HPRoP: Hierarchical Privacy-Preserving Route Planning for Smart Cities

FRANCIS TIAUSAS, Nara Institute of Science and Technology (NAIST), Japan
KEIICHI YASUMOTO, Nara Institute of Science and Technology (NAIST), Japan and RIKEN Center for Advanced Intelligence Project (AIP), Japan
JOSE PAOLO TALUSAN, Vanderbilt University, USA
HAYATO YAMANA, Waseda University, Japan
HIROZUMI YAMAGUCHI, Osaka University, Japan
SHAMEEK BHATTACHARJEE, Western Michigan University, USA
ABHISHEK DUBEY, Vanderbilt University, USA
SAJAL K. DAS, Missouri University of Science and Technology, USA

Route Planning Systems (RPS) are a core component of autonomous personal transport systems essential for safe and efficient navigation of dynamic urban environments with the support of edge-based smart city infrastructure, but they also raise concerns about user route privacy in the context of both privately-owned and commercial vehicles. Numerous high profile data breaches in recent years have fortunately motivated research on privacy-preserving RPS, but most of them are rendered impractical by greatly increased communication and processing overhead. We address this by proposing an approach called Hierarchical Privacy-Preserving Route Planning (HPRoP) which divides and distributes the route planning task across multiple levels, and protects locations along the entire route. This is done by combining Inertial Flow partitioning, Private Information Retrieval (PIR), and Edge Computing techniques with our novel route planning heuristic algorithm. Normalized metrics were also formulated to quantify the privacy of the source/destination points (*endpoint location privacy*) and the route itself (*route privacy*). Evaluation on a simulated road network showed that HPRoP reliably produces routes differing only by $\leq 20\%$ in length from optimal shortest paths, with completion times within ~ 25 seconds which is reasonable for a PIR-based approach. On top of this, more than half of the produced routes achieved near-optimal endpoint location privacy (~ 1.0) and good route privacy (≥ 0.8).

CCS Concepts: • **Security and privacy** → *Privacy-preserving protocols*; *Domain-specific security and privacy architectures.*

Additional Key Words and Phrases: Route Planning Services, Location Privacy, Route Planning Algorithms

Authors' addresses: Francis Tiausas, Nara Institute of Science and Technology (NAIST), Ikoma, 630-0192, Nara, Japan, tiausas.francis_jerome.ta5@is.naist.jp; Keiichi Yasumoto, yasumoto@is.naist.jp, Nara Institute of Science and Technology (NAIST), Ikoma, 630-0192, Nara, Japan and RIKEN Center for Advanced Intelligence Project (AIP), Tokyo, 103-0027, Japan; Jose Paolo Talusan, jose.paolo.talusan@vanderbilt.edu, Vanderbilt University, 2201 West End, Nashville, Tennessee, USA; Hayato Yamana, yamana@waseda.jp, Waseda University, 1-104 Totsukamachi, Shinjuku, Tokyo, Japan; Hirozumi Yamaguchi, h-yamagu@ist.osaka-u.ac.jp, Osaka University, 1-1 Yamadaoka, Suita, Osaka, Japan; Shameek Bhattacharjee, shameek.bhattacharjee@wmich.edu, Western Michigan University, 1903 W Michigan Ave., Kalamazoo, Michigan, USA; Abhishek Dubey, abhishek.dubey@vanderbilt.edu, Vanderbilt University, 2201 West End, Nashville, Tennessee, USA; Sajal K. Das, sdas@mst.edu, Missouri University of Science and Technology, 300 W 13th St., Rolla, Missouri, USA.

2378-962X/2023/10-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Francis Tiausas, Keiichi Yasumoto, Jose Paolo Talusan, Hayato Yamana, Hirozumi Yamaguchi, Shameek Bhattacharjee, Abhishek Dubey, and Sajal K. Das. 2023. HPRoP: Hierarchical Privacy-Preserving Route Planning for Smart Cities. *ACM Trans. Cyber-Phys. Syst.* 6, 4, Article 111 (October 2023), 25 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Route Planning Services (RPS) are web-based applications which can calculate routes between two chosen points in a road network, and are indispensable navigational aids for commuters, vehicle operators, autonomous vehicles, etc. However, these same routes can reveal *points of interest* [15] which may contain users' places of work and residence, in addition to highly-sensitive information about their political, sexual, or religious tendencies [30]. Furthermore, this exposes users to risks of targeted criminal acts, mass surveillance, discrimination, etc. The rise in high-profile data breaches over the past decade has highlighted the importance of protecting user location (and route) data, and has spurred many recent works focusing on adding privacy-preserving mechanisms to RPS.

In the RPS context, privacy primarily entails keeping users' origin, destination, and route information from being acquired by untrusted entities. It exists side-by-side with other important Quality of Service (QoS) metrics such as Utility (e.g. the accuracy of the routes, etc.) and Performance (e.g. the response time of the RPS, etc.) which affect whether or not an RPS would gain widespread adoption. Finding a good balance between these metrics is crucial for any privacy-preserving RPS. For instance, privacy-preserving protocols for querying road traffic data have been developed for Vehicle Ad-hoc Networks (VANETs) but these either offload the computation cost of route planning onto the vehicle itself [24, 25, 39] or an external trusted entity [3, 4]. The latter approaches are already done by modern RPS, while the former are not web-based RPS at all. Structured encryption schemes [17, 26, 37] provide privacy-protection for both user queries and road network data while also being relatively lightweight and efficient but at the cost of inherently leaking some query-related information which are detrimental to the privacy of user routes. **Rev#3, Comm. #7**

Aside from these, there are also works that utilize Private Information Retrieval (PIR) [7] since this protocol has strong privacy guarantees. In an RPS, their efficiency depends mostly on the routing algorithm but are often considered too computationally-heavy to be used on very large databases — such as the road network graph of a large city. As such, only a few examples of PIR-based RPS have been developed over the past decade. One approach [35] compresses road network graphs in a novel PIR-queryable manner but results in longer pre-processing and query response times. The other approach [28] partitions the road network into disjoint subgraphs and these are individually retrieved via PIR to inform a local routing algorithm on the user's device. This results in longer route completion times since many PIR queries are needed to complete the route. Both approaches clearly entail a significant degradation of QoS which dissuades most mainstream RPS from experimenting with and adopting them.

Our approach aims to create a RPS that addresses the aforementioned issues by fulfilling the following three objectives: (1) produce close-to-optimal users' routes, (2) provide strong privacy guarantees for users' route data, and (3) maintain an adequate level of performance by minimizing processing and communication overhead as much as possible. The approach involves two phases. In the *pre-processing phase*, a graph partitioning technique is used to hierarchically divide the road network into balanced partitions. Routes within each partition are then pre-computed and stored in separate databases, and the same is done for an additional set of routes between neighboring partitions. In the *routing phase*, a novel hierarchical heuristic algorithm on the user's device privately obtains partial routes from the different partition databases using PIR, and iteratively **Rev#1, Comm. #3**

combines them to create a progressively finer route. We named the proposed approach, *Hierarchical Privacy-Preserving Route Planning* (HPRoP), which makes the following key contributions:

- A RPS that uses a hierarchical partitioning of the road network with a novel hierarchical route planning algorithm to produce routes with good *optimal route approximation* while maintaining strong privacy guarantees and low route completion times, **Rev#3, Comm. #6**
- A pair of privacy metrics — *endpoint location privacy* and *route privacy* — that aim to be general enough to be applicable to other privacy-preserving RPS while also accounting for specific characteristics of PIR-based RPS (such as providing strong privacy guarantees between routes in the same database), and **Rev#3, Comm. #6**
- A comprehensive evaluation of the Utility, Privacy, and Performance of the proposed RPS on a simulated road network based on Osaka City against two PIR-based baseline approaches.

Note that the baseline approaches mentioned above were also developed solely for this paper to address the lack of recent PIR-based route planning approaches which HPRoP can be compared against. However, we do not count them as separate contributions since they are primarily used as an evaluation tool. **Rev#1, Comm. #6** The rest of this article is organized as follows. Section 2 presents a summary of prior work related to privacy-preserving route planning. Section 3 discusses the mathematical models, assumptions, and other preliminaries. Section 4 presents the concept and intuition behind route planning under the constraints of PIR. Section 5 defines and discusses the privacy metrics used to evaluate the RPS. Section 6 presents the key ideas behind the approach itself, and the details of the heuristic algorithm. Section 7 discusses the evaluation framework and the results. Section 8 concludes with a brief summary of the article and some notes on potential future work.

2 RELATED WORK

Most algorithms for calculating exact shortest paths require *knowledge* of the exact source and destination locations. Classical algorithms like Dijkstra's and Bellman-Ford [10] calculate routes by repeatedly scanning connected vertices from some source point and assigning them weights until a path to the destination point is found. Modern algorithms improve upon this by leveraging unique properties of road networks. ALT [18] pre-computes distances to fixed landmarks, and uses them as lower bounds to informa a bidirectional A* search [21]. *Contraction Hierarchies* [16] pre-processes the network graph to establish "shortcut edges," facilitating faster route calculation between distant points. *Customizable Route Planning* [11] uses the network graph's topology in their metric-independent hierarchical routing method. Regardless, deploying these on the server-side inevitably means that the user's origin and destination must be divulged so that the final route can be computed. Meanwhile, deploying these on the client-side means downloading large amounts of road network data, and computing routes on more resource-constrained machines. In other words, the first case compromises privacy while the second degrades functionality.

Alternatively, algorithms for finding *approximate shortest paths* also exist, focusing on quickly obtaining short routes rather than finding the exact shortest ones. Point-to-point variants of these [8, 22, 23] were developed at a time when mobile computational power was very limited, and have been outclassed by modern exact shortest path algorithms. Yet, these remain useful in All-Pairs Shortest Path (APSP) *distance oracles* [1, 34] for speeding up goal-directed route calculations.

Recent research on privacy-preserving route planning techniques generally fall under three categories: (1) *Structured Encryption*-based, (2) *PIR*-based, and (3) *Other encryption-based* schemes. **Rev#3, Comm. #7** In addition, a number of schemes for privately querying road traffic information with applications to vehicle navigation systems also exist, but, since these either perform route planning on the vehicle itself [24, 25, 39] or an external trusted entity [3, 4], these works have been excluded here.

Structured Encryption [6] refers to techniques that allows data structures to be encrypted such that these can be queried later in a privacy-preserving manner. They are typically more efficient in terms of computation time and communication overhead at the cost of leaking a small amount of information about the data and the queries. The approaches in [26, 37] use structured encryption to find the shortest distance between vertex-pairs in encrypted graphs. These, however, are only able to find shortest distance values instead of the actual shortest paths, and are vulnerable to collusion between the storage and computing servers — which are essentially the same entity in the RPS context. In contrast, [17] is able to retrieve the actual shortest paths on a single-server setup which effectively eliminates the aforementioned issues. However, pre-computing the encrypted database for large sparse graphs (i.e. with $|V| \geq 10,000$ and $|E| \geq 30,000$) took upwards of 16.5 hours and produced very large files (around 4.4 GB), rendering it impractical for dynamic scenarios such as real-time route planning. Additionally, while all three have heavily-constrained leakage profiles, some of the information they inherently leak may be detrimental to route privacy. For instance, the number of potential query elements (i.e. the database size) can be used to determine the specific subgraph of the road network graph being used. Query repetitions and total queries for route completion can be jointly analyzed across multiple sessions to deduce the actual route. Different approaches may also leak other information in addition to the ones mentioned here. **Rev#3, Comm. #4**

PIR [7] is a technique that allows remote databases to be queried in such a way that the retrieved element would not be revealed to the service provider or any third-party entity. PIR-based schemes, therefore, have slightly stronger privacy guarantees in that repeated queries and underlying database sizes are not leaked, but have the disadvantage of much higher communication overhead. While most modern implementations [2, 19, 27, 29] have become very communication-efficient (i.e. up to $O(\sqrt{N})$), they remain impractical for accessing a database of APSP in a large city. This is because PIR schemes need to go through each individual element to avoid leaking information about the element being retrieved [5]. The approach in [35] describes a method for compressing road network graphs via sign-decomposition combined with Yao's garbled circuits [36] and PIR to protect both user queries and said graph, giving it strong end-to-end privacy guarantees. However, it also has relatively long pre-processing and query response times since the protocol must operate on compressed and encrypted data at all times. The approach in [28] partitions the network graph into disjoint sections (i.e. each consisting of a separate subgraph) which can then be retrieved via PIR during local computation of a shortest path during the routing phase. While this requires minimal pre-processing time, the total route completion time remains rather long since the locally-run routing algorithm would need multiple PIR queries to complete a single route.

Other encryption-based schemes with much stronger privacy guarantees also exist but they are also much less efficient than the previous two. For instance, [38] allows users to request routes between arbitrary source and destination partitions, as well as within said partitions in a privacy-preserving manner using 1-of-n Oblivious Transfer [31]. However, the scheme needs to compute All-Pairs of Shortest Paths (APSP) for the aforementioned partitions during the routing phase, drastically slowing down query response times. Similarly, the work in [14] uses Paillier's Encryption to privately query outgoing edge weights from vertices in the road network graph which, in turn, is used to inform a route planning algorithm running locally on the user's own device. The scheme, unfortunately, has a very high communication overhead since it has to make a separate query for every vertex that needs to be "scanned" by the routing algorithm.

As this works aims to achieve strong privacy guarantees for users' routes while also meeting utility and performance targets, PIR was chosen as the core privacy-preservation mechanism for the proposed approach. Unlike structured encryption, it does not leak information about query

repetitions, which can potentially be analyzed by an adversary to distinguish between different route requests by the same user.

3 MODELS AND ASSUMPTIONS

This section presents the assumptions and mathematical models related to the road network graph, its corresponding partition graph, and the "approximate" routes that can be derived from the latter.

Table 1. Summary of Symbols used in Sec. 3

Symbol	Description
$G = (V, E)$	Road network graph with road segments E and intersections V
$l_G(u, v)$	Length of a road segment having endpoints (u, v)
s, d	Source (s) and Destination (d) vertices
$r_G(s, d)$	Path/route between s and d
$L_G(r_G(s, d))$	Total length of path/route $r_G(s, d)$
$R_G(s, d)$	All possible paths between s and d
$\rho_G(s, d)$	*Shortest path* between s and d
$G_P = (P, C)$	Partition graph derived from G with the set of all partitions P and the connections between them C
$p = (V_p, E_p)$	A partition (i.e., a subgraph of G) consisting of road segments E_p and endpoints V_p within it
NB_{p_u}	Set of all neighboring partitions for partition p_u
ℓ	Partition level
$\mathcal{L}$	Maximum partition level ($0 \leq \ell \leq \mathcal{L}$)
P^ℓ	Subset of partitions in P at level ℓ
p_i^ℓ	A partition belonging under P^ℓ with a given index i
x_u	Representative vertex for the partition p_u
$c_{p_u p_v}$	A connection between partitions p_u and p_v through shortest path $\rho_G(x_u, x_v)$
$dist_G(u, v)$	*Shortest path* distance between u and v in G
$\mathcal{D}(p)$	Database of shortest paths for partition p
$C(\rho_0, ..., \rho_n)$	Arbitrary route combination heuristic
$r^*_G(u, v)$	Approximate shortest path between u and v
$\alpha(r^*_G(u, v))$	Optimal route approximation metric

3.1 Road Network Partitioning Model

A road network can be modelled as a directed graph $G = (V, E)$ where the edges E represent road segments, and the vertices V represent either road intersections or terminals. Each road segment $e \in E$ is a directed edge between two vertices $u, v \in V$ such that $e = (u, v)$, with parallel opposing lanes being represented by two directed edges going in opposite directions. The traversal cost for e is given by its length, $l_G(u, v)$. A *route* or *path* between two vertices $s, d \in V$ is defined as a sequence of vertices $r_G(s, d) = (v_1^{sd}, v_2^{sd}, \ldots, v_{n-1}^{sd}, v_n^{sd})$ where the first vertex $v_1^{sd} = s$ and the last vertex $v_n^{sd} = d$. The total length of $r_G(s, d)$ is given by $L_G(r_G(s, d)) = \sum_{i=1}^{|r_G(s,d)|-1} l_G(v_i^{sd}, v_{i+1}^{sd})$. Denoting *all* possible paths (between s and d) as $R_G(s, d)$, the *Shortest Path* is then:

$$\rho_G(s, d) = arg\ min_{r \in R_G(s,d)}\ L_G(r) \tag{1}$$

Most modern route planning algorithms can already deal with very large road networks, but typically do not incorporate route privacy protection mechanisms out of the box, as they tend to significantly increase processing and communication overhead as mentioned in Sec. 2. However, if the route planning task can be divided, then the additional processing cost incurred by the privacy mechanism can be distributed across multiple devices instead and ultimately improve RPS performance. This is done by first dividing the road network into different areas called *partitions*.

An arbitrary partitioning of the road network graph G is represented by a separate *partition graph* $G_P = (P, C)$ where the vertices P represent partitions and the edges C represent connections

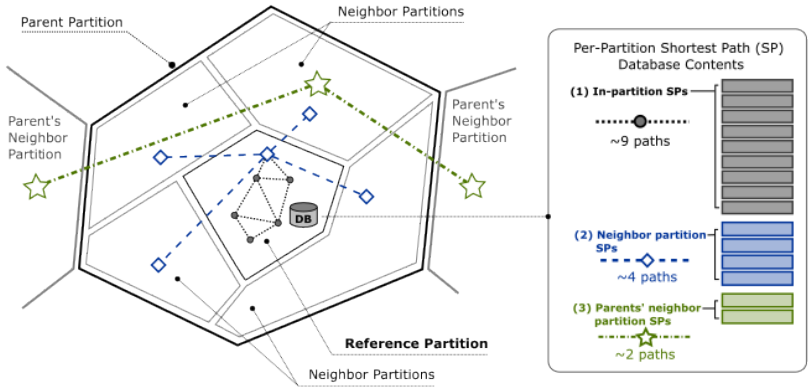

Fig. 1. Types of shortest paths stored by each partition

between them. Each *partition* $p \in P$ is a subgraph $p = (V_p, E_p)$ such that $V_p \subseteq V$ and $E_p \subseteq E$. Each connection $c_{p_u p_v} \in C$ is a shortest path $\rho_G(x_u, x_v)$ between the *representative vertices* $x_u, x_v \in V$ of any two neighboring partitions $p_u, p_v \in P$ where $x_u \in V_{p_u} \wedge x_v \in V_{p_v}$. Two partitions are considered "neighbors" if $\exists\, e = (u, v)$ s.t. $e \in E \wedge u \in V_{p_u} \wedge v \in V_{p_v}$. Finally, the set of all *neighboring* partitions for p_u is defined as $NB_{p_u} = \{p_v | p_v \in P \wedge (c_{p_u p_v} \in C \vee c_{p_v p_u} \in C)\}$

Representative vertices are ideally chosen to minimize the shortest path distance to all other vertices in the partition. Vertices with high graph centrality are good candidates, but our preliminary experiments show that it is viable to simply choose the vertex closest to the geospatial average of the coordinates of each partition's vertices with no impact on routing performance. Thus, representative vertices were chosen via this method. It then follows that $l_{G_P}(p_u, p_v) = L_G(\rho_G(x_u, x_v))$. For simplicity, the shortest path distance between any $u, v \in G$ is represented by $dist_G(u, v)$, while $dist_{G_P}(p_u, p_v)$ refers to the shortest path distance between their *containing partitions*, $p_u, p_v \in P$.

Rev#1, Comm. #2

Partitions are defined in a hierarchical manner with the maximum partition level being $\mathcal{L}$, and the subset of partitions for each level $\ell < \mathcal{L}$ denoted as $P^\ell \subseteq P$ such that $P^\ell \subset P^{\ell-1}$ for $\ell > 0$. A partition may also be defined as p_i^ℓ where ℓ is the partition's level and i is the partition's index at that level given $P^\ell = \{p_0^\ell, p_1^\ell, \ldots, p_n^\ell\}$. The base level partition set P^0 contains only one partition/subgraph $p_0^0 = G$ at $\ell = 0$. Conversely, p_0^0 might be composed of several smaller partitions p_0^1, p_1^1, p_2^1, and p_3^1 at $\ell = 1$ such that all of them are subgraphs of p_0^0, and so on. For clarity, partition levels will henceforth be referred to by their position (i.e., higher or lower) instead of their subgraph's size.

Each partition p^ℓ has three sets of shortest path data as depicted in Fig. 1: (1) the shortest paths *within* the partition p^ℓ (black edges), (2) the shortest paths to its *neighboring* partitions, NB_{p^ℓ} (blue edges), and (3) the shortest paths between the *containing* partition, $p^{\ell-1}$, to its own neighbors, $NB_{p^{\ell-1}}$ (green edges). This database of shortest paths is represented by $\mathcal{D}(p^\ell)$.

3.2 Approximate Shortest Path Model

Our approach relaxes the shortest path problem by accepting approximate shortest paths between two areas (in this case, partitions) containing s and d in place of the exact shortest path between the two points. This, in turn, reduces the number of queries required to obtain a route (hence, faster route completion times) at the cost of potentially having slightly longer paths. These are formally defined here as follows. Given an arbitrary $s, d \in V$, the exact shortest path $\rho_G(s, d)$ rarely coincides with the shortest path $\rho_{G_P}(x_s, x_d)$ where x_s and x_d are the representative vertices in the same partitions as s and d, respectively. This is because only routes between representative vertices

of partitions in P can be produced from G_P, and it is highly likely that $s \neq x_s$ or $d \neq x_d$. However, it it is still possible to "complete" the route by adding in the missing start and end sections. Letting $C(\rho_1, ..., \rho_n)$ be a route combination heuristic, the simplest "completed" route would be:

$$r^*_G(s,d) = C(\rho_G(s,x_s), \rho_{G_P}(x_s,x_d), \rho_G(x_d,d)) \tag{2}$$

This can be generalized further by replacing $\rho_{G_P}(x_s,x_d)$ with $C(\rho_{G_P}(x_1,x_2),\ldots,\rho_{G_P}(x_{n-1},x_n))$:

$$r^*_G(s,d) = C(\rho_G(s,x_s), \rho_{G_P}(x_1,x_2),\ldots,\rho_{G_P}(x_{n-1},x_n), \rho_G(x_d,d)) \tag{3}$$

where $x_1 = x_s$ and $x_n = x_d$. In this work, these kinds of combined paths are designated as *approximate shortest paths*, and their quality is measured based on how well they approximate their counterpart exact shortest paths. This metric is defined as the *optimal route approximation*:

$$\alpha(r^*_G(s,d)) = \frac{l_G(r^*_G(s,d))}{dist^*_G(s,d)} \tag{4}$$

where $l_G(r^*_G(s,d))$ is the length of the approximate shortest path, and $dist_G(s,d)$ is the length of the exact shortest path in G.

3.3 Assumptions

Having presented the mathematical models relevant to route planning, we now state the core assumptions in this work as follows. Foremost is that the RPS operates over a particular service region, and is primarily used by the general public for their day-to-day activities. The *service region* in this case is assumed to be a geographical area of arbitrary shape and size that has fixed bounds. This area is assumed to have comprehensive road network data available such that a RPS can be used to calculate routes within it. This road network is assumed to be represented as a graph that can be divided multiple times to produce subgraphs representing partitions as per Sec. 3.1. Each highest-level partition is assumed to be handled by a distinct physical or virtual device for the purpose of the RPS. Additionally, it is assumed that each partition can calculate, build, and maintain its own database of shortest paths $\mathcal{D}(p)$ independent of its other tasks. This per-partition database is then assumed to be queryable by users in a privacy-preserving manner through PIR. **Rev#1, Comm. #1**

We additionally assume that all entities other than the user are potential threats — henceforth, simply called "adversaries" — interested in gaining access to the user's route information. Note that no distinction is made between the service providers themselves and malicious third-parties. The kinds of information that can be leaked include the user's: (1) exact origin, (2) exact destination, and (3) the calculated "route" between them.

4 ROUTE PLANNING WITH PIR

Table 2. Summary of Symbols used in Sec. 4

Symbol	Description
R^*_G	Set of shortest paths between all possible pairs of vertices in V
L^{max}_G	Length of the longest path in the set, R^*_G
C_{pir}	Constant representing the impact of database sizes on record retrieval times for PIR
$\mathcal{N}_p$	Average number of vertices in a partition (across all $p \in P$)
V^{adj}_p	Set of vertices in partition p that are adjacent to other partitions
$\mathcal{N}^{adj}_p$	Average number of vertices adjacent to other partitions (i.e. V^{adj}_p) (across all $p \in P$)
$\mathcal{N}_c$	Average number of external connections (across all $p \in P$)

PIR can be used by RPS to provide strong privacy guarantees by protecting the database representation of the road network graph used to calculate routes. In the simplest case, consider **Rev#1, Comm. #6**

a database containing APSP for the whole road network graph, indexed by pairs of origin and destination vertices, (s, d). PIR can then be used to retrieve any route between any two locations on the road network using a single query (i.e. $O(1)$) with very high privacy. This is the *naive* approach which is not feasible in practice for two reasons: (1) it requires a prohibitively large amount of storage space, and (2) pre-computing APSP information for large road network graphs takes a very long time.

Assuming a graph with 10,000 vertices, a longest path length of 20 vertices, and a 1-byte representation for vertex data, the total PIR database size is already be around 2 GB. This is even larger for city-sized road networks such as Osaka City's which has $\sim$ 99,000 vertices ($\sim$ 200.8 GB keeping all other parameters same). In short, letting $R^*_G = \{r^*_G(u,v) | u, v \in V\}$ and $L^{max}_G = L_G(\arg\max_{r \in R^*_G} |r|)$, the space complexity for such an approach is $O(|V|^2 \cdot L^{max}_G)$. This issue is made even worse by PIR since individual record retrieval times become slower with larger database sizes. This has been verified through preliminary experiments where data retrieval times were observed to scale linearly with database sizes by a constant factor, C_{pir}, such that accessing a single route would have a time complexity of $O(C_{pir} \cdot |V|^2) \approx O(|V|^2)$ instead of the expected $O(1)$.

Since time complexity is heavily dependent on space-complexity for PIR-based approaches, existing works [28, 35] have focused on tackling the space complexity problem since pre-processing is assumed to be done offline only once. This is not the case for real-world road networks, however, where traffic conditions can change very quickly. One way to solve this is by distributing route data among several edge servers such that each one only manages vertices for a *distinct* partition. This reduces the time complexity to $O(|V| \cdot \mathcal{N}_p)$ where $\mathcal{N}_p = 1/|P| \cdot \sum_{p \in P} |V_p|$ is the average vertex count per partition, while the per-partition space complexity becomes $O(\mathcal{N}_p \cdot |V| \cdot L^{max}_G)$. These databases still need to be kept updated, but it is much easier to do so in a distributed manner. However, dividing the data comes at the cost of weaker privacy since the set of route data stored on the *accessed* edge server is assumed known to the adversary. This is analyzed further in Sec. 5.

Table 3. Summary of Time and Space Complexity for Sec. 4

	Space Complexity	Time Complexity
Naive	$O(\|V\|^2 \cdot L^{max}_G)$	$O(\|V\|^2)$
EPR-D	$O(\mathcal{N}_p \cdot [\mathcal{N}_p + \mathcal{N}^{adj}_p])$	$O(\mathcal{N}_p \cdot [\mathcal{N}_p + \mathcal{N}^{adj}_p] \cdot [\|V\| + \|E\| log \|V\|])$
APR-D	$O([\mathcal{N}^2_p + \mathcal{N}_c] \cdot R^{*,max}_G)$	$O([\mathcal{N}^2_p + \mathcal{N}_c] \cdot [(\|P\| + \|C\| log \|P\|) + 2)])$

4.1 Exact Partial Region Dijkstra's Algorithm (EPR-D)

The space complexity problem can be mitigated further by storing only edge weights between adjacent vertex pairs as this is the minimum information needed by Dijkstra's algorithm. This is also known as the *adjacency matrix* representation which readily maps into a database which can then be used with *any* PIR scheme. We designated this PIR-adapted approach as **E**xact **P**artial **R**egion **D**ijkstra's Algorithm (**EPR-D**), since it simply partitions and distributes graph data across several edge servers, and produces exact shortest paths. A notable disadvantage is its use of separate PIR queries to retrieve information for each vertex since the original algorithm tends to scan a lot of vertices which can make route completion times very long. It is also possible to reidentify routes based on the sequence of edge servers queried by the user. This is examined further in Sec. 5. This is reflected in its average time complexity of $O(\mathcal{N}_p \cdot [\mathcal{N}_p + \mathcal{N}^{adj}_p] \cdot [|V| + |E| log |V|])$ where $\mathcal{N}^{adj}_p = 1/|P| \cdot \sum_{p \in P} |V^{adj}_p|$ is the average number of vertices adjacent to *other* partitions,

Rev#1, Comm. #6

Rev#1, Comm. #5

in turn, represented by $V_p^{adj} = \{v | v \notin V_p \wedge [(u,v) \in E \wedge u \in V_p]\}$. Meanwhile, its average per-partition space complexity is $O(\mathcal{N}_p \cdot [\mathcal{N}_p + \mathcal{N}_p^{adj}])$. Table 3 summarizes the complexity of EPR-D.

4.2 Approximate Partial Region Dijkstra's Algorithm (APR-D)

A possible solution to the time complexity issue is by allowing the RPS to produce *approximate shortest paths* instead of exact shortest paths. This can be done as follows: (1) calculate an approximate route between source and destination partitions p_s and p_d using Dijkstra's algorithm over partition graph G_P, (2) retrieve *subpaths* to both s and d from *within* their respective partitions via database lookup, and (3) merge all obtained paths to form the final route. We designated this approach as **A**pproximate **P**artial **R**egion **D**ijkstra's Algorithm (**APR-D**), since it produces approximate routes instead of exact ones. While it is significantly faster than EPR-D, it requires more space (since full routes are stored). It also has weaker privacy guarantees since routes between distant areas tend to follow the same intermediate route as will be expanded upon in Sec. 5. Since the database has to store complete routes, the space complexity is larger than EPR-D's being at $O([\mathcal{N}_p^2 + \mathcal{N}_c] \cdot R_G^{*,max})$ where $\mathcal{N}_c = 1/|P| \cdot \sum_{p \in P} |C_p|$ is the average number of external connections from each partition. The different stages have different time complexities, with the first stage having $O([\mathcal{N}_p^2 + \mathcal{N}_c] \cdot [|P| + |C| log |P|])$ and the second stage having $O(2 \cdot \mathcal{N}_p^2 + \mathcal{N}_c)$. The combined time complexity is then $O([\mathcal{N}_p^2 + \mathcal{N}_c] \cdot [(|P| + |C| log |P|) + 2)])$ which is significantly less than that of EPR-D since $|P| << |V|$. The complexity of APR-D is also summarized in Table 3.

Rev#1, Comm. #6

Rev#1, Comm. #5

5 PROPOSED PRIVACY METRICS

To evaluate the effectiveness of a privacy-preserving PIR-based RPS, objectively quantifiable privacy metrics must first be established in the RPS context. Thus, the two models presented in this section jointly characterize the privacy of the different kinds of information that can be leaked by an RPS as described in Sec. 3.3.

Table 4. Summary of Symbols used in Sec. 5

Symbol	Description
$\Omega(s,d)$	Endpoint location privacy metric
$\mathcal{R}_G(u,v)$	Arbitrary routing mechanism operating on some graph G
$Q^{s,d}$	Query sequence used to obtain a route from s to d
Q^*	An arbitrary query sequence for no specific route
$\Phi(Q^*)$	Route privacy metric for some candidate query sequence Q^*
$P_{s,d}$	Partition sequence derived from some query sequence $Q^{s,d}$
$k(Q_{s,d}, Q^*)$	Indicator function for checking if $Q*$ can replace $Q_{s,d}$ and vice versa
V_X	A subset of V containing only the representative vertices

5.1 Endpoint Location Privacy Model

As mentioned in Sec. 3.1, the partition database is used to store the shortest paths for each partition, and is queried privately using PIR in our approach. The queried partitions are assumed to be *knowable* by adversaries, but strong privacy is still guaranteed for exact locations *within* each partition. Let $\mathcal{R}_{G_P}(s,d)$ be a *routing mechanism* which returns the approximate shortest path $\rho_{G_P}(s,d)$, and suppose that the origin location s is replaced with a nearby location s'. If s' is still in the *same* partition as s (i.e., $s' \in V_{p_s}$), then $\mathcal{R}_{G_P}(s',d) = \mathcal{R}_{G_P}(s,d)$. The same applies replacing d with any $d' \in V_{p_d}$. An adversary knowing only $\mathcal{R}_{G_P}(s,d)$ would be unable to distinguish s, d from all other possible s', d' as long as $s' \in V_{p_s}$ and $d' \in V_{p_d}$. Thus, location privacy is guaranteed for s and d within p_s and p_d respectively.

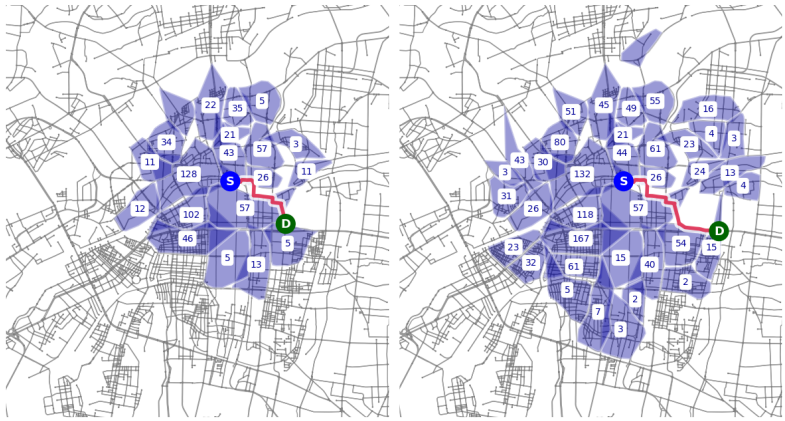

Fig. 2. Queried Partitions using Dijkstra's algorithm to calculate two routes with the same origin but different destinations (~ 1 km away from each other). The numbers indicate how many queries were handled by each partition.

The privacy for any s, d pair is then proportional to the number of possible s', d' pairs that can be drawn between V_{p_s} and V_{p_d}. This is designated as the *endpoint location privacy* metric:

$$\Omega(s,d) = 1 - \frac{1}{|V_{p_s}| \cdot |V_{p_d}|} \tag{5}$$

where V_{p_s} and V_{p_d} give the sets of all vertices in p_s and p_d, respectively. Note that while having more vertices per partition (i.e. larger $|V_{p_s}|$ and $|V_{p_d}|$) is advantageous for endpoint location privacy, this also means higher computation overhead for PIR. Extensive preliminary experiments with the PIR scheme used by our approach showed that the retrieval times scaled almost linearly with the database size at a rate of roughly $\frac{1}{45000} \approx 2.22 \times 10^{-5}$ seconds per record. That is, a database with $|V_p|^2 \approx 10000$ routes was found to have an average retrieval time of ~ 0.25 seconds, while another with $|V_p|^2 \approx 100000$ routes took ~ 2 seconds. **Rev#1, Comm. #4**

5.2 Route Privacy Model

Endpoint location privacy assumes that a route's origin and destination partition are already known, and thus quantifies only the privacy of the *exact* origin and destination points. This section focuses on the privacy of the *routes themselves*. As with endpoint location privacy, it is assumed that the queried partitions are knowable by adversaries. It is also assumed that they have in-depth knowledge about the algorithms used by the RPS.

Let $Q_{s,d} = \{q_0, \ldots, q_n\}$ be the *query sequence* that a user must perform to obtain a route from s to d. This sequence can be transformed into a *partition sequence* $P_{s,d} = \{p_0, \ldots, p_n\}$ (where $p_0 = p_s$ and $p_n = p_d$) using a function $f_{qp} : Q \rightarrow P$ that maps every element of Q to its handling partition in P. If the partitioning is hierarchical, then only the highest-level partitions are considered since they already contain the shortest path data of lower-level partitions as stated in Sec. 3.1.

Note that partition sequences are not simply partitions along the final route. For instance, in the case of Dijkstra's algorithm, they can be thought of as the *entire* sequence of "scanned" vertices as depicted in Fig. 2. It is therefore possible for several routes to share the same partition sequence (though unlikely in the case of Dijkstra's). This is modeled as an indicator function that identifies

whether or not a candidate Q^* can replace $Q_{s,d}$ for calculating $r_{G_P}(s,d)$ and vice versa:

$$k(Q_{s,d}, Q^*) = \begin{cases} 1 & \text{if } f_{qp}(Q_{s,d}) = f_{qp}(Q^*) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

With this, the lower bound for the total number of *distinct routes* that share Q^* is:

$$\sum_{s',d' \in V_X} k(Q_{s',d'}, Q^*) \tag{7}$$

where $V_X \subseteq V$ contains *only* the representative vertices (e.g., x_u, x_v) associated with the partition graph G_P as described in Sec. 3.1. This ensures that routes where $p_{s'} \neq p_s$ and $p_{d'} \neq p_d$ are counted with equal importance as the route where $p_{s'} = p_s \wedge p_{d'} = p_d$ — hence, the focus on *distinct* routes. Finally, the *route privacy* can then be quantified for any Q^* as follows:

$$\Phi(Q^*) = 1 - \frac{1}{\sum_{s',d' \in V_X} k(Q_{s',d'}, Q^*)} \tag{8}$$

6 HIERARCHICAL PRIVACY-PRESERVING ROUTE PLANNING

Our proposed approach, Hierarchical Privacy-Preserving Route Planning (HPRoP), is built up from several key design choices to meet particular requirements. That is, HPRoP should be able to: **Rev#1, Comm. #1**

- REQ1: Compose feasible approximate shortest paths by carefully choosing its component routes from the appropriate partitions,
- REQ2: Privately retrieve component route information from said partitions with minimal processing overhead,
- REQ3: Produce an approximate shortest path with good *optimal route approximation* values (i.e., $\alpha(r^*_G(s,d))$ close to 1.0),
- REQ4: Ensure a good level of privacy protection for the user's exact origin and destination points, and intermediate route,
- REQ5: Reflect dynamic and up-to-date road conditions, and
- REQ6: Scale reasonably well with changes in client demand and computational resource availability over time and per area.

All these are brought together by a novel hierarchical route planning heuristic presented in the latter half of this section, along with other improvements to privacy and routing.

6.1 Private Information Retrieval (PIR)

HPRoP uses PIR as its core route privacy-preservation mechanism. The choice of implementation was the SealPIR [2] library configured to use Brakerski/Fan-Vercauteren (BFV) Homomorphic Encryption (HE)[13] with a database upper bound of $N = 216$, a plaintext modulus of $log(t) = 12$, and a dimensionality factor, $d = 2$. This implementation was chosen specifically for its significantly reduced processing and communication overhead compared to other HE-based ones, making it ideal for an RPS. This along with the hierarchical route planning heuristic drastically reduces the number of queries and, in effect, the route completion time.

6.2 Inertial Flow Partitioning

Optimal route approximation and endpoint location privacy are highly-dependent on how the service region is partitioned. Straightforward methods such as grid partitioning are simple but often result in disjoint partition subgraphs in the presence of natural barriers like rivers, etc. To mitigate this, one way would be to ensure that each partition subgraph is a strongly-connected

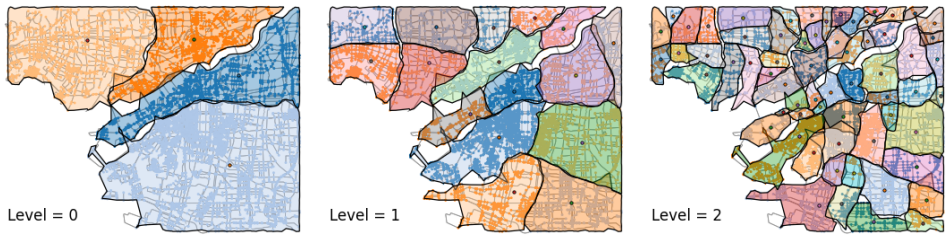

Fig. 3. Road network graph of Osaka City, Japan hierarchically-partitioned using the Inertial Flow algorithm

Fig. 4. Cloud-based Architecture (Left) vs Distributed/Edge-based Architecture (Right)

component, ensuring high internal connectivity and reachability. Examples of road network graph aware partitioning methods include PUNCH [12], *Buffoon* [32], and Inertial Flow [33].

Inertial Flow was chosen for HPRoP as it is a relatively simple algorithm based on maximum flow which results in balanced partitions and also preserves internal connectivity. A vertex threshold of around 300 nodes per partition was chosen instead of an area-based threshold to guarantee an *Endpoint Location Privacy* of $\Omega(s, d) \approx 99.998\%$ (i.e., $< 0.002\%$ probability). Moreover, based on preliminary experiments with SealPIR, a partition database with $300^2 \approx 90000$ routes is expected to have retrieval times between $1.5 - 2.5$ seconds (1.75 seconds on average) which is viable when combined with HPRoP's reduced query counts. A queue is initialized by adding the entire road network graph to it. A graph from this queue is then used as input to the Inertial Flow algorithm to produce two balanced partition subgraphs. The simple iterative technique in [1] was used alongside Inertial Flow to find and apply optimal cuts during this step. If any of the subgraphs do not yet satisfy vertex threshold, they are simply added back to the queue. This entire procedure is then repeated until the queue is empty. Afterwards, every two consecutive cuts was then retroactively denoted as a separate *partition level* as shown in Fig. 3, and the partitions under each are then tagged accordingly. Due to the vertex threshold, the highest level partition may vary greatly from area to area. Some routes therefore require more queries to complete over other routes, which increases profiling risk. A partition level threshold $\ell_{threshold} = 3$ was therefore imposed such that higher level ones were reassigned to $\ell = \ell_{threshold}$.

Rev#1, Comm. #4

6.3 Distributed Architecture

HPRoP leverages the hierarchically partitioned road network by delegating each partition to a different entity. In the cloud-based scenario, these entities would be server instances; while, in the edge-based scenario, these would be edge-servers throughout the smart city. The edge-based architecture presents several advantages. First, it allows PIR queries to be directed only to partitions which have the information necessary to answer them, effectively distributing the computational load of using PIR. Second, it allows the system to better scale based on the number of users,

availability of computing resources, etc. which can vary greatly at different times across different parts of the city as shown in Fig. 4. Finally, it also makes the pre-computation of shortest paths to neighboring partitions more efficient since it can be done independently by every partition after an initial exchange of road condition information with said neighbors. This is useful for reflecting dynamic road conditions bound to some local area.

6.4 Heuristic Algorithm

HPRoP uses a simple heuristic algorithm to arrive at an approximate shortest path as follows:

(1) *Initialization*: Find an initial basis route between the lowest level partitions containing source and destination, then move up one level.
(2) *Subroute Connection*: Connect the source and destination partitions at the current level to the basis route using subroutes.
(3) *Basis Route Merging*: Merge the subroutes into the basis route.
(4) Repeat steps (2) to (3) until the highest partition level is reached

Algorithm 1: Hierarchical Route Planning Heuristic

Input: Source node s, Destination node d, Current level l_c
Output: The final route r_*

1 **begin**
2 $l_o \leftarrow FindBaseLevel(s, d)$;
3 $r_* \leftarrow RetrievePath(p_s^{l_o}, p_d^{l_o}, \text{“from source”})$;
4 Initialize $l_c \leftarrow l_o + 1$;
5 **while** $l_c \leq l_{threshold}$ **do**
6 Initialize $r_s^{l_c}, r_d^{l_c} \leftarrow [\,]$;
7 **if** $l_c < (l_{threshold} - 1)$ **then**
8 $r_s^{l_c} \leftarrow GetSubroute(s, l_c, r_*, \text{“from source”})$;
9 $r_d^{l_c} \leftarrow GetSubroute(d, l_c, r_*, \text{“from destination”})$;
10 **else**
11 $r_s^{l_c} \leftarrow GetEndSubroute(s, l_c, r_*, \text{“from source”})$;
12 $r_d^{l_c} \leftarrow GetEndSubroute(d, l_c, r_*, \text{“from destination”})$;
13 $r_* \leftarrow MergeRoutes(r_*, r_s^{l_c}, r_d^{l_c})$;
14 $l_c \leftarrow l_c + 1$;
15 **return** r_*;

Algo. 1 runs exclusively on the client-side, sending PIR queries to edge servers handling specific partitions. Route information is obtained solely through these PIR queries, and, thus, no information is leaked by the queries themselves. However, the number of queries, their timestamps, and the partitions they were sent to are still assumed to be known to the adversary.

The algorithm starts with an *Initialization* step (lines 2-4 in Algo. 1) which finds an approximate shortest path between the lowest level partitions containing s and d separately as shown in Fig. 5. This level is denoted as the base level $l_o = \arg\min_l (p_s^l \neq p_d^l)$. For simplicity, this is just denoted as $FindBaseLevel(s, d)$ in Algo. 1. The client then sends a PIR query to partition $p_s^{l_o}$, retrieving a route to partition $p_d^{l_o}$. This corresponds to $RetrievePath(p_u, p_v, \mathfrak{D})$ which retrieves the shortest path between two partitions taking into account some direction flag $\mathfrak{D}$. This flag simply indicates whether the path is being calculated from the source or the destination, which will be relevant later.

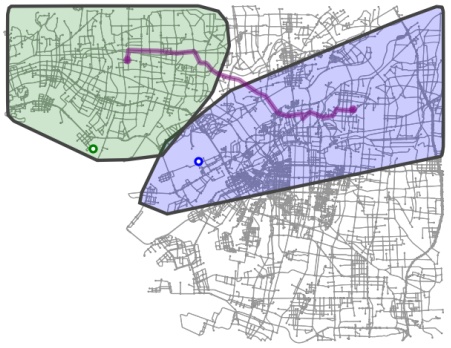

Fig. 5. Initialization: Find a shortest path between the lowest-level partitions containing s (blue circle) and d (green) separately.

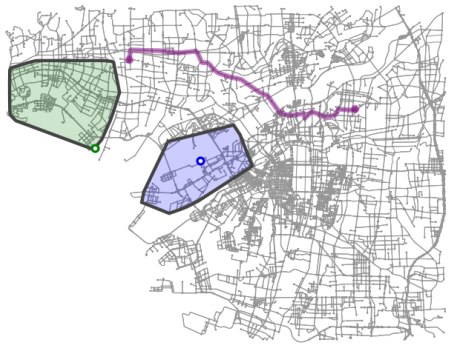

Fig. 6. Initialization: Use discovered path as basis route (purple line) for the next level

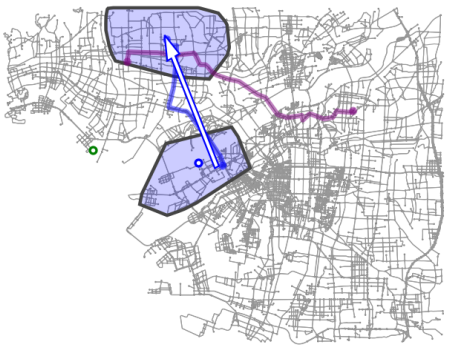

Fig. 7. Source Subroute Connection: Find and connect source subroute (blue line) to basis route (purple line)

Fig. 8. Dest. Subroute Connection: Find and connect dest. subroute (green line) to basis route (purple line)

Fig. 9. Basis Route Merging: Merge routes and use as new basis route (purple line)

The retrieved route is then denoted as the initial *basis route* r_* shown in Fig. 6. At the end of this step, the current level variable l_c is also initialized (line 4).

Algorithm 2: Subroute Connection

Input: Basis node x, Current level l_c, Basis route r_*, Direction $\mathfrak{D}$
Output: Subroute at the current level $r_x^{l_c}$

```
1  begin
2      Initialize r_x^{l_c} ← [];
3      p_c ← p_x^{l_c};          // This retrieves either source or destination sub-partition depending on direction 𝔇
4      RP^{l_c} ← FindRoutePartitions(r_*, l_c)
5      if 𝔇 is "from destination" then
6          Reverse(RP^{l_c});
7      i ← 0;
8      while DoesNotIntersect(r_x^{l_c}, r_*) and i < |RP^{l_c}| do
9          r_part ← RetrievePath(p_c, RP^{l_c}[i], 𝔇);
10         if 𝔇 is "from source" then
11             r_x^{l_c} ← r_x^{l_c} + r_part;
12         else if 𝔇 is "from destination" then
13             r_x^{l_c} ← r_part + r_x^{l_c};
14         p_c ← RP^{l_c}[i];
15         i ← i + 1;
16     return r_x^{l_c};
```

The main loop starts from the *Subroute Connection* steps (lines 6-9 in Algo. 1). The algorithm for Subroute Connection itself is described in Algo. 2. A *subroute* $r_x^{l_c}$ is defined as a path that connects a *basis partition* $p_x^{l_c}$ to the current basis route r_*. The basis partition given by $p_x^{l_c}$ is always the source or destination partition at level l_c containing some vertex x, and is used to obtain the source or destination sub-partitions (depending on the direction $\mathfrak{D}$) at line 3 of Algo. 2. For instance, the *source subroute* is obtained by finding a sequence of shortest paths from $p_s^{l_c}$ that connects to the basis route as shown in Fig. 7. This step also uses several important functions, such as: (1) $FindRoutePartitions(r_*, l)$ which gives the sequence of partitions at level l along r_*, and (2) $DoesNotIntersect(r_x^{l_c}, r_*)$ which is "True" if $r_x^{l_c}$ and r_* have no common vertices. Starting from $p_x^{l_c}$, it builds $r_x^{l_c}$ by retrieving a path to nearby connected partitions (line 9) and then connecting them to the subroute (lines 10-13). Since $r_x^{l_c}$ might not yet intersect r_* during the initial iteration, this is repeated with an updated reference partition (lines 14-15) until an intersection is found. Computing the *destination subroute* follows the same steps but first has to *reverse* the aforementioned sequence (lines 5-7) so that the algorithm can begin from the last route partition. This step is shown in Fig. 8. **Rev#2, Comm. #3**

The *Basis Route Merging* step (line 13 in Algo. 1) is performed once the source and destination subroutes are found. The basic idea is to find the "best" point at which the subroutes intersect with the basis route and join them there. For the source subroute, the "best" point is as far as possible from the start of the basis route; while for the destination subroute, this is as far as possible from the end of the basis route. This is illustrated in Fig. 9. This merged route is then used as the new basis route. The current level l_c is then updated (line 15), and the loop is restarted. The loop is terminated once l_c reaches $l_{threshold}$.

The time complexity of this algorithm depends on the required number of PIR database lookups. Finding the initial basis route and calculating the final routes at p_s and p_d always require a single lookup each. Meanwhile, the number of lookups at each level depends on the maximum length **Rev#1, Comm. #5**

of the shortest path in G_P *at that level* which is given by $max_{p_u^\ell, p_v^\ell \in P^\ell} |r^*_{G_P^\ell}(p_u^\ell, p_v^\ell)|$ where $G_P^\ell \subset G_P$ containing only that level's partitions P^ℓ and connections C^ℓ. The average time complexity is then:

$$O\left([\mathcal{N}_p^2 + \sum_{\ell \in L} \mathcal{N}_c^\ell] \cdot \left[3 + \sum_{\ell \in L} 2 \cdot \max_{p_u^\ell, p_v^\ell \in P^\ell} |r^*_{G_P^\ell}(p_u^\ell, p_v^\ell)|\right]\right) \tag{9}$$

where $\mathcal{N}_c^\ell$ is the average number of connections from each partition *at level* ℓ. Since HPRoP precomputes and stores shortest path data for multiple levels per partition, it is necessary to account for $\mathcal{N}_c^\ell$ in HPRoP's space complexity:

$$O([\mathcal{N}_p^2 + \sum_{\ell \in L} \mathcal{N}_c^\ell] \cdot R_G^{*,max}) \tag{10}$$

6.4.1 Route Privacy Mechanism. Route privacy as defined in Sec. 5.2 quantifies the privacy based on how many other possible routes have a query sequence matching that of a given route, where more matches mean better privacy. That is, an adequate route privacy mechanism should: (1) maximize the matching of query sequences between all possible routes, and (2) minimize the information gain from the order of queries in the sequence itself. HPRoP already achieves the latter via hierarchical execution which can somewhat obfuscate the actual query sequence, but does not necessarily strengthen the former. To address this, the algorithm is extended to *pad* the query sequence with *dummy queries*. The basic idea is to query multiple other partitions instead of just p_s and p_d to ensure that the actual origin and destination partitions are "hidden' among them. In theory, querying more partitions would mean better route privacy at the cost of longer route completion times. Achieving full route privacy, however, would require querying all level partitions at every iteration of the algorithm at least once, which would require prohibitively long route completion times. For example, a service region with a total of 463 partitions would require roughly 810 seconds (13.5 minutes) on average to complete a single route. However, simply querying a small random subset of the aforementioned partitions will not be enough to ensure a certain level of route privacy, since an adversary can simply use the hierarchy of partitions to check for inconsistencies in the set of queried partitions and easily identify the dummy ones. Instead, we chose to limit HPRoP to querying all other partitions *under the same parent* as the highest level partitions containing s and d. This selection method is straightforward and ensures that none of the queried partitions can easily be identified as dummy partitions. Additionally, this ensures that route privacy will be around $\Phi(Q^*) \approx 1 - 1/jk$ where j and k are the total number of partitions under the same parent partitions as p_s and p_d, respectively.

Rev#1, Comm. #7

This is implemented through the *Subroute End Connection* steps (lines 10-12 in Algo. 1), while the procedure itself is presented in Algo. 3. Instead of stopping when the subroute and basis route first intersect, this algorithm continues until all other partitions sharing the same parent $p_x^{l_c - 1}$ as the basis partition have been queried. This is done by repeatedly drawing a partition p_{sub} from a queue of these same-parent sub-partitions SP^{l_c} (line 10). If p_{sub} is the same as the current partition p_c, then a part of the subroute is retrieved as normal (lines 11-18). If it is a route partition, it is instead pushed back to the subpartition queue (lines 19-22). If both prior conditions are not satisfied, then a dummy query is simply sent to p_{sub} (line 24). This ensures that important queries are mixed in with dummy queries, making it more difficult to determine which ones are relevant.

6.5 Shortcut Connections

A simple strategy for improving algorithm performance is through pre-computing and storing shortest path data to partitions *beyond* just the adjacent ones. This reduces the number of queries

Algorithm 3: Subroute End Connection with Dummies

Input: Source or Destination node x, Current level l_c, Basis route r_*
Output: Source or Destination subroute at the current level $r_x^{l_c}$

begin
  Initialize $r_x^{l_c} \leftarrow [\,]$;
  $p_c \leftarrow p_x^{l_c}$;
  $RP^{l_c} \leftarrow FindRoutePartitions(r_*, l_c)$
  if $\mathfrak{D}$ *is "from destination"* **then**
    $Reverse(RP^{l_c})$;
  $SP^{l_c} \leftarrow FindSubPartitions(p_x^{l_c-1})$ **as** *Queue*
  $j \leftarrow 0$;
  while $|SP_x^{l_c-1}| > 0$ **do**
    $p_{sub} \leftarrow Pop(SP^{l_c})$;
    if $p_{sub} = p_c$ **and** $DoesNotIntersect(r_d^{l_c}, r_*)$ **then**
      $r_{part} \leftarrow RetrievePath(p_{sub}, RP^{l_c}[j])$;
      if $\mathfrak{D}$ *is "from source"* **then**
        $r_x^{l_c} \leftarrow r_x^{l_c} + r_{part}$;
      else if $\mathfrak{D}$ *is "from destination"* **then**
        $r_x^{l_c} \leftarrow r_{part} + r_x^{l_c}$;
      $p_c \leftarrow RP^{l_c}[j]$;
      $j \leftarrow j + 1$;
    else if $p_{sub} \in RP^{l_c}$ **and** $DoesNotIntersect(r_d^{l_c}, r_*)$ **then**
      if $Index(p_{sub}, RP^{l_c}) > j$ **then**
        $Push(p_{sub}, SP^{l_c})$
        $Shuffle(SP^{l_c})$
    else
      $SendDummyQuery(p_{sub})$;
  return $r_x^{l_c}$;

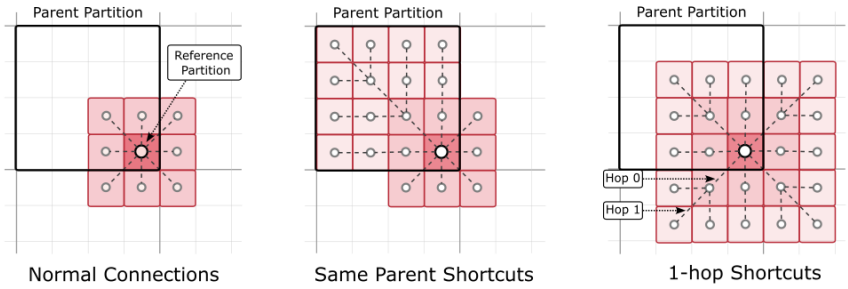

Fig. 10. Demonstration of different shortcut connection strategies for improving the heuristic algorithm's performance against the base case (Left), where the dark red shape represents the starting partition, and the lighter red shapes represent the partitions it connects to. The black outline represents the starting partition's parent. (Middle) uses Same Parent Shortcuts, while (Right) uses 1-hop Neighbor Shortcuts.

to complete a route while also improving optimal route approximation for paths to further away partitions. These paths to non-adjacent partitions were therefore denoted as *Shortcut Connections*, represented as additional edges in the partition graph.

These shortcut connections, however, also increase the pre-computation time for each partition relative to how many of them need to be made. In addition, the extent of the road network graph needed for pre-computation also increases based on the distance to the partitions being connected. It is therefore more useful to limit the number of partitions to connect to and how far those partitions can be. HPRoP considers two methods for determining shortcut connections: (1) the Same Parent Shortcuts method, and (2) the *N-hop Neighbor Shortcuts* method. *Same Parent Shortcuts* simply connects each partition to all other partitions under the same parent partition as shown in Fig. 10 (Middle). *N-hop Neighbor Shortcuts* pre-computes shortcuts to other N-hop away partitions as shown in Fig. 10 (Right). Both would theoretically improve the *optimal route approximation* when calculating subroutes between non-adjacent partitions and greatly reduce the possibility of broken routes. HPRoP currently has no mechanisms to handle these other than deferring to the user's device to perform local route calculation to bridge the final gap. The usefulness of these methods are shown through the results in Sec. 7.3.1. **Rev#2, Comm. #4**

7 EVALUATION

In this section, the details of the evaluation framework for HPRoP are first presented prior to showing the actual evaluation results and their subsequent analysis.

7.1 Environment

HPRoP was implemented in Python on a Jupyter notebook for ease of testing and visualization, with the notebook itself encapsulated in a Docker container for portability. The execution environment was a dedicated Linux server running Ubuntu 20.04.1 SMP equipped with a AMD Ryzen Threadripper 3970X 32-Core processor and 256 GB RAM in total.

The service region was a rectangular geographical area of roughly 546 km^2 (i.e. 26 km in width, 21 km in height) encompassing the entire road network of Osaka City, Japan and a portion of the immediately outlying areas. Its road network graph consists of $|V| = 99,734$ vertices and $|E| = 269,614$ edges. The region is hierarchically partitioned using the Inertial Flow algorithm as shown in Fig. 3 based on the parameters in Sec. 6.2.

7.2 Methodology

Evaluation was done through several metrics under the following categories: (1) Utility, (2) Privacy, and (3) Performance. The Utility category pertains to the usefulness of the service, with the *Optimal Route Approximation* metric falling under this category. Since the base algorithm in Sec. 6.4 cannot guarantee complete routes, *Route Errors* are also included here as a metric. Route errors are then defined as the occurrence count of broken routes during testing. In turn, a *broken route* is defined as a route where a subroute connection *cannot be established* to the highest level partition containing either s or d, and thereby results in a route that cannot be completed by HPRoP's algorithm. The Privacy category is comprised of the *Endpoint Location Privacy*, and *Route Privacy* metrics described in Sec. 5. The Performance category pertains to how well the service can deal with higher client demand, dynamic road conditions, etc. without service quality degradation. The *Memory Usage*, *Route Completion Time*, and *Pre-processing Time* metrics fall under this category. **Rev#2, Comm. #4**

Evaluation was done by comparing HPRoP to the two baseline PIR-based approaches — EPR-D and APR-D — previously presented in Sec. 4. For the *Utility* category, *Privacy* category, and *Route Completion Time* metrics, all three approaches were evaluated by calculating routes for randomly-generated s, d pairs until 4,000 successful routes have been completed. This termination threshold

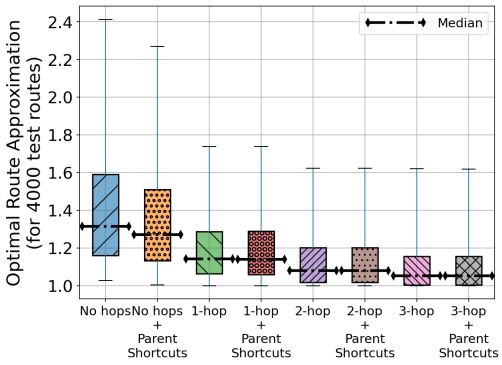

Fig. 11. Distribution of *Optimal Route Approximation* results for different Shortcut Connection methods

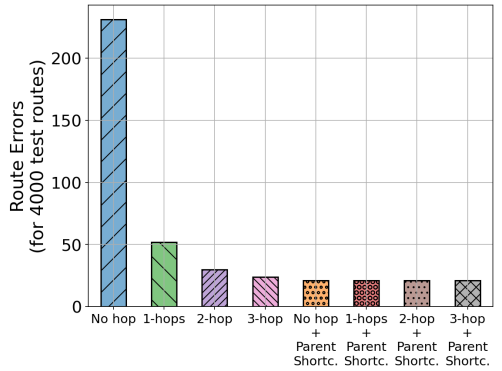

Fig. 12. Comparison of total Route Errors for different Shortcut Connection methods

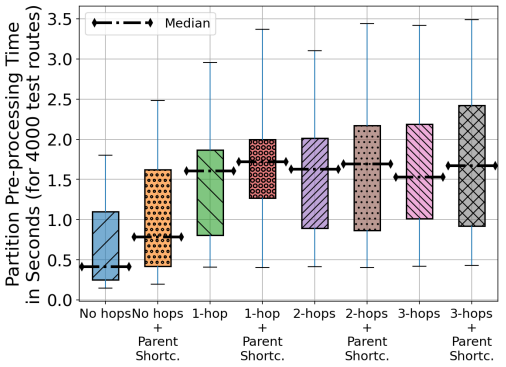

Fig. 13. Distribution of Pre-Computation Times for different Shortcut Connection methods

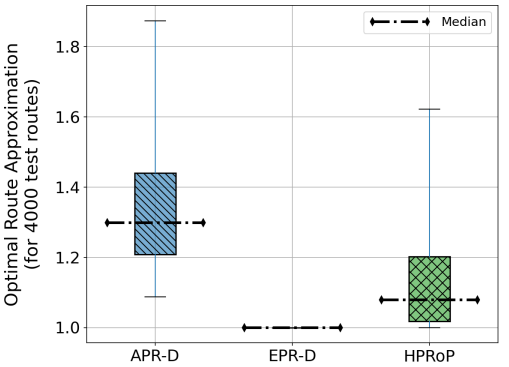

Fig. 14. Distribution of Optimal Route Approximation results using APR-D, EPR-D, and HPRoP

was chosen to be sufficiently high enough to capture any route errors that might occur for HPRoP. Note that both APR-D and EPR-D are guaranteed to produce complete routes so this metric is no longer evaluated for them. *Memory Usage* was evaluated by calculating the projected size of the per-partition databases for the highest-level partitions. Finally, *Pre-processing Time* was evaluated by measuring the time to build each partition's database.

7.3 Results

7.3.1 Effect of Shortcut Connections. The base performance of HPRoP under different shortcut connection methods in Sec. 6.5 is first characterized with the goal of finding the method that maximizes optimal route approximation while minimizing both route completion time and route errors. Fig. 11 shows the resulting distribution of optimal route approximation values for 4,000 successful test routes under different methods. Surprisingly, *Same Parent Shortcuts* have an almost negligible effect on optimal route approximation, with *N-hop Neighbor Shortcuts* being a more effective way to increase the said metric. However, *Same Parent Shortcuts* were highly effective in preventing the occurrence of broken routes, reducing the error count to 21. Further investigation of these remaining errors showed that they were caused by choosing the same vertex as s and d which results in failure as no routing can be done by the algorithm. In short, for all 4,000 test routes, *Same Parent Shortcuts* seem to eliminate all occurrences of true broken routes — i.e., where the highest level partitions containing s or d could not be reached. This also validates the hypothesis

Rev#2, Comm. #4

that broken routes are caused by a lack of reachability at the final partition level that can contain more than 4 child partitions due to the deliberate choice to set $\ell_{threshold} = 3$ mentioned in Sec. 6.2.

Meanwhile, using *1-hop Neighbor Shortcuts* drastically improves the results of the 75-th percentile from $\alpha(r_*) \approx 1.54$ to $\alpha(r_*) \approx 1.28$, but anything beyond *2-hop Neighbor Shortcuts* is seen to have gradually diminishing returns. Finally, the overhead caused by the additional shortcut connections is evaluated based on the pre-processing time metric in Sec. 7.2. Fig. 13 shows that both methods increase the per partition pre-processing time to about ~ 1.5 seconds once *1-Hop Neighbor Shortcuts* are introduced but stabilizes around this value even as the number of hops are further increased. In contrast, the effect of *Same Parent Shortcuts* on pre-processing time is minimal, amounting to an increase of ~ 0.1 seconds on average. Thus, both methods are equally viable in terms of this metric.

Table 5. Summary of Results for different Shortcut Connection Configurations

	Normal				**With Same Parent Shortcuts**			
	0-hop	1-hop	2-hop	3-hop	0-hop	1-hop	**2-hop**	3-hop
Optimal Route Approximation								
Min	1.0	1.0	1.0	1.0	1.0	1.0	**1.0**	1.0
25%	1.160	1.062	1.018	1.003	1.133	1.060	**1.017**	1.003
50%	1.314	1.141	1.080	1.052	1.270	1.138	**1.079**	1.052
75%	1.590	1.287	1.202	1.156	1.510	1.288	**1.202**	1.156
Max	16.256	5.210	5.371	5.371	15.424	5.210	**5.371**	5.371
Per-Partition Routes Calculation Time (in seconds)								
Min	0.04	0.11	0.19	0.19	0.08	0.11	**0.19**	0.19
25%	0.16	0.58	1.88	3.33	0.24	0.75	**1.92**	3.48
50%	0.20	1.43	2.49	4.41	0.47	1.51	**2.90**	4.75
75%	0.28	1.60	3.36	5.70	1.30	1.74	**3.47**	6.03
Max	1.27	3.17	5.78	10.55	3.15	3.37	**6.75**	11.44
Route Errors (per 4,000 successful routes)								
Total	231.0	52.0	30.0	24.0	21.0	21.0	**21.0**	21.0

Table 5 summarizes the results discussed so far. Based on these, it was decided to use *2-hop Neighbor Shortcuts* with *Same Parent Shortcuts* as the representative configuration for HPRoP as it offers good *optimal route approximation* and short pre-computation times with minimal errors.

7.3.2 Optimal Route Approximation. The performance of HPRoP, APR-D, and EPR-D in terms of *optimal route approximation* is shown in Fig. 14. As expected, EPR-D achieves a constant *optimal route approximation* value of $\alpha(r_*) = 1.0$ since it always produces exact shortest paths. APR-D produced routes with $\alpha(r_*) \approx 1.44$ for the 75-th percentile despite operating only over the partition graph. HPRoP produced even better routes with $\alpha(r_*) \approx 1.20$ for the 75-th percentile, whereas APR-D was only able to achieve this for the 25-th percentile of all results. Additionally, HPRoP achieves much better worst-case routes than APR-D.

7.3.3 Endpoint Location Privacy. Endpoint Location Privacy $\Omega(s, d)$ describes how well the exact s, d is kept private as described in Sec. 5.1. Since Inertial Flow partitioning was used, evaluation results showed that all three approaches were able to achieve an average endpoint location privacy of $\Omega(s, d) = 0.999982$. Thus, each route is *indistinguishable* from approximately 99% of all other routes between the same partitions, making all three approaches equally viable.

7.3.4 Route Privacy. Route Privacy $\Phi(Q^*)$ describes how well a route is kept private based on how many other routes share a query sequence similar to its own as described in Sec. 5.2. To evaluate

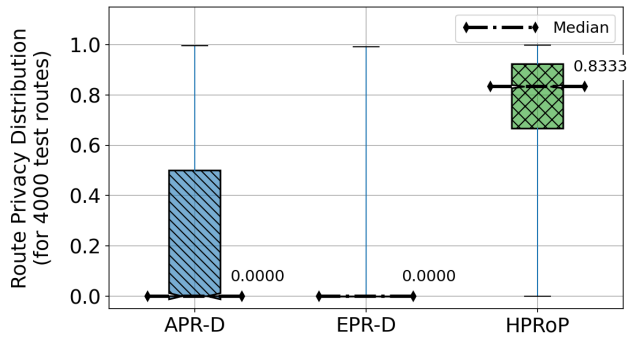

Fig. 15. Distribution of Route Privacy $\Phi(Q^*)$ results for APR-D, EPR-D, and HPRoP

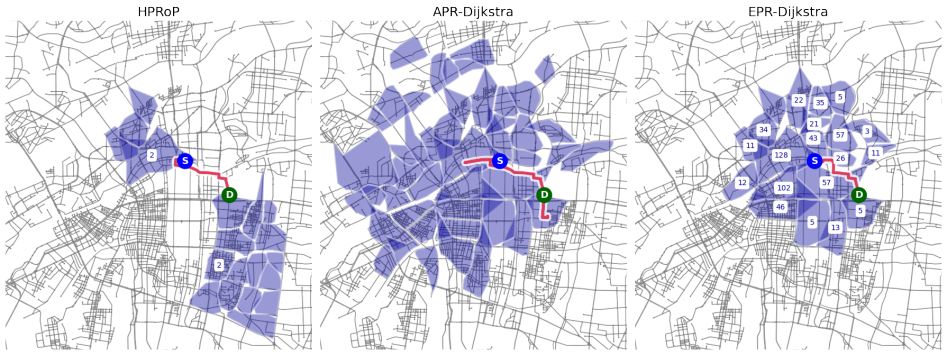

Fig. 16. Visualization of Queried Partitions for APR-Dijkstra (Center), EPR-Dijkstra (Right) and HPRoP (Left) for the same route. Numbers indicate partitions that were queried multiple times.

this, a lookup table for the routes between each $s, d \in V_x$ such that $s \neq d$ was done separately for EPR-D, APR-D, and HPRoP to account for their unique querying behaviors as shown in Fig. 16.

Fig. 15 shows a comparison of the route privacy distributions across all three approaches. EPR-D showed the worst route privacy, achieving $\Phi(Q^*) > 0$ only for the 25-th percentile of all results. APR-D performed only slightly better with $\Phi(Q^*) > 0$ for about 50% of all results, while achieving $\Phi(Q^*) \geq 0.5$ for only 25% of them. This is expected since both have no inherent privacy mechanisms, yet this does illustrate that APR-D is still better than EPR-D in terms of route privacy. HPRoP, in contrast, has route privacy of $\Phi(Q^*) \geq 0.80$ for the 50-th percentile of all results, while also achieving $\Phi(Q^*) \geq 0.50$ for the 75-th percentile, surpassing both EPR-D and APR-D. However, further analysis of the routes showed that the worst-case route privacy (i.e. $\Phi(Q^*) = 0.0$) still happens for $\sim 12.2\%$ of all test routes. This suggests that HPRoP does not fully guarantee route privacy for all cases although this can be increased further by adding more dummy queries.

Additionally, the relationship between optimal route approximation $\alpha(r_*)$ and route privacy metric $\Phi(Q^*)$ was also analyzed for HPRoP. This was done to determine whether some trade-off exists between the two metrics. The results in Fig. 17 show that majority of the routes have $\alpha(r_{s,d}) < 1.2$ across widely-varying levels of route privacy. This suggests that the two metrics have little effect on one another, and performing a simple Pearson correlation confirms that there is only a weak positive correlation ($\rho \approx 0.104$) between the two.

7.3.5 Route Completion Time. Obtaining a route using any of the three algorithms (APR-D, EPR-D, and HPRoP) requires the client to make multiple PIR queries to the RPS. Thus, route completion time is highly dependent on how many such queries need to be made, and is equivalent to the

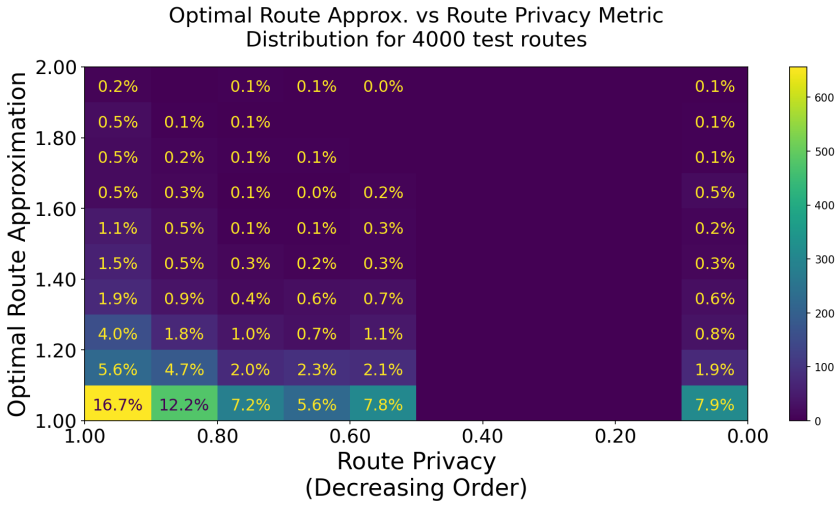

Fig. 17. Distribution of Optimal Route Approximation and Route Privacy results using HPRoP for 4,000 test routes. Note that both axes were reoriented to show the best results on the lower left.

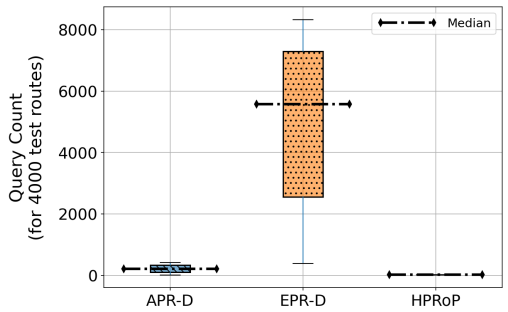

Fig. 18. Distribution of the required number of queries for route completion

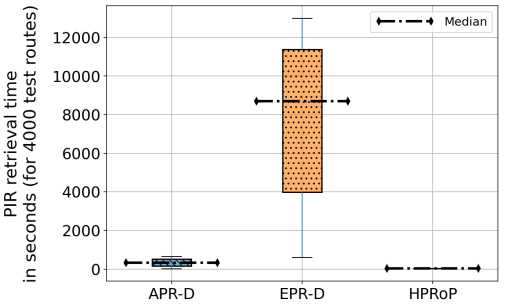

Fig. 19. Distribution of total PIR retrieval times in seconds

total processing overhead for an RPS. In this work, it is calculated based on projections derived from the preliminary experiments on SealPIR database retrieval times mentioned at the end of Sec. 5.1. The empirically derived value of roughly $\frac{1}{45000} \approx 2.22 \times 10^{-5}$ seconds per record is then used to calculate the route completion times which are shown in Fig. 19. Note that we opted to use projections here instead of simulating the actual results as the latter would take a prohibitively long time to conclude in the case of EPR-D (and, to a lesser extent, APR-D). For instance, APR-D requires $\sim$ 214 queries on average which is expected to take 5.55 minutes (333.26 seconds) per route. EPR-D is even worse, requiring $\sim$ 4,958 queries on average which would take at least 2.15 hours (7,731.47 seconds) just to complete a single route. Both are clearly impractical from the perspective of any modern RPS. This is in contrast to HPRoP which requires significantly less queries on average (at $\sim$ 25 queries) and takes only 23.55 seconds per route. This is because HPRoP's algorithm bypasses the need to explore large sections of the partition and road network graph as it already starts with a very coarse route between the origin and destination partitions at the lowest level to guide its route search. APR-D and EPR-D, in contrast, explore outwards from the origin, checking every unexplored partition or vertex on the way as per Dijkstra's algorithm.

Rev#2, Comm. #2

7.3.6 Memory Usage. Memory usage depends on the size of the route databases maintained by each partition, which is very important in a distributed environment with resource-constrained

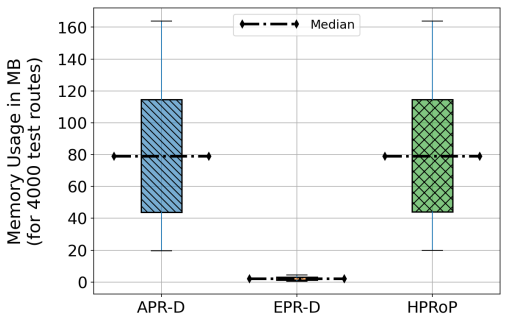

Fig. 20. Distribution of Per-Partition Memory Usage (in MB) for the different approaches

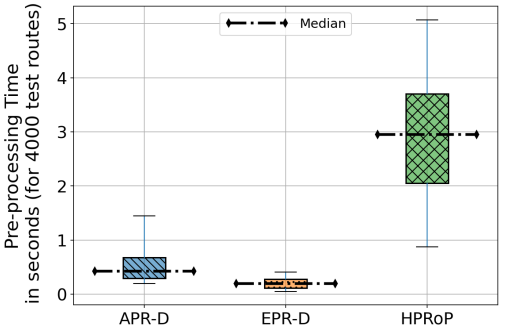

Fig. 21. Distribution of Per-Partition Pre-processing Time results (in seconds) for the different approaches

devices. Fig. 20 shows the distribution of per partition memory usage across the three approaches. APR-D and HPRoP calculate and store a similar number of routes, thus requiring an allocation of around 20-160 MB per partition. EPR-D has a *significantly* smaller memory footprint at around 0.5-4 MB per partition because it essentially stores a next-hop matrix instead. In practice, however, the memory usage of all three approaches are well within the capabilities of standard edge servers.

7.3.7 Pre-processing Time. The pre-processing time metric is total time needed to build each partition's database during the pre-processing phase, which determines how often it can be updated during operation. As shown in Fig. 21, EPR-D require less than ≤ 0.5 seconds per partition on average since it only needs to calculate the routes *inside* each partition, while APR-D requires ≤ 1.5 seconds since it also needs to calculate routes to immediately neighboring partitions in addition. Meanwhile, HPRoP needs to calculate routes inside each partition, routes to neighboring partitions, and routes to other partitions under the same parent. This results in slightly longer pre-processing times at 1-5 seconds per partition. Regardless, the pre-processing time for all three approaches are clearly fast enough to accommodate frequent updates even in under dynamic road conditions.

8 CONCLUSION

In this work, the Hierarchical Privacy-Preserving Route Planning (HPRoP) approach was proposed which combines Inertial Flow partitioning, Private Information Retrieval (PIR), and Edge Computing techniques along with a novel hierarchical route planning heuristic algorithm to produce routes that can adequately approximate the actual shortest paths while also providing *endpoint location privacy* and *route privacy*. HPRoP reliably produced routes with an *optimal route approximation* of $\alpha(r_*) \leq 1.2$, while also achieving near-optimal endpoint location privacy at $\Omega(s, d) \approx 1.0$ and good route privacy at $\Phi(Q^*) \geq 0.5$. In terms of performance, HPRoP has a route completion time of around 23.55 seconds on average which is reasonable for a privacy-preserving RPS. It's viability for deployment in a distributed/edge-based smart city context were also shown through its relatively small memory footprint (20-160 MB for each partition's database), and short pre-processing times (2-5 seconds per partition) which are well within the capabilities of conventional edge servers.

In addition, although most modern route planning algorithms can also use PIR, we didn't use them as the basis of our approach due to several reasons. For instance, *bounded-hop techniques* such as two-hop labelling [9] are the fastest known class of routing algorithms but they require computing and storing prohibitively large indices for city-sized road networks which becomes even worse in combination with PIR. *Separator-based techniques* such as Customizable Route Planning (CRP) [11] and *goal-directed techniques* such as ALT (based on A*) [18] and Arc Flags [20] have very long precomputation times, making them infeasible to use with dynamic road networks having

Rev#2, Comm. #1

edge weights that need to be updated frequently. The *hierarchical technique* called Contraction Hierarchies (CH) [16] has none of the aforementioned problems but still presents a potential route privacy risk since it must explicitly query the partitions along the actual shortest path multiple times to obtain the final path. Nevertheless, it is a good routing algorithm to consider for future PIR-based RPS work.

Rev#3, Comm. #1b

Aside from the experiments presented in Sec. 7, we also conducted tests on how well HPRoP generalizes to other road networks, but the results are omitted for conciseness. In particular, it was tested on three other large cities — New York, Shanghai, and Tokyo — by obtaining 4,000 test routes and examining the optimal route approximation distribution. For 75% of all routes, the optimal route approximation was at $\alpha(r_*) \leq 1.20$ for New York, $\alpha(r_*) \leq 1.23$ for Shanghai, and $\alpha(r_*) \leq 1.24$ for Tokyo. This indicates that HPRoP generalizes somewhat, but further research is still required.

In the future, we also plan to refine HPRoP's route planning algorithm to further improve optimal route approximation and reduce route completion times by exploring more efficient data structures for route information, and improve route privacy by considering out-of-order query execution.

ACKNOWLEDGMENTS

This work was supported by R&D for Trustworthy Networking for Smart and Connected Communities, Commissioned Research of National Institute of Information and Communications Technology (NICT), JSPS KAKENHI Grant Numbers JP21H03431 and JP19H05665, and National Science Foundation through award numbers 1647015, 1818901, CNS-1818942, SaTC-2030624, SaTC-2030611.

REFERENCES

[1] Gaurav Aggarwal, Sreenivas Gollapudi, and Ali Kemal Sinop. 2021. Sketch-based Algorithms for Approximate Shortest Paths in Road Networks. In *Proceedings of the Web Conference 2021*. 3918–3929.

[2] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. 2018. PIR with compressed queries and amortized query processing. In *2018 IEEE symposium on security and privacy (SP)*. IEEE, 962–979.

[3] Ugur Ilker Atmaca, Carsten Maple, Gregory Epiphaniou, and Mehrdad Dianati. 2021. A privacy-preserving route planning scheme for the Internet of Vehicles. *Ad Hoc Networks* 123 (2021), 102680.

[4] Barnana Baruah and Subhasish Dhal. 2022. A security and privacy preserved intelligent vehicle navigation system. *IEEE Transactions on Dependable and Secure Computing* (2022).

[5] Amos Beimel, Yuval Ishai, and Tal Malkin. 2000. Reducing the servers computation in private information retrieval: PIR with preprocessing. In *Advances in Cryptology—CRYPTO 2000: 20th Annual International Cryptology Conference Santa Barbara, California, USA, August 20–24, 2000 Proceedings 20*. Springer, 55–73.

[6] Melissa Chase and Seny Kamara. 2010. Structured encryption and controlled disclosure. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*. Springer, 577–594.

[7] Benny Chor and Niv Gilboa. 1997. Computationally private information retrieval. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. 304–313.

[8] Yu-Li Chou, H Edwin Romeijn, and Robert L Smith. 1998. Approximating shortest paths in large-scale networks with an application to intelligent transportation systems. *INFORMS journal on Computing* 10, 2 (1998), 163–179.

[9] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2003. Reachability and distance queries via 2-hop labels. *SIAM J. Comput.* 32, 5 (2003), 1338–1355.

[10] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to algorithms*. MIT press.

[11] Daniel Delling, Andrew V Goldberg, Thomas Pajor, and Renato F Werneck. 2017. Customizable route planning in road networks. *Transportation Science* 51, 2 (2017), 566–591.

[12] Daniel Delling, Andrew V Goldberg, Ilya Razenshteyn, and Renato F Werneck. 2011. Graph partitioning with natural cuts. In *2011 IEEE International Parallel & Distributed Processing Symposium*. IEEE, 1135–1146.

[13] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive* (2012).

[14] Farhad Farokhi, Iman Shames, and Karl H Johansson. 2020. Private routing and ride-sharing using homomorphic encryption. *IET Cyber-Physical Systems: Theory & Applications* 5, 4 (2020), 311–320.

[15] Sébastien Gambs, Marc-Olivier Killijian, and Miguel Núñez del Prado Cortez. 2010. Show me how you move and I will tell you who you are. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS*. 34–41.
[16] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. 2012. Exact routing in large road networks using contraction hierarchies. *Transportation Science* 46, 3 (2012), 388–404.
[17] Esha Ghosh, Seny Kamara, and Roberto Tamassia. 2021. Efficient graph encryption scheme for shortest path queries. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 516–525.
[18] Andrew V Goldberg and Chris Harrelson. 2005. Computing the shortest path: A search meets graph theory.. In *SODA*, Vol. 5. Citeseer, 156–165.
[19] Alexandra Henzinger, Matthew M Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2022. One server for the price of two: Simple and fast single-server private information retrieval. *Cryptology ePrint Archive* (2022).
[20] Moritz Hilger, Ekkehard Köhler, Rolf H Möhring, and Heiko Schilling. 2009. Fast point-to-point shortest path computations with arc-flags. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge* 74 (2009), 41–72.
[21] Takahiro Ikeda, Min-Yao Hsu, Hiroshi Imai, Shigeki Nishimura, Hiroshi Shimoura, Takeo Hashimoto, Kenji Tenmoku, and Kunihiko Mitoh. 1994. A fast algorithm for finding better routes by AI search techniques. In *Proceedings of VNIS'94-1994 Vehicle Navigation and Information Systems Conference*. IEEE, 291–296.
[22] George Rosario Jagadeesh, Thambipillai Srikanthan, and KH Quek. 2002. Heuristic techniques for accelerating hierarchical routing on road networks. *IEEE Transactions on intelligent transportation systems* 3, 4 (2002), 301–309.
[23] Sungwon Jung and Sakti Pramanik. 2002. An efficient path computation model for hierarchically structured topographical road maps. *IEEE Transactions on Knowledge and Data Engineering* 14, 5 (2002), 1029–1046.
[24] Meng Li, Yifei Chen, Shuli Zheng, Donghui Hu, Chhagan Lal, and Mauro Conti. 2020. Privacy-preserving navigation supporting similar queries in vehicular networks. *IEEE Transactions on Dependable and Secure Computing* 19, 2 (2020), 1133–1148.
[25] Yangfan Liang, Yining Liu, and Brij B Gupta. 2022. PPRP: preserving-privacy route planning scheme in VANETs. *ACM Transactions on Internet Technology* 22, 4 (2022), 1–18.
[26] Chang Liu, Liehuang Zhu, Xiangjian He, and Jinjun Chen. 2018. Enabling privacy-preserving shortest distance queries on encrypted graph data. *IEEE Transactions on Dependable and Secure Computing* 18, 1 (2018), 192–204.
[27] Samir Jordan Menon and David J Wu. 2022. Spiral: Fast, high-rate single-server PIR via FHE composition. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 930–947.
[28] Kyriakos Mouratidis. 2013. Strong location privacy: A case study on shortest path queries. In *2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 136–143.
[29] Muhammad Haris Mughees, Hao Chen, and Ling Ren. 2021. OnionPIR: Response efficient single-server PIR. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2292–2306.
[30] Vincent Primault, Antoine Boutet, Sonia Ben Mokhtar, and Lionel Brunie. 2018. The long road to computational location privacy: A survey. *IEEE Communications Surveys & Tutorials* 21, 3 (2018), 2772–2793.
[31] Michael O Rabin. 2005. How to exchange secrets with oblivious transfer. *Cryptology ePrint Archive* (2005).
[32] Peter Sanders and Christian Schulz. 2012. Distributed evolutionary graph partitioning. In *2012 Proceedings of the fourteenth workshop on algorithm engineering and experiments (ALENEX)*. SIAM, 16–29.
[33] Aaron Schild and Christian Sommer. 2015. On balanced separators in road networks. In *International Symposium on Experimental Algorithms*. Springer, 286–297.
[34] Christian Sommer. 2016. All-pairs approximate shortest paths and distance oracle preprocessing. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
[35] David J Wu, Joe Zimmerman, Jérémy Planul, and John C Mitchell. 2016. Privacy-preserving shortest path computation. *arXiv preprint arXiv:1601.02281* (2016).
[36] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *27th annual symposium on foundations of computer science (Sfcs 1986)*. IEEE, 162–167.
[37] Can Zhang, Liehuang Zhu, Chang Xu, Kashif Sharif, Chuan Zhang, and Ximeng Liu. 2020. PGAS: Privacy-preserving graph encryption for accurate constrained shortest distance queries. *Information Sciences* 506 (2020), 325–345.
[38] Lei Zhang, Jing Li, Songtao Yang, and Bin Wang. 2017. Privacy preserving in cloud environment for obstructed shortest path query. *Wireless Personal Communications* 96, 2 (2017), 2305–2322.
[39] Jun Zhou, Shiying Chen, Kim-Kwang Raymond Choo, Zhenfu Cao, and Xiaolei Dong. 2021. EPNS: Efficient privacy preserving intelligent traffic navigation from multiparty delegated computation in cloud-assisted VANETs. *IEEE Transactions on Mobile Computing* (2021).

Received XX January XXXX; revised XX XXXX XXXX; accepted XX XXXX XXXX