuickR outperform shortest path and multi-path routing in terms of percentage of requests served by a factor of 3.5x on average.

We also observe that the low-overhead version 1 provides
comparable performance to version 2 in a variety of different scenarios, with version 2 outperforming version 1 by up to 10%. However, as both versions employ caching of requests that cannot be served immediately, we note that certain fraction of requests can incur higher delays. But, as version 2 first attempts to serve requests by recomputing new paths before caching them, the number of requests incurring higher delays is lower in version 2 than version 1. Therefore, based on this observation, we conclude that QuickR version 1 that incurs limited overhead can be adopted in scenarios where higher delay in serving requests is acceptable, whereas QuickR version 2 can be used when requests need to be served with minimal delay and sufficient resources are available for recomputation of network paths.

II. RELATED WORK

In this section, we provide an overview of existing literature and show how it differs from our work. We first discuss routing algorithms designed for wireless mobile networks and then describe work specifically related to ICN. A number of different routing strategies have been proposed in prior work for varied mobile networks [1]–[8]. In [2], the authors propose a proactive source routing protocol to facilitate opportunistic data forwarding by maintaining network information in the form of breadth first search spanning trees at each node. Yual et al. [6] propose a joint routing and scheduling scheme based on matching theory for millimeter-wave cellular networks. A simulated annealing-based routing protocol for opportunistic mobile networks is proposed in [7], where nodes replicate messages based on a cost function. In [8], the authors propose a peer-to-peer based market-guided distributed routing mechanism for reliable data routing using base stations.

Routing protocols designed specifically for VANETs have also been proposed [9], [10]. These papers adopt mechanisms such as using PHY and MAC layer parameters for routing, selecting large vehicles as relay nodes to route request packets, and optimized route selection based on current road conditions. RAPID [5] is a forwarding protocol designed for DTN that routes packets by opportunistically replicating it while caching a copy. In [11], the authors propose an agent-based information-centric content retrieval scheme for DTN, where nodes first try to find the content source via wireless multi-hop routing, and then delegate content retrieval to the neighboring mobile nodes. Additionally, algorithms that adapt seamlessly to changing connectivity characteristics have also been proposed in prior work [3], [4].

In recent years, multiple routing approaches have also been proposed for ICN [1], [12], [13]. In [13], the authors propose a cache-aware QoS routing scheme that exploits social relationships based on node proximity, interest similarity, and node capability to forward interest packets. A distributed, content-oriented, topology-agnostic Bloom filter-based routing strategy for ICN is presented in [14], where origin servers advertise their content objects using Bloom filters. Authors in [1] propose dynamic unicast, an ICN routing protocol where nodes first broadcast interests to find the content custodian and then establish unicast paths to that custodian. Banerjee et al. propose searching caches of neighboring nodes to improve performance in a scenario of custodian unavailability [15]. Opportunistic content discovery algorithms based on broadcast requests for wireless ICN are proposed in [16], while [17] proposes a greedy routing algorithm that supports mobility in the CCN architecture. A survey of content routing algorithms for mobile ICN is available in [18]. In contrast to existing research in wireless mobile ICN, we propose a routing strategy that works seamlessly across diverse mobile scenarios with varying connectivity characteristics, and effectively leverages in-network caches to improve performance.

III. QUICKR PROTOCOL

In this section, we first describe the problem studied in this paper, provide a motivating example and then present our solution, Quick Routing (QuickR).

A. Problem Statement and Motivating Example

We consider wireless mobile ICN and evaluate the potential benefits that caching can provide in a mobile environment. To this end, we first identify the key features that maximize the number of requests served in mobile ICN by effectively leveraging in-network caches. We propose Quick Routing (QuickR), a robust routing strategy that effectively combines these features and provides superior performance in diverse networks. QuickR executes on top of a network’s underlying routing algorithm and can be integrated in existing networks with limited effort. Our goal here is not to design the optimal routing protocol, but rather to demonstrate the significant performance gains of intelligently incorporating these features in the design of a routing algorithm (as is done in QuickR).

Before describing QuickR, let us consider a simple example to demonstrate the potential benefits of content search in mobile ICN. We consider five networks, each having different mobility patterns and varying number of nodes—grid, random waypoint, pedestrian (denoted by Ped in Figure 1), taxi and bus (explained in detail in Section IV). We warm up the network caches using 50,000 requests. We assume that Dijkstra’s shortest path algorithm is used for routing requests and the cache insertion and eviction policies used are Leave Copy Everywhere (LCE) and Least Recently Used (LRU) respectively. We assume that the content universe is 50,000 and content popularity varies according to a Zipfian distribution with skewness parameter $\alpha = 0.7$ (details provided in Section IV). We measure performance by sending 200,000 requests and the results are averaged over 10 iterations. For mobility, we assume that network nodes move after every 2000 requests and shortest paths are recomputed after every 10,000 requests.

![Fig. 1: Requests serving potential for different networks](image)
Figure 1 shows the percentage of requests served by only adopting shortest path routing and the request serving potential for cache size 500. The request serving potential indicates the percentage of requests that could potentially be served at that time because the content is cached at some network node. It is evident that for a static network, the number of requests served and the request serving potential are both 100%. From Figure 1, we observe that mobility causes a large number of requests to be dropped, with the percentage of requests served by shortest path routing ranging from 3% to 35%. Interestingly, the request serving potential is significantly higher and lies between 30% to 67%. This demonstrates the need to design smart routing strategies for mobile ICN that exploit cached content.

B. QuickR Design

To design an effective routing protocol, we first outline the key principles (i.e., multi-path routing, path recomputation) adopted by routing strategies developed for traditional mobile networks, and then identify additional features (i.e., neighbor search, random walk, request caching) that can be included in mobile ICN. We then effectively incorporate these features into QuickR’s design to create a lightweight protocol that runs on top of the network’s underlying routing algorithm.

Multi-path Routing: Determining multiple paths for routing requests to the custodians instead of a single path adds limited computation overhead, but provides nodes the opportunity to route requests along alternate paths when the shortest path is unavailable. We use both shortest path and multi-path routing as baselines with which QuickR is compared.

Path Recomputation: Recomputing paths to serve requests when shortest paths break has also been explored [5]. Though recomputing paths incurs significant overhead (e.g., knowing current network state), in traditional networks that do not employ in-network caching, rediscovering new paths to the origin servers is the main approach for serving requests.

Neighbor Search: In this approach, a user broadcasts the request to all its immediate neighbors. Any node that has the requested content returns it back to the requesting node. Implementing neighbor search incurs minimal overhead as it only requires knowledge of a node’s immediate neighbors, an information that is usually readily available.

Random Walk: In this stateless forwarding approach, a node randomly picks one of its neighbor and forwards the request to it. This node first checks its cache for the content. If the content is found, it returns it to the requester, otherwise it randomly picks one of its neighbors and forwards the request. This process continues until either the content is found, in which case it is returned to the requesting node along the reverse path, or a threshold is reached and the request is dropped.

Request Caching: In this approach, if the next hop link along the shortest path is broken, a node notifies the user to cache the request for a certain interval of time. As node mobility allows nodes to come in contact with new nodes, caching a request can enable a node to serve the request by periodically searching its neighbors or when new paths to the custodians become available. After a fixed time interval, if the request is not served, it is dropped.

Having identified the key features that aid in serving more requests, we next propose two versions of QuickR that effectively combine these features.

Algorithm 1 QuickR version 1

1: route request from user to custodian along the shortest path determined by the underlying routing strategy (originalpath)
2: if any link (u,v) on originalpath broken then
3: if next shortest path from u to custodian (multi-path routing) available then
4: send request over this path
5: else if content present with immediate neighbors of u then
6: serve request
7: else if content found by random walk from u then
8: serve request
9: else
10: cache request at user
11: continue serving next request
12: end if
13: end if
14: periodically attempt to serve cached requests using content search and random walk

Version 1: When a user requests a piece of content, it is first routed using the default shortest path routing strategy. In our work, we consider that all network nodes have information regarding the shortest two paths to the custodians. Therefore, if the path taken by the default strategy is broken, the node from where the path is broken uses the second shortest path from it to the custodian (i.e., multi-path routing). If the second path is broken, this node searches for content in its neighbors by broadcasting the request to them. The neighbor that has a copy of the content returns it. If the content is not found in its neighbors, the node performs a random walk in the network to acquire the content.

If the content is still not found, the node notifies the user to cache the request. The user periodically searches its neighbors and performs a random walk in an attempt to serve the cached request. If the request is not served within a certain time interval, it is dropped. From our experiments, we find that caching the request at the user helps in serving up to 22% additional requests than caching it at the intermediate node from where the path is broken. This is because if the return path from that node to the user becomes unavailable due to node mobility, the request cannot be served even if the node is able to locate the content. Algorithm 1 provides the pseudocode for QuickR version 1.

Version 2: All steps in version 2 are same as version 1. The only difference is that before caching a request, a node recomputes paths to the custodian based on the current network state. If a path to the custodian exists, the request is sent over that path, otherwise, the request is cached in a manner similar to version 1. We note that using this version incurs significant overhead as new routes are recomputed every time. As path recomputation is an overhead-intensive process, QuickR version
2 can be used in scenarios where sufficient network resources are available for path recomputation.

**Discussion on order of exploration:** In QuickR, we conduct neighbor search followed by random walk. A natural question that arises is—is QuickR’s performance dependent on this order? We note that the ordering of content search and random walk will not impact the total percentage of requests served. This is because the total percentage of requests served is dependent on the number of network nodes explored and not on the order of exploration. The primary difference is in the number of requests served by each of the two approaches, with the approach that precedes serving a greater portion of requests. However, in terms of the number of hops needed to serve requests, searching neighboring nodes before conducting a random walk is likely to yield a lower hop count, as fewer number of requests need to be satisfied by probing deeper into the network. We verify both these claims via our experiments. Based on these arguments, in the QuickR algorithm, we opt to conduct neighbor search before performing a random walk.

**IV. Implementation Details**

To demonstrate QuickR’s widespread applicability and adaptability, we conduct experiments on the grid mobility model, random waypoint mobility model, Stockholm pedestrian mobility trace, Rome taxi cab trace, and Seattle bus trace. We also evaluate QuickR on real-world request stream traces from YouTube and Wikipedia. We also test the performance of QuickR on multiple cache insertion and eviction strategies.

**A. Mobility Models**

We next describe the mobility models used to evaluate the performance of QuickR.

**Grid Mobility Model:** We consider grid networks of multiple sizes, though the results shown here are for a 7*7 grid with 49 nodes. We assume that each node can move one step above, below, left or right from its current position with equal probability based on its location in the grid. For every network movement, each node decides to remain in its position with a probability 0.6 and move with a probability 0.4.

**Random Waypoint Mobility Model:** In this model, we consider that 60 nodes are initially uniformly distributed in a simulation area of 1000m * 1000m and each node is connected to all other nodes that lie within an euclidean distance of 100m from it. For every network movement, all nodes move to a random location within a radius r from their current position in the simulation area. We assume r = 100m.

**Stockholm Pedestrian Trace:** It contains simulation traces of pedestrians walking in downtown Stockholm covering an area of 5872 sq. m. In our experiments, we consider 300,000 location entries of 587 pedestrians. We use the first 100,000 location entries to create a base network and the remaining 200,000 location entries for mapping movement of nodes in the network. We assume that two nodes are connected if they are within a distance of 100m. We assume one network movement to include a set of 2000 entries and update the network connections based on these entries.

**Rome Taxi Cab Trace:** It contains GPS coordinates of approximately 320 taxis collected over 30 days in Rome, Italy. For our experiments, we use 300,000 location entries comprising of 162 taxis. Base network creation and movement of network nodes is the same as the Stockholm Pedestrian Trace. Following [23], we assume that two nodes are connected to each other if they are within a distance of 650m.

**Seattle Bus Trace:** It contains location entries of over 1200 city buses in the Seattle, Washington metropolitan area, covering 5100 sq. km. We use 300,000 location entries comprising of 1078 buses. Once again, the base network creation, nodes movement and network connectivity metric are the same as the Rome Taxi Cab trace.

**B. Request Stream Traces**

In addition to synthetic independent request streams that fit a Zipfian distribution with skewness parameter α, we use two real-world request streams to evaluate the performance of QuickR. The YouTube Trace was collected by monitoring YouTube traffic at the campus gateway router between June 2007 and March 2008 by the University of Massachusetts Amherst. In our experiments, we use 250,000 requests which contains 136,043 unique content. The Wikipedia Access Trace contains 10% of all user requests issued to Wikipedia from September 2007 to January 2008. We consider 250,000 requests from September 2007 in our experiments which contains 90,328 unique content.

**C. Cache Insertion Policies**

We also consider the following cache insertion policies in our experiments: In Leave Copy Everywhere (LCE) a copy of the content is placed at all nodes on the return path from the content custodian, while in Leave Copy Down (LCD) a copy of the content is placed at the node downstream towards the user for each network hit. Cache Less for More (CLAM) leverages the idea of betweenness centrality to decide whether to cache a content whereas ProbCache uses a probabilistic measure to determine which content to cache [24].

**D. Cache Eviction Policies**

We also consider the following cache eviction policies in our experiments: In Least Recently Used (LRU) the content that has not been accessed for the longest duration of time is evicted from the cache. First in First Out (FIFO) evicts content in the order in which it is inserted into a cache, while RANDOM evicts a randomly chosen content from the cache to make room for new content. In Least Frequently Used (LFU) the counter for a particular content is incremented when it is requested and the content whose counter value is least is evicted [24].

**E. Simulation Setup**

We conduct experiments on Icarus [25], a simulator designed for ICN research. Each experiment (i.e., one iteration) is conducted with 250,000 requests. Caches are warmed up with 50,000 requests and performance is measured over the remaining 200,000 requests. For a synthetic request stream, we assume that the default content universe is 50,000 and content popularity follows a Zipfian distribution with α = 0.7.
assume that all content is of equal size. We assume that the content universe is uniformly divided and permanently stored at the custodians. For the synthetic and real-world mobility traces, we randomly pick 1% of the total nodes as the custodians and 4% of the total nodes as users in every iteration. For grid and random waypoint mobility models, as the network size is small, we assume 1 content custodian and vary the number of users randomly between 2 and 6 for every iteration. All performance results are obtained as an average over 10 iterations.

The cache sizes range from 300 to 600. The default cache insertion and eviction policies considered are LCE and LRU respectively. We assume that Dijkstra’s algorithm is used for computing the shortest path(s) in the single and multi-path routing strategies. Recall that QuickR adopts a random walk approach to search for content. We limit the random walk hop count to 4 in our experiments. We assume that the nodes move (i.e., one network movement) after a fixed number of requests (say $x$) and shortest paths from users to custodians are recalculated after every $y$ requests. In order to serve the cached requests, we assume that the user searches its neighboring nodes after every $x$ requests. If the request still cannot be served after $y$ requests, it is dropped. The default parameters are $x = 2000$, $y = 10,000$. Therefore, paths are recomputed after every 5 network movements. As the network moves after every 2000 requests and we consider 200,000 requests for performance evaluation in our simulations, the network moves 100 times over the entire simulation period. Our main performance metric is the percentage of requests served as it demonstrates QuickR’s capability to serve content during path unavailability. We also consider in-network hits and the additional delay incurred by caching requests within the network.

V. EXPERIMENTAL RESULTS

In this section, we discuss the performance of QuickR with respect to the different mobile network traces, request stream traces and caching strategies outlined in Section IV. To facilitate understanding, we first discuss results for grid networks and then proceed to other networks. We note that most of the insights obtained by analyzing the grid network hold true for other mobile settings. Our experiments show that QuickR consistently outperforms shortest and multi-path routing by a factor of $3.5x$ on average.

A. Performance Results

Grid Network: We begin our study by investigating the benefits of separately including the features discussed in Section III (Figure 2a). Each of these features is executed individually on top of the shortest path routing algorithm. We also present results for shortest path routing alone in Figure 2a to highlight the contributions of these individual features. This study isolates the performance impact of individual features and helps in identifying the most important features. We observe that using shortest path routing alone serves only 33% of the total requests on average. We observe that multi-path routing provides approximately 1.5% performance improvement over shortest path routing. This occurs because the alternate route selected to send the request in multi-path routing is calculated based on the prior network state and is thus likely to be broken due to node mobility.

In comparison, searching the caches of a node’s immediate neighbors or conducting a random walk serves around 11% and 8% additional requests over shortest path routing respectively. We observe that as the network cache increases, the benefit (i.e., the number of requests served) of searching content increases. This is because, as the cache size increases, more content is likely to be available in in-network caches, resulting in larger number of requests getting served. We observe that adopting path recomputation serves an additional 20% requests over shortest path, and caching requests and serving them later serves an additional 30% requests on average irrespective of the network cache size. This demonstrates that caching requests within the network has significant benefits with some of these cached requests getting served when new paths to the content custodian become available. As path availability is dependent on network mobility and not network cache size, we observe similar performance for all cache sizes.

Having studied the performance benefits of the individual features, we next investigate how these combined features provide superior performance in QuickR. Figure 2b shows the comparison of our proposed algorithms with respect to shortest path and multi-path routing strategies for different cache sizes. We observe that QuickR significantly outperforms these baseline strategies, serving approximately 37% more requests than them. We also see that immediately recomputing paths in QuickR version 2 serves around 7% additional requests than the lightweight QuickR version 1.

In Figure 2c, we investigate the total number of in-network hits served by intermediate nodes. We see that shortest path routing has the lowest number of in-network hits followed by multi-path routing with both versions of QuickR having approximately 4x more in-network hits than shortest path alone. This once again illustrates that QuickR’s approach of searching neighboring caches for content and performing a random walk when the shortest path is unavailable is successful in serving a large number of requests. We observe that QuickR version 2 obtains a lower number of in-network hits in comparison to version 1. This is understandable because QuickR version 2 attempts to serve requests by recomputing new paths to custodians before caching the requests. We observe in QuickR version 2 that majority of requests sent over the newly computed paths are served by the custodians rather than en route caches.

Figure 2d shows the average number of hops taken by every request before it gets served for the four routing strategies. We see that QuickR incurs higher number of hops than the baseline strategies. This is expected because QuickR might unsuccessfully attempt to serve cached requests multiple times by searching neighboring nodes when network nodes move before finally succeeding. We consider these unsuccessful attempts to contribute toward the hop count for requests in QuickR.

Other Networks: We next discuss the performance of QuickR for the other networks mentioned in Section IV. Table I shows the percentage of requests served when each key approach is run individually on top of shortest path routing. Similar to grid networks, we find that multi-path routing provides the
least performance improvement for all networks over shortest path routing, with the lowest being for the random waypoint mobility model (less than 1%) and highest being for the pedestrian network (around 3%).

### TABLE I: Average Percentage of Requests Served

<table>
<thead>
<tr>
<th>Networks</th>
<th>Shortest Path</th>
<th>Multi-path</th>
<th>Neighbor Search</th>
<th>Random Walk</th>
<th>Compute Paths</th>
<th>Request Caching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid</td>
<td>32.90%</td>
<td>34.69%</td>
<td>44.12%</td>
<td>41.32%</td>
<td>52.63%</td>
<td>62.58%</td>
</tr>
<tr>
<td>Random</td>
<td>6.82%</td>
<td>6.83%</td>
<td>22.29%</td>
<td>18.42%</td>
<td>7.14%</td>
<td>7.84%</td>
</tr>
<tr>
<td>Pedestrian</td>
<td>34.27%</td>
<td>37.16%</td>
<td>53.32%</td>
<td>43.27%</td>
<td>77.79%</td>
<td>79.75%</td>
</tr>
<tr>
<td>Taxi</td>
<td>3.37%</td>
<td>3.61%</td>
<td>12.48%</td>
<td>10.75%</td>
<td>8.52%</td>
<td>10.62%</td>
</tr>
<tr>
<td>Bus</td>
<td>13.54%</td>
<td>14.57%</td>
<td>27.17%</td>
<td>23.74%</td>
<td>20.14%</td>
<td>21.23%</td>
</tr>
</tbody>
</table>

We make an interesting observation that searching content in the caches of immediate neighbors gives the highest benefit in all networks except the pedestrian network. Content search serves around 15%, 19%, 9% and 14% additional requests for random waypoint, pedestrian, taxi and bus networks respectively. The next key strategy that provides high performance for all networks except the pedestrian network is random walk. It serves around 11%, 9%, 7% and 10% additional requests for random waypoint, pedestrian, taxi and bus networks respectively. This demonstrates that leveraging in-network caching to serve requests when paths are unavailable can significantly enhance performance. Additionally, we observe that for the pedestrian network, request caching for a certain time interval gives the highest performance with a 45% increase. This is because as the pedestrian network is densely connected, broken paths tend to get re-established at a faster rate, which in turn enables request caching to serve a high percentage of requests.

After investigating the contribution of each of the individual factors for the various networks, we evaluate QuickR’s performance with shortest path and multi-path routing. From Figure 3, we see that QuickR outperforms the baseline routing strategies for all the networks, approximately serving an additional 17%, 54%, 20% and 24% requests for random waypoint, pedestrian, taxi and bus networks respectively. We make three additional observations. First, comparatively less percentage of requests are served in the random waypoint and taxi networks. This is because both these networks are sparsely connected networks, thus resulting in a large number of requests being dropped. Second, for the pedestrian, taxi and bus networks, we see that QuickR version 2 outperforms version 1. Version 2 serves an additional 10%, 3.5% and 2% requests for the pedestrian, taxi and bus networks respectively. Third, in case of the pedestrian network, QuickR is able to serve approximately 92% requests on average. As stated earlier, the pedestrian network is a densely connected network which increases the probability of finding content in neighboring caches or finding new paths to the custodians when old ones break.

We next discuss the impact of altering the order of neighbor search and random walk in QuickR. We observe that irrespective of the order, the total percentage of requests served is the same. The main difference lies in the contribution of the individual approaches. For example, for a grid network with cache size 500, we observe from Figure 2b that QuickR version 1 serves around 70% of total requests. If we perform neighbor search followed by random walk, then approximately 18% requests are served by neighbor search and around 1% requests are served by random walk. In comparison, if we reverse the order, 14% requests are served by random walk and around 5% requests are served by neighbor search. We observe similar results for all other networks. We also observe lower hop counts if neighbor search is conducted first, which is evident from the numbers presented above.

### B. Discussion on varying network parameters

In this subsection, we study the performance impact of varying the different network parameters on QuickR. Due to lack of space, we only present results for the taxi network for varying the different network parameters on QuickR. Due to lack of space, we only present results for the taxi network.
Fig. 4: Taxi: Performance impact of varying network parameters

Skewness Parameter $\alpha$: Figure 4a shows the results of varying the skewness parameter $\alpha$ of the Zipfian distribution from 0.4 to 1.0 keeping all other parameters unchanged. We observe that QuickR outperforms the baseline approaches for all $\alpha$ values. We see that as the $\alpha$ value increases, the percentage of requests served by QuickR also increases. This is because as the value of $\alpha$ increases, the content popularity skewness also increases that results in fewer number of content being far more popular than the rest. This entails that more requests get served as popular content is always present within network.

Content Universe: Figure 4b shows the results of varying the content universe from 20,000 to 80,000 keeping all other parameters unchanged. We observe that QuickR outperforms other strategies for all content universe sizes. Additionally, QuickR serves more requests for smaller content universe sizes. As the total network cache size remains the same, the chance of serving more requests increases for smaller content universe.

Cache Insertion Policies: In Figure 4c, we compare the performance of the routing algorithms for the different cache insertion policies outlined in Section IV. We once again see that QuickR significantly outperforms other strategies for all cache insertion policies. We also observe that performance of QuickR is better for LCD, CL4M and ProbCache than LCE. This is because in comparison to other strategies that carefully place content in the caches of a few nodes to improve network content diversity, LCE caches content on all nodes on the return path of the request, thus resulting in reduced performance.

Cache Eviction Policies: In Figure 4d, we compare the performance of routing algorithms with respect to other widely accepted cache eviction policies such as LFU, FIFO and RANDOM. We see that QuickR significantly outperforms the baseline routing strategies, with the highest performance being for LFU. This is because most popular content is always present in the caches for the LFU cache eviction policy. As popular content can get evicted from the caches for the LRU, FIFO and RANDOM policies, LFU provides the best performance.

C. Discussion on Real-world Request Streams

We next investigate QuickR’s performance on real-world YouTube and Wikipedia request stream traces (Figure 5). We once again only present results for taxi network. We observe that QuickR outperforms shortest and multi-path routing for both Wikipedia and YouTube request streams. We see that QuickR serves a larger portion of requests for Wikipedia in comparison to YouTube. We identify the higher content popularity skewness for the Wikipedia trace that results in popular content being available within the network as the primary reason behind this performance difference.

D. Discussion on varying extent of mobility

In this subsection, we study the performance of QuickR by varying the extent of network mobility. Figure 6 shows the performance of routing strategies for grid and taxi networks for cache size of 500. We assume that nodes move after every 500, 1000, 2000 and 5000 requests. We still recompute network routes after every 10,000 requests, i.e., after every 20, 10, 5 and 2 network changes respectively. We again see that QuickR significantly outperforms the baseline routing strategies. For all networks, we observe that the performance of shortest path and multi-path routing decreases as network mobility increases. This is because increasing network mobility causes shortest paths between users and custodians to be available for a shorter time duration. In comparison, as increased mobility enables QuickR version 2 to find new paths to the custodians during path recomputation, its performance increases. We again observe that QuickR version 2 outperforms version 1.

E. Discussion on Delay

In this subsection, we study the delay in serving cached requests for the two versions of the QuickR protocol (Table II). We define delay as the number of network changes that occur before a request is served. We observe that both versions
of QuickR require 2 to 3 network changes on average to serve cached requests, with version 2 incurring lower delay than version 1. The two exceptions are seen for the grid and pedestrian networks where QuickR version 2 requires only 1.93 and 1.64 network changes, respectively. Path recomputation for each network change coupled with the fact that the pedestrian network is a densely connected network results in QuickR version 2 having a lower delay than version 1 for the pedestrian network. The densely connected nature of the pedestrian network enables QuickR version 2 to find new paths to the custodians when old ones break.

Table II also shows the total number of cached requests served. We note that it is these cached requests that incur delays as they are cached for 2 to 3 network changes before getting served. We observe that as QuickR version 2 first recomputes new paths before caching requests, it ends up caching a significantly lesser number of requests in comparison to version 1, which in turn leads to lesser number of cached requests getting served. From our experiments, we conclude that QuickR version 1 that incurs minimal overhead can be adopted in scenarios where higher delay in serving requests is acceptable, whereas QuickR version 2 can be used when requests need to be served with minimal delay and sufficient resources are available for recomputation of network paths.

### TABLE II: Average delay in serving cached requests

<table>
<thead>
<tr>
<th>Mobile Networks</th>
<th>Quick Routing</th>
<th>Cache Size</th>
<th>300</th>
<th>500</th>
<th>#Cached Requests Served</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>300</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grid Version 1</td>
<td>2.41</td>
<td>2.41</td>
<td>49790</td>
<td>47585</td>
<td></td>
</tr>
<tr>
<td>Version 2</td>
<td>1.93</td>
<td>1.92</td>
<td>31029</td>
<td>29730</td>
<td></td>
</tr>
<tr>
<td>Random Version 1</td>
<td>2.98</td>
<td>2.97</td>
<td>3220</td>
<td>3524</td>
<td></td>
</tr>
<tr>
<td>Version 2</td>
<td>2.97</td>
<td>2.97</td>
<td>3024</td>
<td>3310</td>
<td></td>
</tr>
<tr>
<td>Pedestrian</td>
<td>2.59</td>
<td>2.58</td>
<td>63834</td>
<td>57371</td>
<td></td>
</tr>
<tr>
<td>Version 2</td>
<td>1.64</td>
<td>1.64</td>
<td>25669</td>
<td>23729</td>
<td></td>
</tr>
<tr>
<td>Taxi Version 1</td>
<td>2.90</td>
<td>2.89</td>
<td>17483</td>
<td>18126</td>
<td></td>
</tr>
<tr>
<td>Version 2</td>
<td>2.85</td>
<td>2.84</td>
<td>16346</td>
<td>17060</td>
<td></td>
</tr>
<tr>
<td>Bus Version 1</td>
<td>2.89</td>
<td>2.88</td>
<td>15513</td>
<td>15967</td>
<td></td>
</tr>
<tr>
<td>Version 2</td>
<td>2.88</td>
<td>2.87</td>
<td>11671</td>
<td>12354</td>
<td></td>
</tr>
</tbody>
</table>

### VI. CONCLUSION

In this paper, we proposed QuickR, a routing strategy that executes on top of a network’s underlying routing algorithm and provides superior performance in diverse network environments. We proposed two versions of QuickR—the first version incorporates features such as multi-path routing, search, random walk and request caching while the second version includes route recomputation along with all the other features of version one. Via extensive experimentation, we demonstrated that both versions of QuickR provide significant improvement, outperforming baseline shortest path and multi-path routing algorithms in a variety of different network settings by a factor of 3.5x on average, with QuickR version 2 providing up to 10% improvement on version 1. Based on our experiments, we conclude that the low overhead QuickR version 1 can be used in scenarios where delay caused due to caching requests is acceptable while QuickR version 2 is beneficial in settings where requests need to be served with minimal delay and resources are available for frequent path recomputation.

### REFERENCES