Hindawi
Mobile Information Systems
Volume 2018, Article ID 5186495, 18 pages
https://doi.org/10.1155/2018/5186495

*Research Article*

# Bringing Geospatial Data Closer to Mobile Users: A Caching Approach Based on Vector Tiles for Wireless Multihop Scenarios

**Chao Li,[1] Huimei Lu,[1] Yong Xiang,[2] Zhuoqun Liu,[3] Wanli Yang,[4] and Ruilin Liu[5]**

[1]*School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China*
[2]*Department of Computer Science and Technology, Tsinghua University, Beijing, China*
[3]*Neoclub Information Technology Company Limited, Shanghai, China*
[4]*The First Research Institute of the Ministry of Public Security, Beijing, China*
[5]*Department of Computer Science, Rutgers University, Piscataway, NJ, USA*

Correspondence should be addressed to Yong Xiang; xyong@csnet4.cs.tsinghua.edu.cn

Received 29 November 2017; Revised 13 February 2018; Accepted 19 February 2018; Published 8 May 2018

Academic Editor: Joaquín Torres-Sospedra



Mobile applications based on geospatial data are nowadays extensively used to support people's daily activities. Despite the potential overlap among nearby users' geospatial data demands, it has not been feasible to share geospatial data with peer wireless devices directly. To address this issue, we designed a scheme based on vector tiles to organize spatial data and proposed a system named GeoTile for geospatial data caching and sharing. In GeoTile, a tile request from the mobile client relies on multihop communication over intermediate nodes to reach the server. Since GeoTile enables all network nodes to cache and process geospatial data tiles, requests may be handled before they actually reach the server. We implement the GeoTile prototype system and conduct comprehensive real-world experiments to evaluate the performance. The result shows that the GeoTile system can serve vector tiles for users conveniently and friendly. In addition, the caching mechanism based on vector tiles can substantially reduce the response time and network throughput under the wireless multihop scenarios.

## 1. Introduction

With the ubiquitous penetration of mobile devices, especially smartphones, mobile applications based on geospatial data are nowadays considered as one of the necessities of people's daily life. About half of mobile users around the world have map applications installed in their own devices [1]. People consume geospatial data for local services (e.g., point of interest recommendation and navigation) and also generate local data to share among different applications (e.g., [2]) through crowdsourcing [3, 4].

Despite the benefits brought by the map-related application, accessing geospatial data incurs nontrivial data consumption and sometimes can even be annoying. For instance, free public hot spots may be too sparse to provide real-time data updates, especially in developing countries. Mobile data usage over the cellular network, on the other hand, can be expensive, leading to a high cell phone bill and consequently user's cost concern [5]. Offline data packages, that is, map data downloaded beforehand, are considered as a naive solution to this problem. Nevertheless, while the offline data packages are usually bulky (up to tens of megabytes, e.g., [6, 7]), they are now treated as atomic and exclusive: (i) offline map data can be used only when the entire data package is downloaded and does not support partial updates and (ii) such data packages are formatted as per application, and users may find it difficult to share among peers [8, 9].

More recently, researchers explore to tackle the challenge of geospatial data access by exploiting the wireless communication and distributed storage among multiple mobile devices. Nowadays, the storage space of mobile devices is usually adequate to carry a considerable amount of data, which makes every mobile device a potential content provider. Considering the prevalence of the mobile device and the fact that similar geospatial data are frequently

requested from hot regions, such as urban centers or transportation hubs [10], the geospatial data requested by a user can be very likely available in other nearby users' devices. This idea is known as "Device-to-Device (D2D)" [11], and if implemented properly, it can result in an instant and cost-effective solution for users' geospatial data needs.

In this paper, with the idea of D2D for geospatial data sharing in wireless multihop scenarios, we present GeoTile, a novel system that enables geospatial data caching and access in a distributed and collaborative manner. GeoTile adopts the GeoJSON format with optional compression to partition the large-scale geospatial data into small units, that is, vector tiles, for efficient data sharing. Moreover, GeoTile also explores the possibility to use the peer cache and amalgamation technique to reuse the data cached locally in order to reduce both the response time and network data usage.

Specifically, the paper makes three main contributions:

(i) Based on the state-of-art geospatial data formats, we propose a new geospatial exchange standard, which constructs vector tiles with optional compression to efficiently index, exchange, and organize geospatial data with various granularities and level of details (LoDs).

(ii) We implement the GeoTile system for geospatial data caching and sharing. GeoTile not only contains the vector tile system in the data-provider server but also enables network intermediate nodes to cache vector tiles and to make responses to the geospatial data request that overlaps with the local cache.

(iii) We evaluate the GeoTile system in real-world settings and show that the GeoTile system provides efficient vector tile service in a convenient and user-friendly way. Meanwhile, the evaluation result also demonstrates that the caching mechanism based on vector tiles can substantially reduce the response time, wireless network traffic, and the dependency on root data sources in distributed scenarios.

The rest of this paper is organized as follows. First, we revisit the related works in Section 2. Then, in Section 3, we describe the main idea of the vector tile system. After that, Section 4 elaborates on the design of the GeoTile system by introducing tile service component and mobile prototype component, respectively. Performance evaluations and analysis of GeoTile under realistic environments are presented in Section 5. Finally, we conclude the paper together with a short future work discussion in Section 6.

## 2. Related Works

In this section, we take a brief review of literatures on multiple aspects concerning our work in this paper.

*2.1. Tile-Based Mapping and Geospatial Information Service.* Map tiling has been proved as the most effective way of delivering spatial data over the recent decade [12]. By creating tiny map fragments at numerous predefined scales, map tiling enables users to just retrieve the pieces of data needed for the current view to reduce the data transfer volume and response time. Meanwhile, tile caching at the server side reduces the computational resource requirement and increases the concurrent request rate a server can handle.

Lately, the mobile geographical information system (GIS) [13] stretches users' ability of geospatial information access to mobile devices. However, the mobile GIS follows the client/server architecture as in traditional Internet GIS [14]: an active connection to the GIS server is required for a user to fetch the desired spatial data. In our work, by utilizing cached geospatial data at intermediary nodes, users can get required data from locations as near as possible instead of accessing server directly.

*2.2. The Emergence of Vector Tiles.* Raster tiles dominated the earlier years of map tiling (especially for mobile clients), mainly because of its simplicity and the limited computation capacities on the mobile client [15]. However, as mobile users' demand for a more interactive and informative mapping service growth [16], vector tiles have come to light recently: vector tiles provide readable, descriptive, and extensible content, which facilitates the provision of data supplementation, manipulation, and customization. For instance, by using vector tiles, users can scale the map smoothly without the problem of pixelation [17]. Expressive features, such as highlight, labeling, styling, and animation, are also well supported by popular libraries and mainstream browsers.

To bring the idea of the vector tile into reality, one scheme of the vector tile is proposed in [18]. Those objects crossing multiple tiles are clipped to the boundary they belong to. Although it makes little impact on visualization, the integrity of original data is damaged and extra points are brought in. Mobile users will find it challenging to collect full information of objects and restore them. Another shortage of splitting tiles according to fixed areas is that the size of tiles varies greatly. In order to improve the efficiency of transmission, in [19] tiles are generated according to data density, so the tiles are of irregular shapes. But this method is inimical to sharing as it is hard to define global rules that guarantee each tile can be identified and reused by everyone.

*2.3. Quadtree Indexing Structure.* The pyramid-structured quadtree indexing structure has become the de facto standard of online map and has been adopted by most major web map service providers. The most representative model is the XYZ Map Tiling System (a.k.a. Slippy Map) [20]. As a matter of fact, such structure, owing to its characteristics of top-down hierarchy and multiscale [21], has wide applications in a variety of areas of science and technology (e.g., image processing [22], indexing [23], artificial intelligence [24], movement prediction [25], privacy protection [26], and just to name a few). In Slippy Map, multiple zoom levels are defined to represent different scales. Tiles follow a recursive subdivision strategy. The region of one tile on zoom level $N$ is equally divided into four portions, namely, four tiles on zoom level $N + 1$. Each tile is indexed by a unique group of values $(x, y, z)$. With the naming scheme, each node can identify demanded tiles unambiguously. Users or intermediate

nodes can clearly know if they have contents of certain locations. It is the foundation of tile sharing among mobile counterparts. In our work, we follow this indexing structure and naming scheme to identify and arrange vector tiles.

*2.4. The Encoding of Vector Data.* Various formats can be the container of vector data [27, 28]. ESRI Shapefile and OSM XML are mainly used for online data release. Shapefile [29] stores data in three separate parts (geometry of features, positional index of features, and attributes of features) and professional tools are necessary to process it. OSM XML [30] stores complex relation information in addition to the geometry data and metadata. The user has to traverse the file along the relations to get complete object geometry data. GML (Geography Markup Language) [31] and KML (Keyhole Markup Language) [32] are two formats based on XML. KML has a more concise schema, while GML provides advanced features to describe complex maps. However, being subject to the heavyweight nature of XML syntax, KML and GML are usually bigger in size [33].

Comparatively speaking, lightweight formats are more appropriate for data-interchange among mobile devices. GeoJSON [34] is an open standard format of widespread use for vector tiles. It is dedicated for encoding geographic data along with various nonspatial attributes. All structures and columns are human readable and straightforward to users. We here note that the significance of the readability is that it enables users to find information or make modifications easily without any specialized software. Another JSON-based format TopoJSON [35] can achieve smaller size, and reduce data redundancy by storing the shared boundary only once. As the expense, it impairs the independence and completeness of original objects. Thus, it is not proper for further data editing and sharing. Moreover, the coordinates have been transformed to integers so that it could not be edited or resolved before decoding. Protocol buffers [36], developed by Google, is a binary serialization method. Although it may have better performance and much smaller size [37], it is not human readable and specialized tools, or libraries to resolve them are not widespread among mobile users. Additionally, the schema (proto file) which contains a description of the data structure and data types should be made aware by users beforehand to parse received data.

*2.5. Compression Techniques Applied in GIS.* Compression is usually deemed as an effective way to preserve and distribute geospatial data in GIS systems, as it reduces the amount of data on transmission and storage. General-purposed methods can also be applied directly to geodata units, such as ZIP used in KMZ [38], LZMA and DEFLATE used in [39, 40], gzip used in [41], and LZW used in [42]. The description of the above mainstream compression methods can be found in [43]. Apart from these, some dedicated-purpose methods are proposed to make optimizations for certain formats. For example, methods are summarized in [44] for GML. In our work, compression is also of importance as the bandwidth and storage resources are quite precious in wireless scenario. Generally speaking, every compression method has its characteristics. We should take factors such as data type, platform, processing speed, and application scenario into consideration to make the most suitable decision. We apply a new promising compression method, Brotli [45], to our GeoTile system to see whether expected effect can be achieved. The reasoning of selecting Brotli is explained in Section 3.3.

*2.6. Simplification and Quantization of Vector Geospatial Objects.* Another way to address the oversize problem of geodata units is the simplification and quantization. Both of them can be categorized into lossy compression. The former one reduces the visually insignificant vertice of objects, while the latter one rounds and shortens the coordinates. Simplification (a.k.a. generalization) [46, 47] is often used in progressive transmission [48, 49]. By dividing the original vector object into multiresolution representations, a subset of the data returns to user upon request at first, and then subsequently it will be refined incrementally. The main goal of progressive transmission is avoiding transmitting too much detailed data at one time to manage near real-time response at client side, that is, no longer than 1 or 2 seconds [50].

However, to our best knowledge, most progressive transmission methods require stable Internet connections as the prerequisite to perform frequent and timely data transferring. But under mobile circumstances, multiple rounds of data transmitting may not finish due to intermittent links. When it is impossible to establish connection again, the client has to resort to other nodes for missing parts of the object, which will result in longer delay. In aspect of correctness, as simplification techniques are not mature and still under research, features with distortions or topological errors may appear [51, 52]. Moreover, as object information is divided, much differential data should be maintained, which incurs higher complexity on data identification, exchanging and caching for mobile devices. In order to better suit the mobile environment, we will use units that are relatively complete and independent to accomplish the goal of facilitating the sharing of partial data among users and providing the demanded data for requesters cooperatively within a moderate delay.

The quantization method shrinks the data size by reduction of the digits and precision of the real-valued coordinates [53]. Compared with simplification, it can maintain the topological relationship of objects. If data are not being excessively quantized, the difference on visualization is inconspicuous. Even so, we want to emphasize that the visualization is not the only use of geospatial data units. Driven by the enthusiasm of crowdsourcing and rapid development of related technologies, the amount of volunteered geographic information (VGI) grows tremendously [54, 55]. Correspondingly, researches have shown that geodata from OSM in some parts of the world is locationally and semantically more accurate, complete, and timely than proprietary commercial datasets or information provided by government agencies [56, 57]. With such data, users can perform editing and updating, distance measurement, pathfinding [58], surrounding POI lookup [59], and so on. Our future work is also towards this orientation. Undoubtedly, the loss in data

precision will undermine the value of data which users have collected and contributed for future sharing and potential use.

*2.7. Caching for Data Dissemination.* Caching has been regarded as a highly effective solution for data dissemination in mobile wireless networks. With replicas residing closer to the requesting client, we can reduce the communication traffic over the limited wireless channel and hence shorten the overall latencies. Caching is especially crucial to improve data accessibility in multihop mobile networks [60, 61] because intermittent links often impede users from contacting the root data sources. Recently, as D2D communications gain research attentions, caching on terminal devices also starts to show its significance [62]. For that reason, researchers are making efforts to answer questions around caching such as where to cache (placement) [63, 64], what to cache (content type, popularity, and mobility awareness) [65], and how to cache (replacement policies) [66, 67]. In this work, we adopt the idea of caching for vector tile sharing, and we thus introduce the definition of vector tile layers and naming scheme for data caching and exchanging.

## 3. Vector Tile System

The foundation of data caching and exchange is the format of the data. In this section, we elaborate on the definition of our vector tiles, that is, the basic unit for data caching and exchange in the GeoTile system. Specifically, we cover the partition, naming, and format of vector tiles, together with the optional compression approach and the client-side amalgamation to reduce the data communication volume, and finally, the vector tile's support to custom extensions.

*3.1. Generation of Geospatial Data Tiles.* Tiles are the fundamental units for data caching and exchanging in the GeoTile system. When breaking bulky geospatial data into smaller tiles, each of them can be sent separately to effectively speed up the map delivery. We use the naming scheme of the Slippy Map system, the most popular way of organizing map tiles. In this way, we can utilize the data provided by other online resources for the layers that our server cannot provide or to supplement our data to add additional customization.

In our vector tile system, each tile is associated with a coordinate in the form of $(x, y, z)$, where $x$, $y$, and $z$ represent the longitude index, latitude index, and zooming level of the tile, respectively. We use the following steps to build the vector representation for the tile.

*3.1.1. Geographic Boundary of a Tile.* We create vector tiles according to the rules defined in XYZ Quadtree Indexing System [68] to determine the location and the scope of the tile. Specifically, given the coordinate $(x, y, z)$ of a tile $t$, we can get the west longitude line and the north latitude line of $t$ using (1) and (2). Likewise, the east longitude line and the south latitude line of $t$ can also be determined by reusing (1) and (2) on the tile $t'$: $(x + 1, y + 1, z)$, which is diagonally adjacent to $t$ at the bottom right direction. This is because that the east longitude line and south latitude line of $t$ overlap with the west longitude and the south latitude lines of $t'$:

$$\text{lon} = \frac{x}{2^z} \cdot 360 - 180, \tag{1}$$

$$\text{lat} = \arctan\left(\sinh\left(\pi - \frac{y}{2^z} \cdot 2\pi\right)\right) \cdot \frac{180}{\pi}. \tag{2}$$

Table 1: Several rules from zoom level reference.

| Category | Type | Zoom level | | | | |
|---|---|---|---|---|---|---|
| | | 13 | 14 | 15 | 16 | 17 |
| Highway | Primary | ✓ | ✓ | ✓ | ✓ | ✓ |
| Highway | Tertiary | ✓ | ✓ | ✓ | ✓ | ✓ |
| Highway | Pedestrian | | ✓ | ✓ | ✓ | ✓ |
| Railway | Station | ✓ | ✓ | ✓ | ✓ | ✓ |
| Amenity | Hospital | | | ✓ | ✓ | ✓ |
| Amenity | Restaurant | | | | | ✓ |
| Shop | Supermarket | | | | ✓ | ✓ |

*3.1.2. Level of Detail.* As demanded by most online map services, we apply the idea of LoD in our tile system. At zooming levels of low values, only limited objects with coarse-grained appearance are included into the tile, whereas more objects will present on the map when a user zooms into a more granular level. To achieve this, we extract a set of rules (some instances are shown in Table 1) from the well-known render engine Mapnik [69]. Based on these rules and the zoom level of $t$, we can tell which objects should be included in $t$.

Based on the LoD rules, we include in the tile $t$ those objects which have an intersection with $t$ and are qualified to show at zoom level $z$. This implies that if an object crosses multiple tiles, all these tiles will contain it. This approach will introduce some redundancy, whereas the integrity of objects is not broken. Users can trust that what they receive is the description of the entire object without worrying about seeking for missing parts when they need to render full objects or to perform calculations.

*3.1.3. Formatting and Postprocessing.* We select GeoJSON as the format of vector tiles for the following reasons. First, the accuracy of raw geometries, such as position coordinates, will be kept in GeoJSON. It is meaningful as we expect that many calculations and navigations could be performed locally to reduce the reliance on infrastructures and to obtain speedups. Second, it is supported natively by HTML5 and mapping tools such as Leaflet and OpenLayers, which eases the utilization at the terminal [70]. Third, it is human readable. Users can easily make modifications and contributions to original data with common editors. Finally, it is more compact compared to XML-based formats.

After the tile $t$ is published using the GeoJSON format, we make some optimizations. First, we use the compact mode of GeoJSON to remove unnecessary spaces and line breaks. Then, we delete fields whose value is null. Lastly, we remove some fields which are meaningless to users, such as the information of database or server.

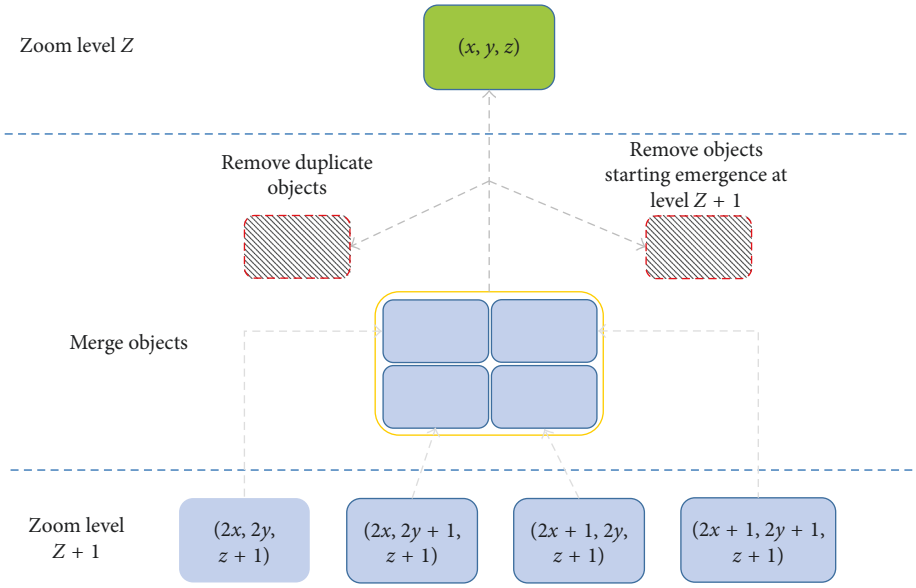


Figure 1: The procedure of amalgamating local tiles.

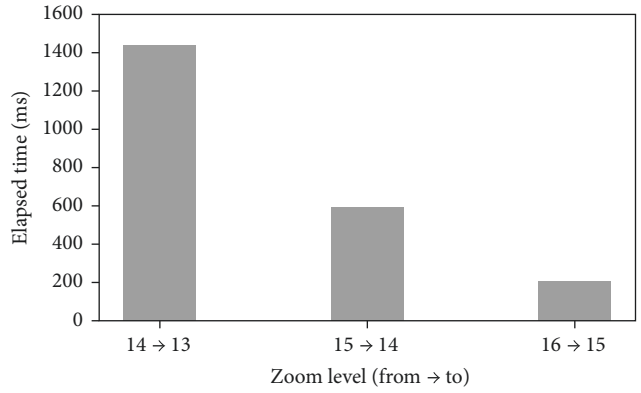


Figure 2: Time cost of tile amalgamation on Nexus 7. The result in each setting is based on the average processing time of 30 random tile amalgamations.

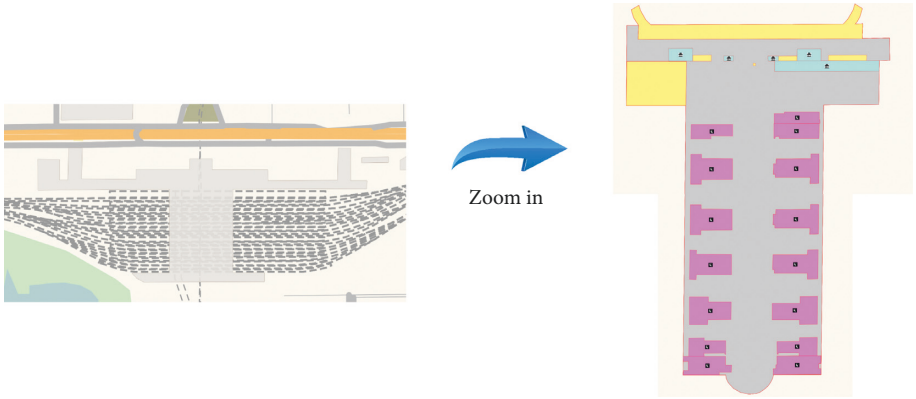


Figure 3: An example of interior map of Beijing west railway station.

*3.2. Tile Compression.* Given the tile definition and formatting presented in Section 3.1, one can expect the size of tile grows substantially as the zoom level goes up. The main reason is that some large objects, such as trunk roads, railways, or district boundaries, are more likely to be included by more tiles with the scope of each tile enlarged. To handle this issue, we apply compressions on the tiles using Brotli [45], a generic-purpose lossless compression

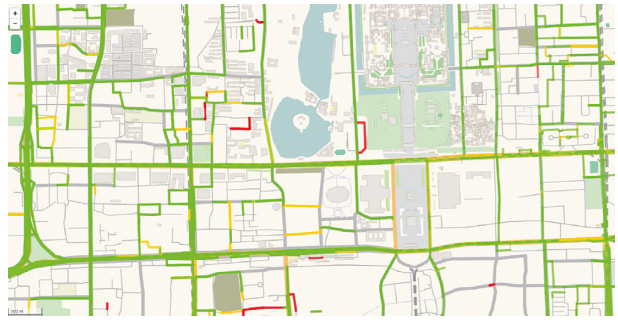

Figure 4: Map with traffic condition.

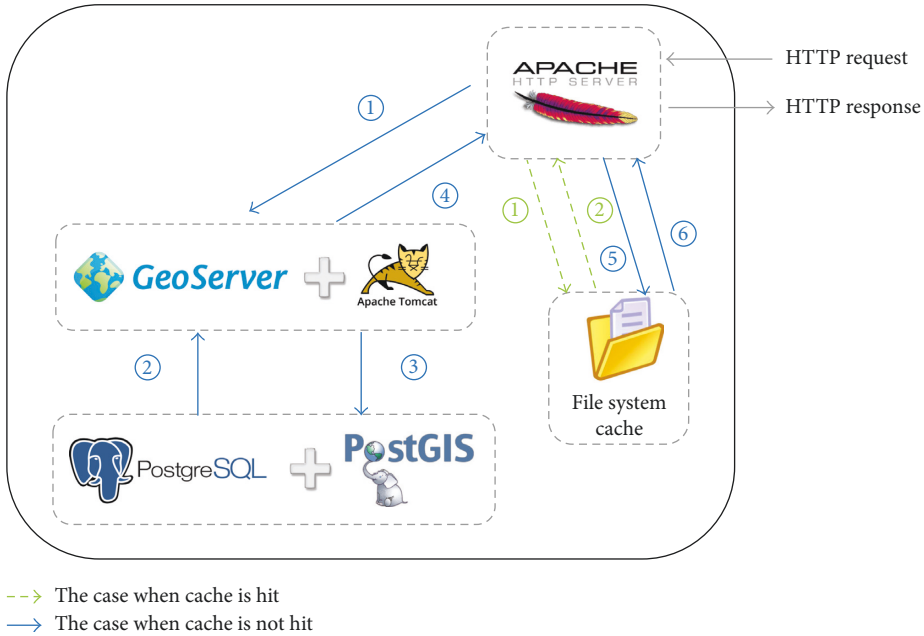


Figure 5: The structure and working flow on server.

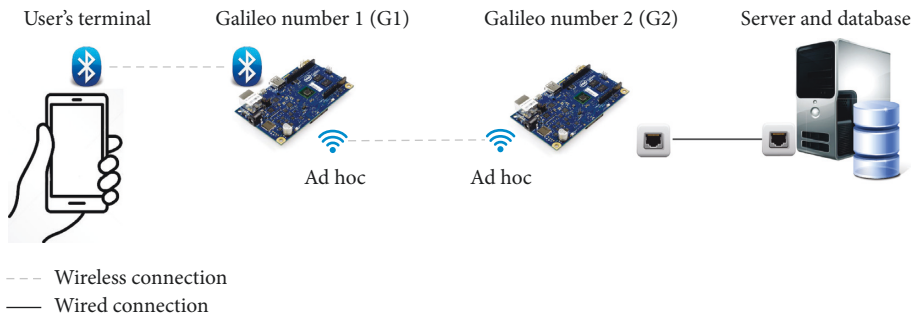


Figure 6: The structure of mobile prototype component.

algorithm presented by Google recently. Brotli achieves better compression ratio, especially for small-sized text file [71, 72], and provides fast decompression speed at the client mobile device, regardless of the compression quality levels. Although Brotli is relatively slow on compression, considering most compressions are performed at the powerful server side and only happen infrequently, we choose Brotli as our default tile compression method and decide to use compression level 10 to trade-off the compression time and the compression ratio. Based on both the benchmark test [73] and the real test on our server, Brotli high-compression levels, for example, level 10 or level 11, achieve about 20%~30% savings in the file size compared to the middle-quality levels (level 7~9). However, level 11, which provides the best compression ratio, reduces the file size by 3%~5% than level 10 but incurs 30% longer compression time.

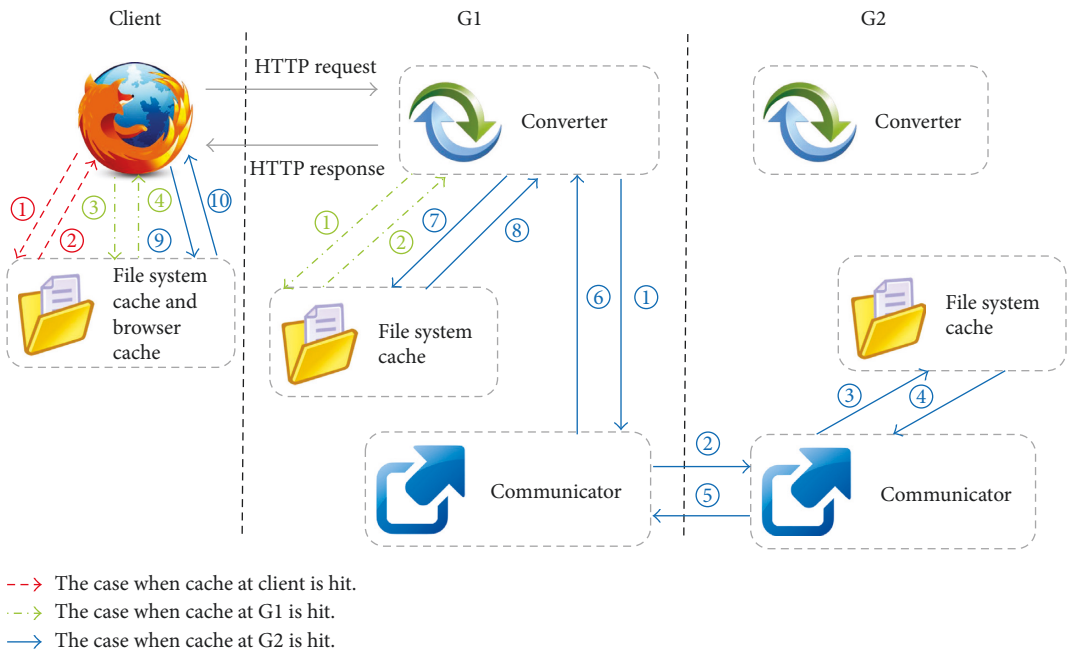


Figure 7: The working flow of the mobile prototype when the cache is available at one of the locations.

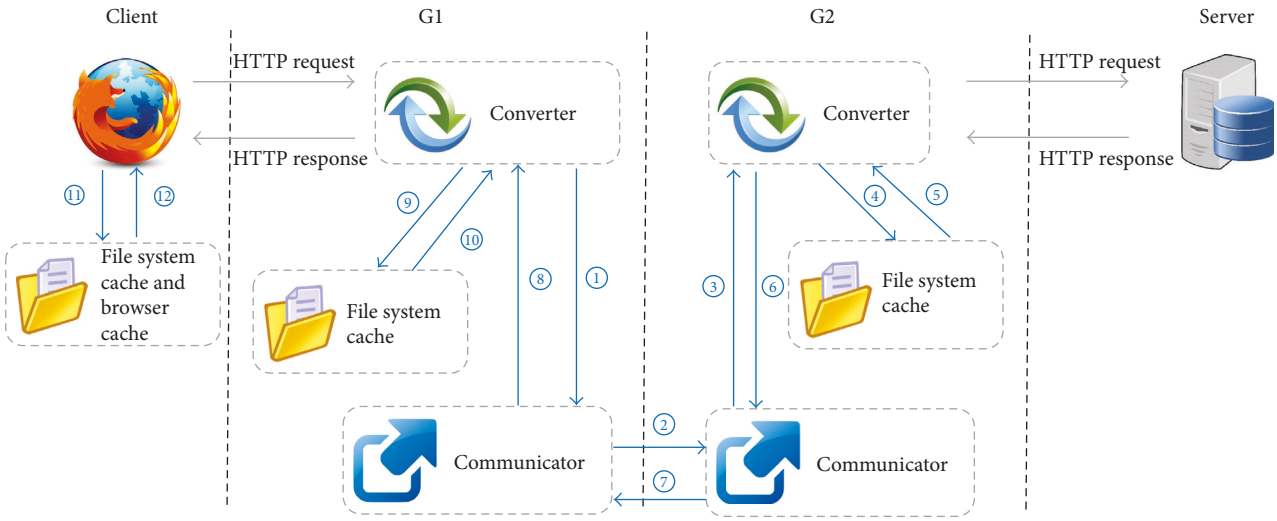


Figure 8: The working flow of the mobile prototype when the cache is not available.

*3.3. Client-Side Tile Amalgamation.* When users are under extremely poor network conditions, even the compressed tiles may take too much time to fetch. Tiles cached at the terminal will help shorten the waiting period and lessen the data traffic. Based on the rules for organizing the tiles above, we can use local tiles to generate corresponding upper layer tiles directly without requests. As illustrated in Figure 1, given the $(x, y, z)$ coordinate of target tile $t$, we check the existence of the four lower-layer tiles in the local cache, that is, tile $(2x, 2y, z+1)$, tile $(2x+1, 2y, z+1)$, tile $(2x, 2y+1, z+1)$, and tile $(2x+1, 2y+1, z+1)$. If all the four tiles exist, we will merge all objects into a new tile and remove duplicate objects or those objects whose emerging zoom level is $z+1$.

As shown in Figure 2, the computation overhead for amalgamation is trivial for smaller tiles, for example, hundreds of milliseconds for tiles in level 14 or 15. For tiles of larger size, for example, level 13 and above, even though a long amalgamation processing period may be needed, it is still useful when a user has very limited network data budget or when the network connection is so poor that fetching the tile instead of generating it using amalgamation takes even longer time.

*3.4. Support for Extended Information.* Besides the basic geometric information, the flexibility of GeoJSON brings numerous new possibilities to the tile unit. Users can append

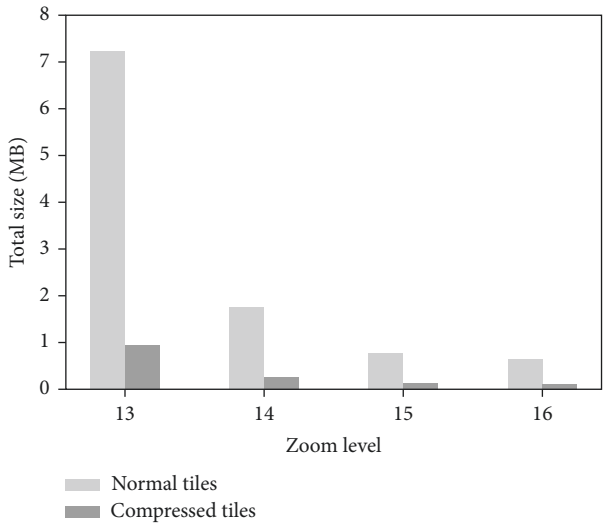


Figure 9: Transferred data size of the initial map of each experiment group.

Table 2: The cache availability of experiments of normal tiles.

| Group | Device: Terminal | G1 | G2 | Server |
|---|---|---|---|---|
| None | ⊘ | ⊘ | ⊘ | ⊘ |
| Server | ⊘ | ⊘ | ⊘ | GeoJSON |
| G2 | ⊘ | ⊘ | GeoJSON | GeoJSON |
| G1 | ⊘ | GeoJSON | GeoJSON | GeoJSON |

extra key-value pairs to make original objects more informative. To exemplify, we implement an extension of building interior structure and an extension of road traffic conditions on our vector map. Figure 3 shows the zoom in effect for the interior structure of Beijing West Railway Station. To implement this effect, we add a "room" attribute, which indicates the type of room, and a "floor" attribute, which indicates which floor the room locates, to the geometry records of objects. Likewise, we add a "speed" attribute to road objects in the basic map to include traffic conditions calculated in a travel time estimation system and visualize it in Figure 4.

With the support for the extended information, it can be expected that maps with additional specific functions could be easily transformed from the basic vector map. For instance, Beijing Government has released 12 kinds of specialized map recently focusing the convenience for civilians, including air quality, housing, and tourism. The customization potential of

Table 3: The cache availability of experiments of compressed tiles.

| Group | Device: Terminal | G1 | G2 | Server |
|---|---|---|---|---|
| None | ⊘ | ⊘ | ⊘ | ⊘ |
| Server | ⊘ | ⊘ | ⊘ | Brotli |
| G2 | ⊘ | ⊘ | Brotli | Brotli |
| G1 | ⊘ | Brotli | Brotli | Brotli |

vector maps enables the integration and presentation of diversified information. A user can just switch the content layer defined by the customized data to browse different information on the same basic vector map.

## 4. System Structure and Design of GeoTile

*4.1. GeoTile Server.* Figure 5 demonstrates the architecture of our GeoTile server, which provides the vector tile service for mobile applications. The implementation of the GeoTile server is built upon Ubuntu OS 14.04 and a series of open source software packages [74]. The GeoTile server consists of four main modules, that is, the Apache HTTP server, the caching system, the web server based on GeoServer, and the data storage module based on PostgreSQL [75]. All the raw data from OpenStreetMap are imported into PostgreSQL with PostGIS [76] support. When a request arrives at the GeoTile server, Apache will first check whether the demanded tile is available at the local cache. If the cache hits, the tile is returned as an HTTP response right away. Otherwise, Apache will pass the request to the GeoServer. The GeoServer deployed on top of Tomcat queries the PostgreSQL database and publishes the data using GeoJSON format through a web feature service (WFS), where our vector tile logic presented in Sections 3.1, 3.2, and 3.4 is implemented. Finally, the tile is returned through Apache with also the result updated in the cache.

As stated above, we use an on-the-fly way to generate tiles gradually rather than preprocessing. The design decision is based on the following considerations. (i) Not all tiles are equally demanded by users. For the sake of resource efficiency, we should keep those tiles needed by most users. (ii) Generating all vector tiles in advance may be a waste when the map data get updated periodically, for example, for the road condition use case, and may exhaust the resource of the server.

*4.2. Mobile Prototype Component.* In order to test the feasibility and performance of our design under the realistic environment, we implement a mobile multihop prototype as another part of GeoTile. The topology of the prototype is

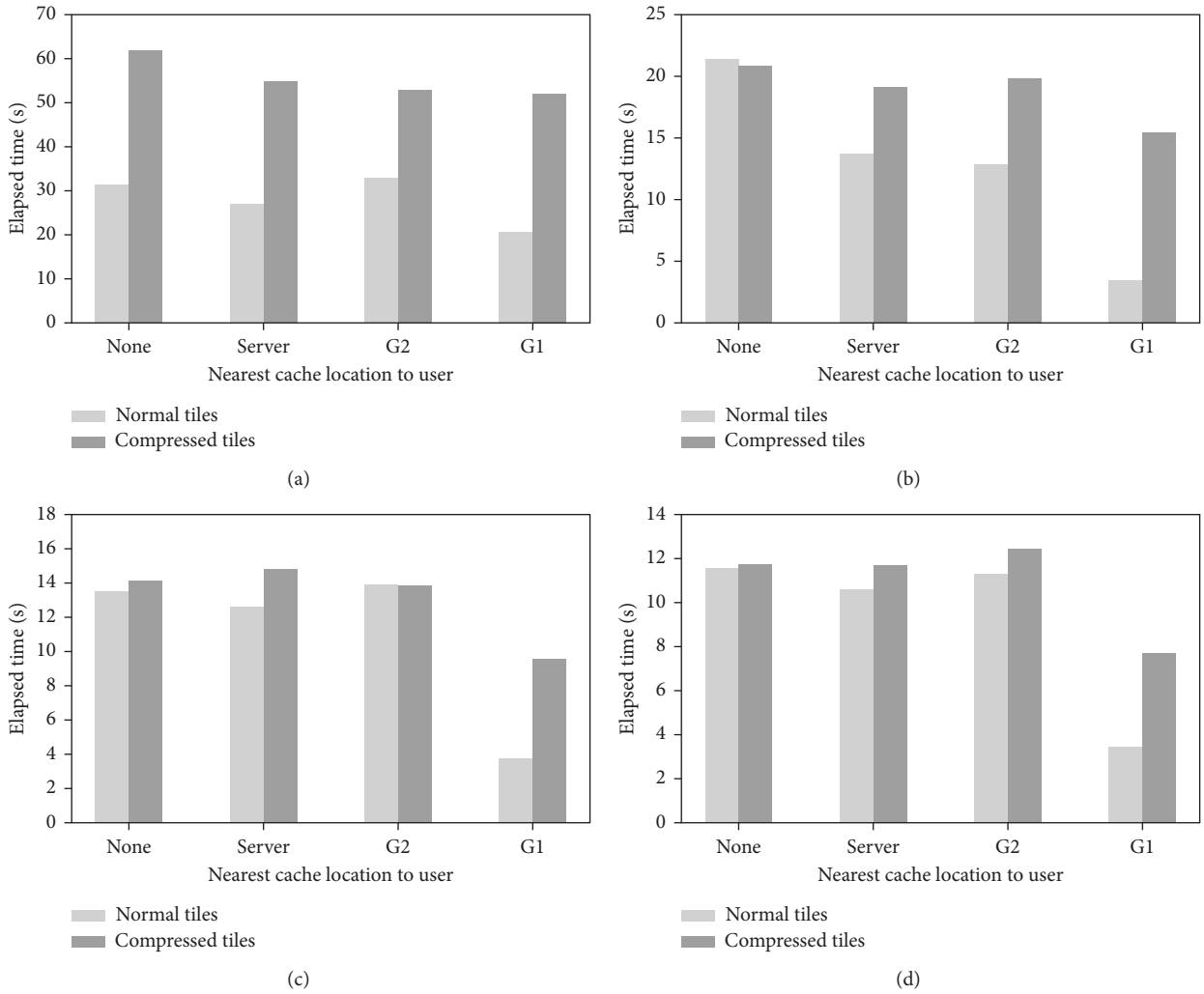


Figure 10: Time cost from terminal's view on showing the initial map (cache location varies at different zoom levels). Zoom level: (a) 13; (b) 14; (c) 15; (d) 16.

demonstrated in Figure 6. The requesting client (also called terminal) device is Nexus 7 (Qualcomm Snapdragon S4 Pro 8064 Quad-Core 1.5 GHz, 2 GB memory). We use a cross-platform Firefox browser and a webpage based on HTML5 and Leaflet for vector map request and presentation. The two intermediate wireless nodes are Intel Galileo Gen 2 board (Intel Quark SoC X1000 400 MHz, 256 M DRAM, and 8 GB Class-4 MicroSD card as storage). The terminal connects to the first intermediate node (called G1 hereafter) via Bluetooth. The reason why we choose Bluetooth is that Bluetooth 4.0 is energy-efficient compared to Wifi-based device-to-device methods to provide satisfied transmission for small data traffic [77]. The connection between G1 and the second intermediate node (called G2 hereafter) is an ad hoc network to support mobility. G2 downloads data from the GeoTile server through a wired link. The converter module inside intermediary nodes is implemented with Anyproxy [78].

Both G1 and G2 have the ability to cache received tiles. The demanded tile available at the nearest location to the user is used to respond to a user request. Note that, when we say a tile is "available" locally, it can either be cached locally or can be generated from other tiles by using the method described in Section 3.3. The procedure of how the mobile prototype works is illustrated in Figures 7 and 8.

## 5. Experiment and Evaluation

In this section, we conduct practical experiments on GeoTile from multiple angles. We start by investigating the performance of the implementation from user's standpoint. More specifically, we quantify how much time and data transfer are needed to load the initial map at the client side and the impact of available cache location, together with the improvement brought by using tile compression. After that, we take a closer look into the mobile prototype component. Through studying the detailed time cost of each phase, we will gain a deeper understanding of whether the system works properly and accordingly find the room for further improvements.

*5.1. Performance from User's Perspective: Response Time.* In this part, we evaluate the performance of GeoTile in terms of the response time from a user's standpoint. In particular, we

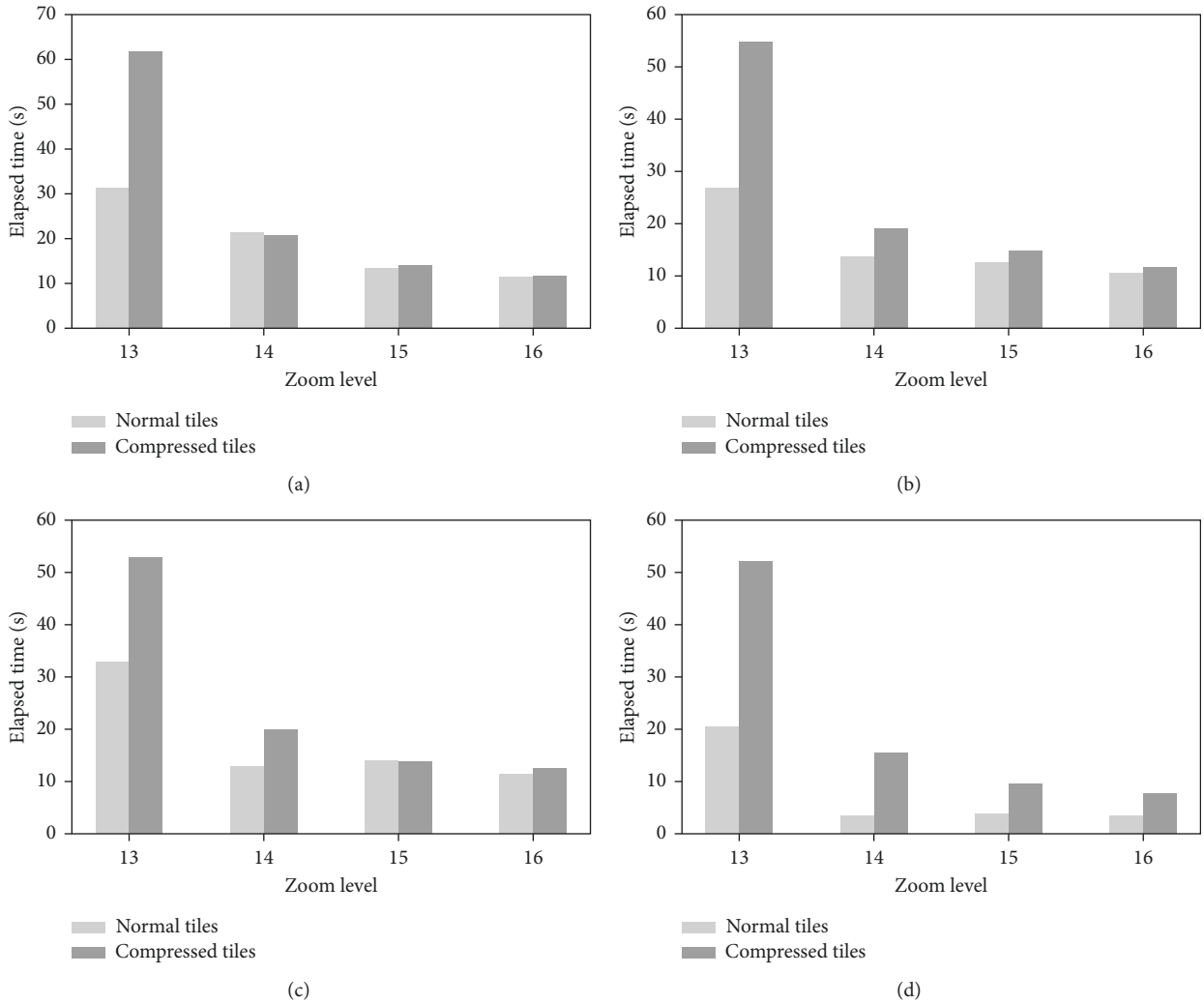


Figure 11: Time cost from terminal's view on showing the initial map (zoom level varies at different cache locations): (a) cache is unavailable at all the locations; (b) cache is available at server; (c) cache is available at G2; (d) cache is available at G1.

measure the time needed to load a complete initial map in the browser. The response time is measured from the moment when the request is issued from the terminal until when all data are successfully returned to the terminal and loaded into the browser. The total volume of the transferred data for presenting the initial map in each group is given in Figure 9. Each experiment group is repeated for three times, and the average number is presented as the result. The available location(s) of cache in each experiment setting is given in Tables 2 and 3, for normal tiles and compressed tiles, respectively.

Figure 10 demonstrates the response time when the cache-hit location varies at different zoom levels. Generally speaking, it takes less time to show the map completely when the cache hit is closer to the requesting client.

For the normal tile groups, cache hit at the GeoTile server saves about 10%~20% overall time compared to that without any cache hits. The cache hit at G1 can significantly save the response time, resulting in approximately 25%~30% of the response time when the cache hit only happens at the GeoTile server. This means when the cache hits just 1 hop away from the user, the user could get them quite instantly without reaching the server. The response time when the cache hits at G2 is sometimes higher than others, and we will explain the reason in the following parts. For the compressed tile groups, as the required data volume of layer 16~14 is very low, not much time is used for tiles generating, transferring, and processing, so the effect brought by the cache at server or G2 is subtle. However, cache hit at G1 still achieves the most response time save.

Figure 11 presents the response time when cache hits at the same place, but the zoom level varies. We can find that the response time for both normal tile and compressed tile decreases as the zoom level number increases. For the normal tile, the response time differences between level 16 and level 14 are not very significant, as they all have the similar size of data transfer. At level 13, since the data transfer size is 3~4 times higher, much more time is spent on transmission among nodes, rendering at the terminal.

Figures 10 and 11 also show that the response time of the compressed tiles is generally longer than that of normal tile groups under the same conditions. This means that although the compressed tiles have less data amount compared to

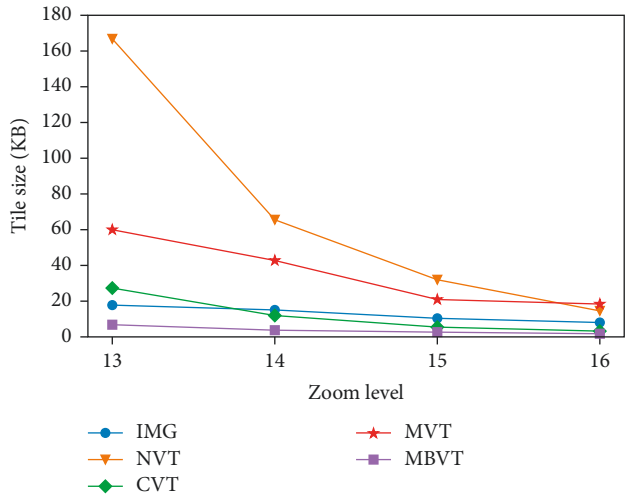


Figure 12: Size of different tile types.

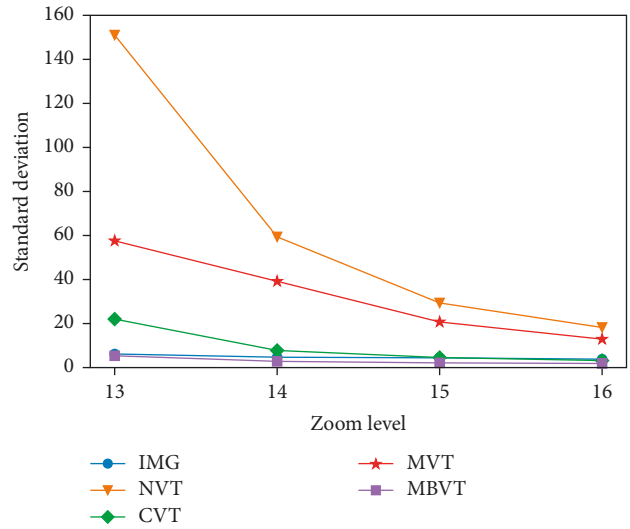


Figure 13: The standard deviation of size of different tile types.

normal tiles, the compression and decompression undermine the time saved within the transmission procedure. Therefore, we conclude that when the quality of the network is poor, the compressed tiles are more recommended as they have smaller size and better tolerate the connectivity failure. When the network quality is good, or when the computation capability of the user's device is not so powerful, normal vector tiles without compression should be preferred to achieve shorter response time.

*5.2. Performance from User's Perspective: Data Usage.* In addition to the response time, the data usage, that is, the volume of data transfer, is a metric that a user may also be interested in. To evaluate the performance of GeoTile in data transfer efficiency, we conducted two experiments to evaluate (i) the size comparison of different tile types and (ii) the data transfer volume when a user zooms out. The first dimension clarifies the characteristics of our tile scheme in the aspect of size and shows the effect of our tile compression method, while the second additionally shows the effect of client-side tile amalgamation.

*5.2.1. Tile Size Comparison.* In this part, we compare the size of five different tile types, that is, OSM PNG (IMG), Mapzen Vector Tiles (MVT) [79], Mapbox Vector Tiles (MBVT) [80], our vector tile format (called as NVT), and our compressed tile format (called as CVT). Figures 12 and 13 present the mean and the standard deviation of the tile data usage at different zoom levels. IMG, MVT, and MBVT, which are the tile types available publicly, are included as the comparison baselines. The key difference between MVT and NVT is that MVT clips vector objects along tile boundaries. We sampled 100 tiles of each kind of tiles from each zoom level.

From Figure 12, we can clearly see that the size of CVT is comparable with that of image tiles and is significantly lower than MVT and NVT, which proves that the compression provides a great advantage on reducing the size of vector tiles. The tile size of NVT and MVT increases as zoom level goes up because the scope of each tile becomes larger resulting in more objects being included. At lower zoom levels, NVT and MVT have similar average tile size. However, at upper zoom levels, tile size of NVT becomes much higher than that of MVT due to the redundancy discussed in Section 3.1.

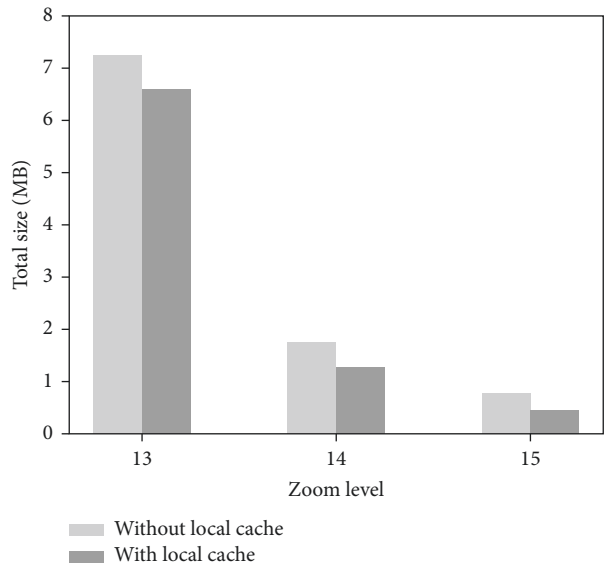


Figure 14: The comparison on transferred data size with and without utilizing the local cache.

From Figure 13, the standard deviation and average size of the MBVT are both the minimum among all these types. This can be ascribed to the following: (1) this format is based on protocol buffers, and its binary serialization shows good effect on reducing size; (2) it also clips objects around the tile boundary; and (3) the objects are simplified and the coordinates are rounded. Likewise, the two metrics of image tiles are also stable and keep in very low level because they have fixed resolution and do not have a direct linear correlation with how much geospatial information they contain. The standard deviation of NVT is very high at upper zoom levels, indicating that the size of NVT distributes over a wide range due to the fact

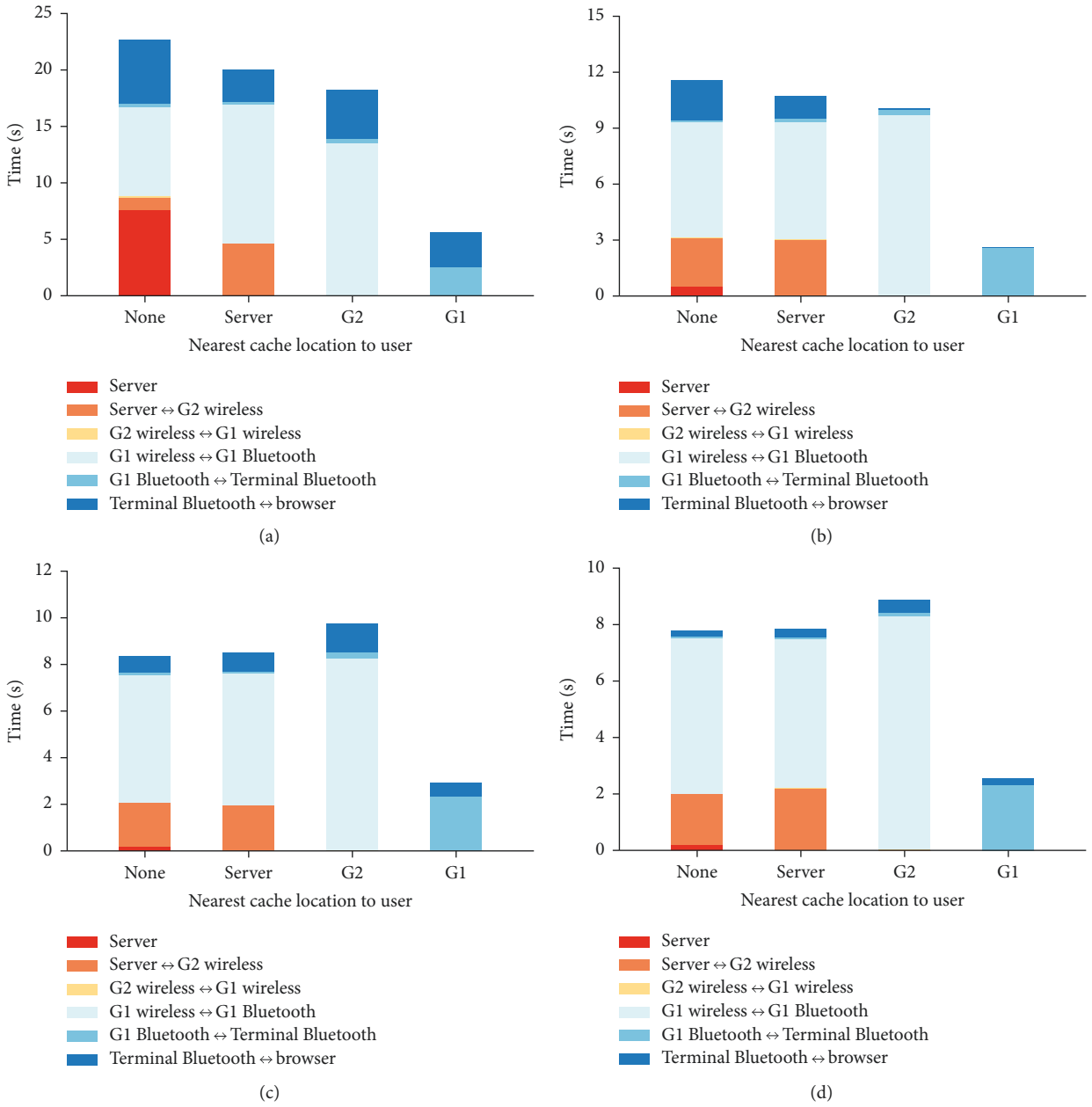


Figure 15: Time cost by each phase of transmission procedure of normal tiles. Zoom level: (a) 13; (b) 14; (c) 15; (d) 16.

that some tiles incorporate large objects. After compression, the size differences among tiles get diminished. So when GeoJSON and compression are used together, all tiles returned to users are guaranteed to be relatively small. Although some of NVT tiles of upper layers may be oversized, such zoom levels are probably too macroscopic for daily use of pedestrians or drivers. According to statistical results, for commonly used zoom levels, the majority of NVT does not contain an exceeding amount of data and is acceptable for GeoTile.

*5.2.2. Effect of Client-Side Tile Amalgamation.* During this experiment, we use normal vector tiles and enable the local cache at the terminal. After loading a complete initial map in the browser, we make a zoom out operation to an upper zoom-level view. As presented in Section 3.3, GeoTile will check the local cache and generate as many usable tiles as possible and request only the missing ones. Theoretically, if the map screen includes 20 tiles, amalgamation can save up to 25% data usage. From Figure 14, we can see that in both experiments when zooming from level 16 to 15 and when zooming from level 15 to 14, 4 tiles are generated from the local cache and about 20% of transferred data are saved. In the experiment, when zooming from level 14 to level 13, the number of tiles grows up in the level 13 view, resulting in more tiles to be requested. Additionally, because of the size limit of the local cache, many level-13 tiles cannot find the corresponding level-14 tiles in the cache for amalgamation. These two factors cause the saving proportion to become small. If there are enough tiles in the cache, more tiles could be produced locally.

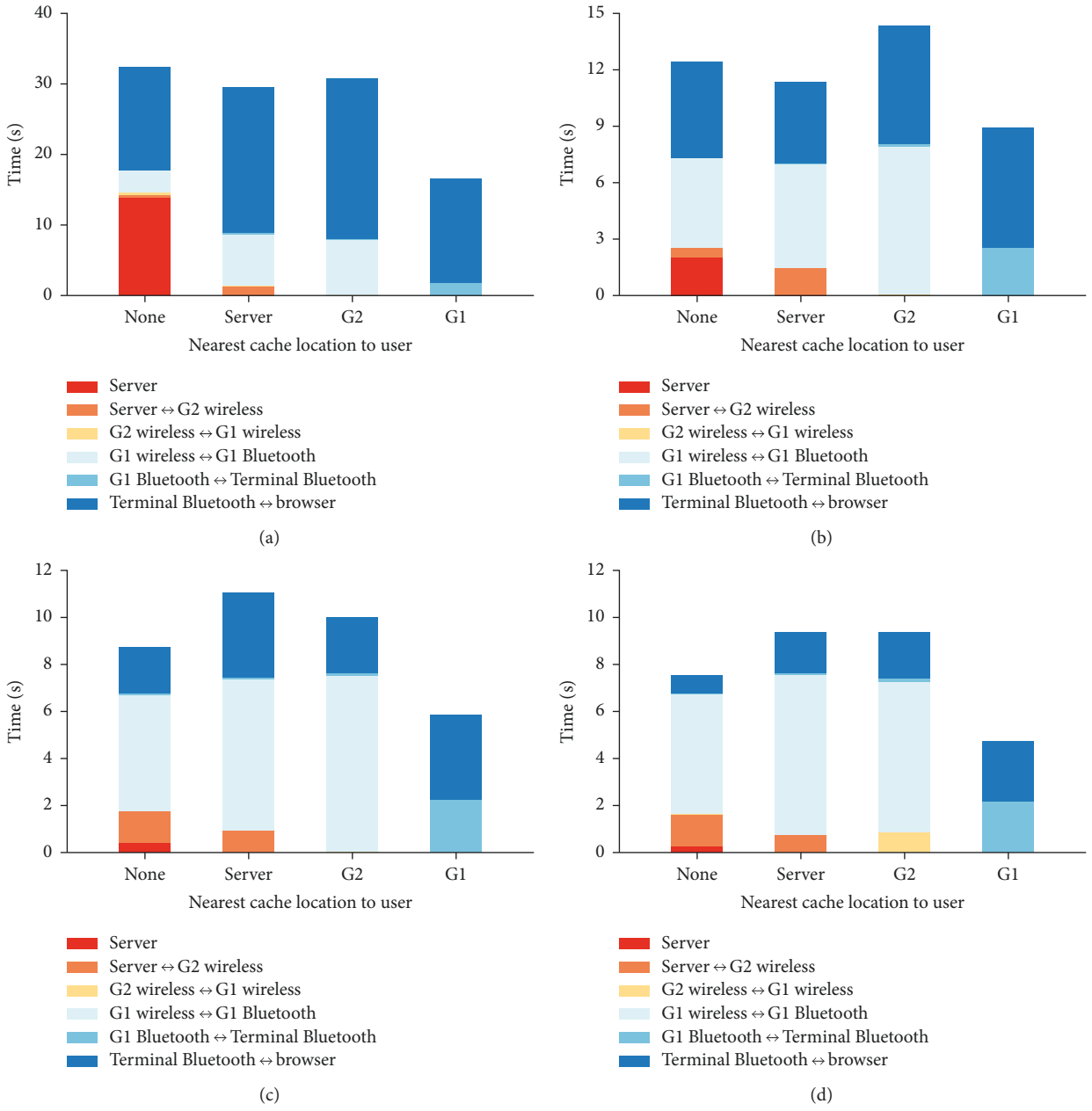


Figure 16: Time cost by each phase of transmission procedure of compressed tiles. Zoom level: (a) 13; (b) 14; (c) 15; (d) 16.

*5.3. Performance from Prototype's Perspective: Initial Map.* In this part, we will investigate the detail of the mobile prototype to find out how much time each phase costs. This will help us realize the advantages and deficiencies of our design or implementation for future improvements.

Figures 15 and 16 illustrate the total time cost for a tile from request to finally load in the browser. The total time represented by a bar in the figures is partitioned into various colors to represent the time taken in different phases. Each phase contains the time cost in two directions, that is, both request and response, and reports the average value of all tiles in one experiment.

Figure 15 shows the result of normal tile groups. It can be observed that the server has to generate demanded tiles when there is no cache. However, this phase only costs trivial time unless the tile size is large (e.g., level 13). When the cache is available on the server, no generation time is needed. But the conversion time between request and data at G2 takes longer, indicating that the requests and data are queued at this point. The time cost of transmission between G2 and G1 is nearly invisible. It is mainly because of the following reasons: First, there is no conversion operation at this phase. Second, the transmission speed of ad hoc port is fast enough and is not the bottleneck. Third, we use a RAMDisk for caching to accelerate file I/O performance. These reasons can be also applied to explain the short time between the Bluetooth port of G1 and the terminal.

Another thing to be addressed is the long time spent between the G1 wireless port and the Bluetooth port. As we

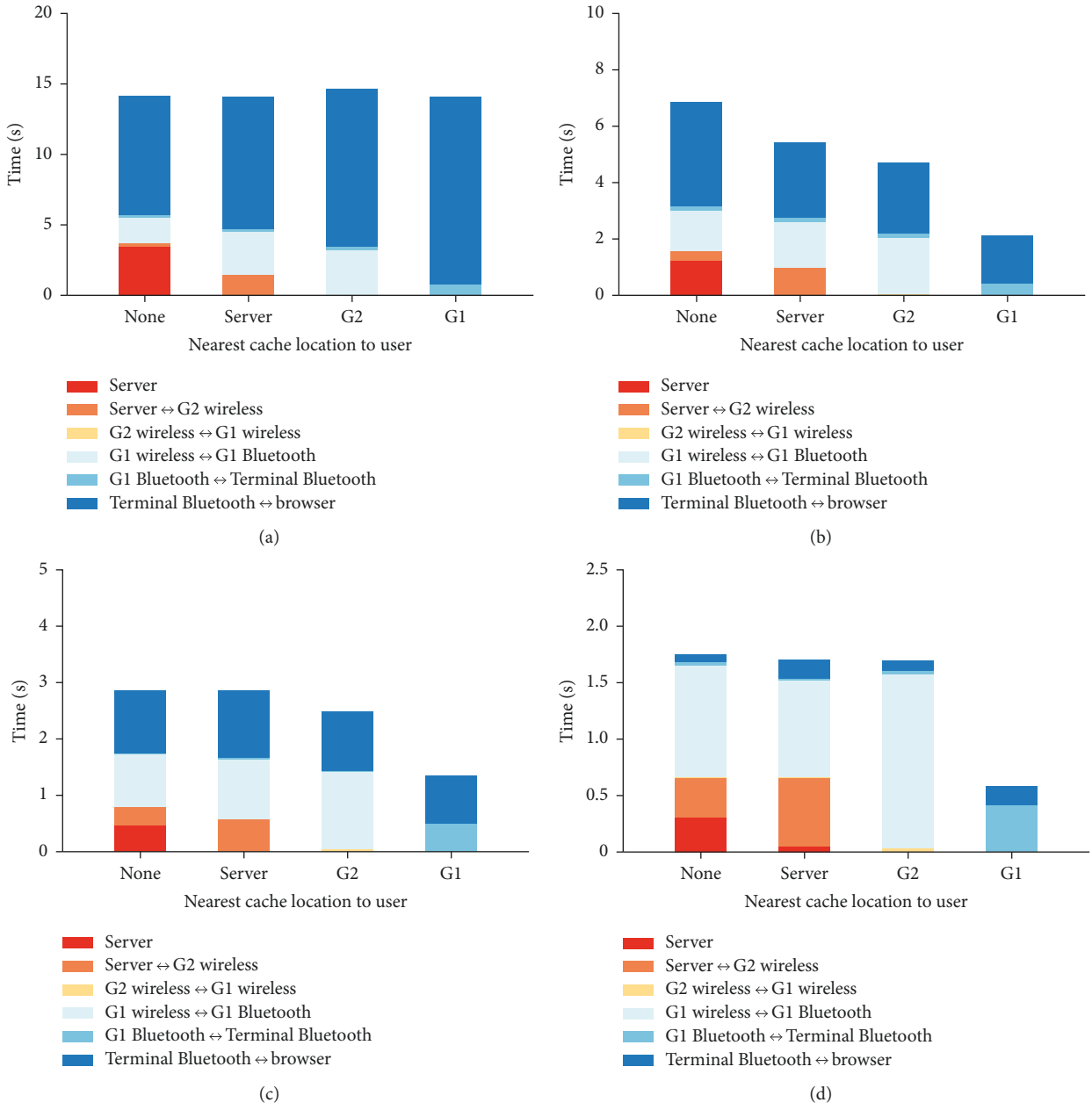


Figure 17: Time cost by each phase of transmission procedure of normal tiles by move operation. Zoom level: (a) 13; (b) 14; (c) 15; (d) 16.

have verified that the link capacity is not fully utilized and the speed of accessing local cache is fast, we can deduce the possible reasons as follows: (i) the support for parallel processing of Anyproxy (implemented in NodeJS) is limited, and we do not find any better alternatives yet; (ii) the computational ability of Galileo is restricted so that the processing speed is relatively slow. These will result in the phenomenon that tiles wait for processing or transfer at the converter. This problem is more obvious when cache hits at G2. In this situation, the request can be turned into data by G2 immediately and send back to G1, which will make the queuing phenomenon at G1 much worse. This will also explain why sometimes G2 group takes longer overall time than other groups. When cache hits at G1, data is returned from G1's Bluetooth port quickly. Note that the time spent at the terminal shows considerable randomness across the groups. We consider that it may have something to do with the internal working mechanisms of the browser which we cannot control so far.

Figure 16 shows the result of compressed tile groups. Compared to the normal tile groups, the time cost at server side and terminal side rises up notably due to compression and decompression. This is also the main reason that leads to longer overall response time. On the other hand, because the size of tiles is reduced, the time needed by transferring is shorter and queuing problem is mitigated to some extent.

*5.4. Performance from Prototype's Perspective: Move Operation.* In this part, we conduct experiments on move and drag

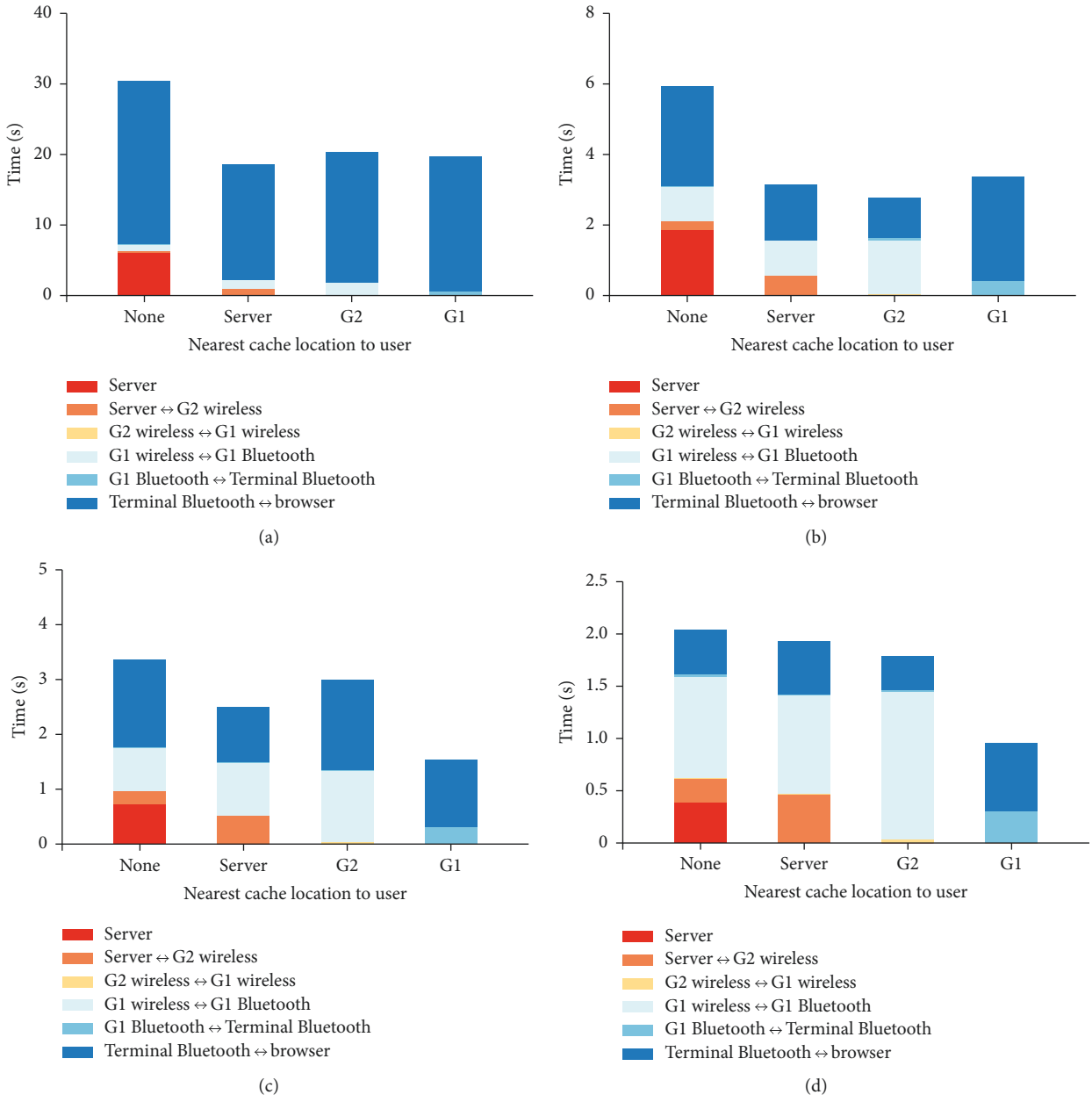


Figure 18: Time cost by each phase of transmission procedure of compressed tiles by move operation. Zoom level: (a) 13; (b) 14; (c) 15; (d) 16.

operations. The action of loading the initial map is taken as the case in which many tiles are requested simultaneously, while the drag and move behavior represents the case where only a few supplementary tiles are requested. We intend to test how our prototype performs under the situation of light pressure.

In the experiment, we execute two move operations, and each time 4 new tiles are requested upon each movement. Figure 17 is the result of normal tile groups. Figure 18 is the result of compressed tile groups. It can be noticed that the queuing phenomenon at the server side and the G2 side is clearly mitigated, proving that too many requests at one time will overload the system's capability. Moreover, the average overall time of each tile goes down, indicating that GeoTile is more suitable for exchanging a small amount of data. Another problem that needs explanation is that, in the compressed tile groups, it takes much longer time at the browser. It is mainly because the move operation will incur reprocessing and rerendering data on the whole screen, so the overall time is not less than that in initial map experiments.

*5.5. Evaluation Summary.* Based on the experiment results, our evaluation reveals the following findings. (i) Loading the initial map made of smaller tiles (i.e., tiles each of which covers a smaller area) is less time-consuming compared with larger tiles. The waiting time to load a same initial map becomes shorter as the location of the cache hit draws nearer to the user. However, the problem of queuing caused by the converter module undermines the advantage of caching to some extent. (ii) The compressed tiles can reduce the transmission time and

queuing time, but the extra compression and decompression time are required. They are suitable for those users who would like to get tiles more quickly and use a device with good computational abilities. They will also benefit scenarios with poor communication quality or intermittent links. (iii) A small number of tiles provoked by user's move or drag operation is well supported by GeoTile. Users can incrementally obtain vector tiles on demand efficiently. (iv) The tile amalgamation approach can save around 20% of user's traffic given the tiles of the initial map of lower zoom level in the cache. If there is plenty of tile resource in the cache, users can generate more new tiles without requesting.

## 6. Conclusion

In this paper, we present GeoTile, a novel system to share and exchange vector geospatial data in the distributed and collaborative way. GeoTile adopts the GeoJSON format with optional compression to partition the large-scale geospatial data into small units, that is, vector tiles, to enable efficient data sharing. In addition, GeoTile also explores the possibility to use peer cache and amalgamation technique to reuse the data cached locally or nearby in order to reduce both the response time and network data usage.

We implement the GeoTile system, including the tile service component and mobile prototype component. The experiments show the following findings: (i) Caching at locations closer to the user will shorten the overall waiting period. (ii) The tile amalgamation based on the local cache can save time for some requests. (iii) Users can get tiles through GeoTile in an incremental way effectively. (iv) Using compressed tiles can relieve the problem of oversizing and overloading and is suitable for the networks with low quality.

In our future work, we plan to explore a more flexible and progressive way for client-side amalgamation so that if there is a part of needed tiles available, by complementing small missing tiles, the amalgamation can also be conducted. In addition, we also plan to introduce the disruption tolerant techniques to better support the mobile environment with intermittent connectivity and to make better use of the distributed cache resources.

## Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

## Acknowledgments

The authors would like to express their gratitude to Dr. Jingbang Wu for his support on wireless prototype system and suggestions on experiments and to Yu Zhang for his support on road traffic information.

## References

[1] *The Mobile Consumer-A Global Snapshot*, http://www.nielsen.com/content/dam/corporate/us/en/reports-downloads/2013%20Reports/Mobile-Consumer-Report-2013.pdf.

[2] M. D. Dunlop, M. Roper, M. Elliot, R. McCartan, and B. McGregor, "Using smartphones in cities to crowdsource dangerous road sections and give effective in-car warnings," in *Proceedings of SEACHI 2016 on Smart Cities for Better Living with HCI and UX*, pp. 14–18, San Jose, CA, USA, November 2016.

[3] C. Heipke, "Crowdsourcing geospatial data," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 65, no. 6, pp. 550–557, 2010.

[4] M. Goetz and A. Zipf, "The evolution of geo-crowdsourcing: bringing volunteered geographic information to the third dimension," in *Crowdsourcing Geographic Knowledge: Volunteered Geographic Information (VGI) in Theory and Practice*, D. Sui, S. Elwood, and M. Goodchild, Eds., pp. 79–95, Springer, Berlin, Germany, 2014.

[5] *2016 Global Mobile Consumer Survey: US Edition*, https://www2.deloitte.com/content/dam/Deloitte/us/Documents/technology-media-telecommunications/us-global-mobile-consumer-survey-2016-executive-summary.pdf.

[6] *Comparing Google Maps and HERE Maps Offline*, https://360.here.com/2015/11/17/comparing-google-maps-and-here-maps-offline/.

[7] *Osmand Offline Map Data Package Indexes List*, http://osmand.net/srtm-countries/.

[8] *Google Maps Help-Download Areas and Navigate Offline*, https://support.google.com/maps/answer/6291838.

[9] TIME, *How to Use Google Maps Offline*, TIME, New York, NY, USA, 2016, http://time.com/4437599/how-to-use-google-maps-offline/.

[10] O. Cats, Q. Wang, and Y. Zhao, "Identification and classification of public transport activity centres in Stockholm using passenger flows data," *Journal of Transport Geography*, vol. 48, pp. 10–22, 2015.

[11] Y. Wu, S. Yao, Y. Yang et al., "Challenges of mobile social device caching," *IEEE Access*, vol. 4, pp. 8938–8947, 2017.

[12] A. Clouston and M. P. Peterson, "Tile-based mapping with opacity," in *Developments in the Theory and Practice of Cybercartography Applications and Indigenous Mapping*, D. Taylor and T. P. Lauriault, Eds., pp. 79–95, Elsevier, Amsterdam, Netherlands, 2014.

[13] R. Abdalla, "Mobile GIS and location-based services (LBS)," in *Introduction to Geospatial Information and Communication Technology (GeoICT)*, pp. 83–103, Springer, Berlin, Germany, 2016.

[14] W. Alsabhan and S. Love, "Platforms and viability of mobile GIS in real-time hydrological models: a review and proposed model," *Journal of Systems and Information Technology*, vol. 13, no. 4, pp. 425–444, 2011.

[15] J. Gaffuri, "Toward web mapping with vector data," in *Proceedings of International Conference on Geographic Information Science, GIScience 2012*, pp. 87–101, Columbus, OH, USA, September 2012.

[16] A. Poplin, "How user-friendly are online interactive maps? Survey based on experiments with heterogeneous users," *Cartography and Geographic Information Science*, vol. 42, no. 4, pp. 358–376, 2015.

[17] S. Torbert, *Applied Computer Science*, CRC Press, Boca Raton, FL, USA, 2nd edition, 2016.

[18] V. Antoniou, J. Morley, and M. Haklay, "Tiled vectors: a method for vector transmission over the web," in *Proceedings of the 9th International Symposium on Web and Wireless Geographical Information Systems*, pp. 56–71, Maynooth, Ireland, December 2009.

[19] A. Dufilie and G. Grinstein, "Feathered tiles with uniform payload size for progressive transmission of vector data," in

*Proceedings of the 13th International Symposium on Web and Wireless Geographical Information Systems*, pp. 19–35, Seoul, South Korea, May 2014.

[20] *Google Map—Tile and Coordinate Specification*, https://developers.google.com/maps/documentation/javascript/maptypes.

[21] H. Samet, "The quadtree and related hierarchical data structures," *ACM Computing Surveys*, vol. 16, no. 2, pp. 187–260, 1984.

[22] T. Markas and J. Reif, "Quad tree structures for image compression applications," *Information Processing and Management*, vol. 28, no. 6, pp. 707–721, 1992.

[23] E. El-Qawasmeh, "A quadtree-based representation technique for indexing and retrieval of image databases," *Journal of Visual Communication and Image Representation*, vol. 14, no. 3, pp. 340–357, 2003.

[24] H. Noborio, T. Naniwa, and S. Arimoto, "A quadtree-based path-planning algorithm for a mobile robot," *Journal of Robotic Systems*, vol. 7, no. 4, pp. 555–574, 1990.

[25] M. Xu, D. Wang, and J. Li, "DESTPRE: a data-driven approach to destination prediction for taxi rides," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp'16*, pp. 729–739, Heidelberg, Germany, September 2016.

[26] X. Ju and K. G. Shin, "Location privacy protection for smartphone users using quadtree entropy maps," *Journal of Information Privacy and Security*, vol. 11, no. 2, pp. 62–79, 2015.

[27] C. Portele, *Encoding of Geographic Information*, Springer, Berlin, Germany, 2011.

[28] F.-J. Behr, K. Holschuh, D. Wagner, and R. Zlotnikova, "Vector data formats in Internet based geoservices," in *Advances in Web-based GIS, Mapping Services and Applications*, pp. 349–367, CRC Press, Florida, USA, 2011.

[29] *ESRI Shapefile Technical Description*, https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf.

[30] *OSM XML Format*, http://wiki.openstreetmap.org/wiki/OSM XML.

[31] R. Lake, "The application of geography markup language (GML) to the geological sciences," *Computers and Geosciences*, vol. 31, no. 9, pp. 1081–1094, 2005.

[32] *KML Tutorial*, https://developers.google.com/kml/documentation/kml_tut.

[33] S. Steiniger and A. J. Hunter, "Data structure: spatial data on the web," in *International Encyclopedia of Geography: People, the Earth, Environment and Technology*, Wiley, Hoboken, NJ, USA, 2016.

[34] *GeoJSON*, http://geojson.org/geojson-spec.html.

[35] *TopoJSON*, https://github.com/topojson/topojson-specification.

[36] *Protocol Buffers Overview*, https://developers.google.com/protocol-buffers/docs/overview.

[37] A. Sumaray and S. K. Makki, "A comparison of data serialization formats for optimal efficiency on a mobile platform," in *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, p. 48, Kuala Lumpur, Malaysia, 2012.

[38] *Compressed KMZ Files*, https://developers.google.com/kml/documentation/kmzarchives.

[39] W. Li, M. Song, B. Zhou, K. Cao, and S. Gao, "Performance improvement techniques for geospatial web services in a cyberinfrastructure environment—a case study with a disaster management portal," *Computers, Environment and Urban Systems*, vol. 54, pp. 314–325, 2015.

[40] H. Shao and W. Li, "A comprehensive optimization strategy for real-time spatial feature sharing and visual analytics in cyberinfrastructure," *International Journal of Digital Earth*, pp. 1–20, 2018.

[41] S. Sikdar, K. Teymourian, and C. Jermaine, "An experimental comparison of complex object implementations for big data systems," in *Proceedings of the 2017 Symposium on Cloud Computing*, pp. 432–444, Santa Clara, California, September 2017.

[42] Y. Zhao, Z. Cheng, H. Dong, and J. Fang, "One efficient compression method used on WebGIS," in *Proceedings of the 19th International Conference on Geoinformatics*, pp. 1–5, Shanghai, China, June 2011.

[43] D. Salomon and G. Motta, *Handbook of Data Compression*, Springer, Berlin, Germany, 5th edition, 2010.

[44] Q. Wei, J. Guan, S. Zhou, and X. Wang, "A new and effective approach to GML documents compression," *Computer Journal*, vol. 57, no. 11, pp. 1723–1740, 2014.

[45] *Brotli Compressed Data Format*, https://tools.ietf.org/html/rfc7932.

[46] B. P. Buttenfield, "Transmitting vector geospatial data across the Internet," in *Proceedings of the International Conference on Geographic Information Science*, pp. 51–64, Boulder, CO, USA, September 2002.

[47] L. Zhang, L. Zhang, Y. Ren, and Z. Guo, "Transmission and visualization of large geographical maps," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 66, no. 1, pp. 73–80, 2011.

[48] B. Yang, R. Purves, and R. Weibel, "Efficient transmission of vector data over the Internet," *International Journal of Geographical Information Science*, vol. 21, no. 2, pp. 215–237, 2007.

[49] R. Miao, J. Song, and M. Feng, "A feature selection approach towards progressive vector transmission over the Internet," *Computers and Geosciences*, vol. 106, pp. 150–163, 2017.

[50] R. E. Roth, "Interactive maps: what we know and what we need to know," *Journal of Spatial Information Science*, no. 6, pp. 59–115, 2013.

[51] M. Guo, Y. Huang, Q. Guan, Z. Xie, and L. Wu, "An efficient data organization and scheduling strategy for accelerating large vector data rendering," *Transactions in GIS*, vol. 21, no. 6, pp. 1217–1236, 2017.

[52] W. Shi and C. Cheung, "Performance evaluation of line simplification algorithms for vector generalization," *Cartographic Journal*, vol. 43, no. 1, pp. 27–44, 2006.

[53] *OpenLayers 3 Vector Rendering with Topology Simplification*, https://boundlessgeo.com/2014/03/openlayers-vector-rendering/.

[54] J. J. Arsanjani, P. Mooney, M. Helbich, and A. Zipf, "An exploration of future patterns of the contributions to OpenStreetMap and development of a contribution index," *Transactions in GIS*, vol. 19, no. 6, pp. 896–914, 2015.

[55] J. J. Arsanjani, M. Helbich, M. Bakillah, and L. Loos, "The emergence and evolution of OpenStreetMap: a cellular automata approach," *International Journal of Digital Earth*, vol. 8, no. 1, pp. 76–90, 2015.

[56] S. Maguire and M. Tomko, "Ripe for the picking? Dataset maturity assessment based on temporal dynamics of feature definitions," *International Journal of Geographical Information Science*, vol. 31, no. 7, pp. 1334–1358, 2017.

[57] D. Zielstra, H. H. Hochmair, P. Neis, and F. Tonini, "Areal delineation of home regions from contribution and editing patterns in OpenStreetMap," *ISPRS International Journal of Geo-Information*, vol. 3, no. 4, pp. 1211–1233, 2014.

[58] J. Itoi, M. Sasabe, J. Kawahara, and S. Kasahara, "An offline mobile application for automatic evacuation guiding in outdoor environments," *Scientific Phone Apps and Mobile Devices*, vol. 3, no. 1, 2017.

[59] D. Bahrdt, "OSMFIND: fast textual search on OSM data—on smartphones and servers," in *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*, pp. 35–42, Orlando, FL, USA, November 2013.

[60] S. Glass, I. Mahgoub, and M. Rathod, "Leveraging MANET based cooperative cache discovery techniques in VANETs: a survey and analysis," *IEEE Transactions on Wireless Communications*, vol. 19, no. 4, pp. 2640–2661, 2017.

[61] W. Gao, G. Cao, A. Iyengar, and M. Srivatsa, "Cooperative caching for efficient data access in disruption tolerant networks," *IEEE Transactions on Mobile Computing*, vol. 13, no. 3, pp. 611–625, 2014.

[62] D. Liu, B. Chen, C. Yang, and A. F. Molisch, "Caching at the wireless edge: design aspects, challenges, and future directions," *IEEE Communications Magazine*, vol. 54, no. 9, pp. 22–28, 2016.

[63] J. Rao, H. Feng, C. Yang, Z. Chen, and B. Xia, "Optimal caching placement for D2D assisted wireless caching networks," in *Proceedings of the 2016 IEEE International Conference on Communications (ICC)*, pp. 1–6, Jiangsu, China, May 2016.

[64] G. Zheng, H. A. Suraweera, and I. Krikidis, "Optimization of hybrid cache placement for collaborative relaying," *IEEE Communications Letters*, vol. 21, no. 2, pp. 442–445, 2017.

[65] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[66] M. Akon, M. T. Islam, X. Shen, and A. Singh, "OUR: optimal update-based replacement policy for cache in wireless data access networks with optimal effective hits and bandwidth requirements," *Wireless Communications and Mobile Computing*, vol. 13, no. 15, pp. 1337–1352, 2013.

[67] S. Drakatos, N. Pissinou, K. Makki, and C. Douligeris, "A future location-aware replacement policy for the cache management at the mobile terminal," *Wireless Communications and Mobile Computing*, vol. 9, no. 5, pp. 607–629, 2009.

[68] *Slippy Map Tilenames*, http://wiki.openstreetmap.org/wiki/Slippy map tilenames.

[69] *Mapnik: An Open Source Toolkit for Developing Mapping Applications*, http://www.mapnik.org.

[70] G. Farkas, "Applicability of open-source web mapping libraries for building massive web GIS clients," *Journal of Geographical Systems*, vol. 19, no. 3, pp. 273–295, 2017.

[71] *Text Compression in R: brotli, gzip, xz and bz2*, https://cran.r-project.org/web/packages/brotli/vignettes/benchmarks.html.

[72] *Compression Benchmarks: brotli, gzip, xz, bz2*, https://www.opencpu.org/posts/brotli-benchmarks/.

[73] *Squash Compression Benchmark*, https://quixdb.github.io/squash-benchmark/.

[74] S. Steiniger and A. J. Hunter, "The 2012 free and open source GIS software map—a guide to facilitate research, development, and adoption," *Computers, Environment and Urban Systems*, vol. 39, pp. 136–150, 2013.

[75] *PostgreSQL: The World's Most Advanced Open Source Database*, https://www.postgresql.org/.

[76] *PostGIS–Spatial and Geographic objects for PostgreSQL*, http://postgis.net/.

[77] R. Friedman, A. Kogan, and Y. Krivolapov, "On power and throughput tradeoffs of WiFi and Bluetooth in smartphones," *IEEE Transactions on Mobile Computing*, vol. 12, no. 7, pp. 1363–1376, 2013.

[78] *Anyproxy*, https://github.com/alibaba/anyproxy.

[79] *Mapzen Vector Tile Service*, https://mapzen.com/projects/vector-tiles/.

[80] *Mapbox Vector Tile Specification*, https://www.mapbox.com/vector-tiles/specification/.