Handling Timestamps in OpenStreetMap

Roland M. Olbricht

September 1, 2018

There have been considerations whether timestamps should be blurred or removed, either immediately or gradually after some time.

A likely outcome of putting timestamps behind the login is:

- Rogue actors are not deterred.
- Institutional users with legal departments remain unaffected.
- Technically less capable users are locked out of crucial tools.

As this result may look surprising, I will explain how I got to this conclusion in the rest of this report.

In detail, this report is about barring timestamps behind the login requirement. Obviously, the consequences can only get more harshly if timestamps are hidden even from the logged in users. Therefore, that option is not pursued.

To keep the judgement as simple and clear as possible, this report is also not about redacting timestamps from old planets. Timestamps of changesets are not of concern because changeset information will get anyway behind the login.

To keep the report relevant, other potential sources of the information in the timestamps are taken into consideration. It were of very little use to bar timestamps if the conveyed information is anyway reconstructible from elsewhere.

1 What breaks without timestamps?

There are several different tasks that depend on having timestamps available: On the one hand theses are certain forms of quality assurance based on fixing a reviewed version. On the other hand these are workflows that rely on getting the newest edits in one way or another within minutes. In addition, timestamps are crucial when an old state of the database shall get on display, regardless whether for now historic facts, to relate project achievements to points in time, or to disentangle bugs at an unknown point in the toolchain. Or even much simpler, to offer a meaningful way of undo. To understand the precise nature of the dependency, we need to walk through the technical details.

Nodes, ways, and relations are stored in two different settings in OpenStreetMap: The current data represents the latest data available. The museum data represents all nodes, ways, and relations that at a point in the past had existed. (I do not use "historic data" to avoid confusion with [3].) The basic idea behind this is the same as with version control systems: There is still all the data that is needed to get back to the state the database has been in at any point in the past required.

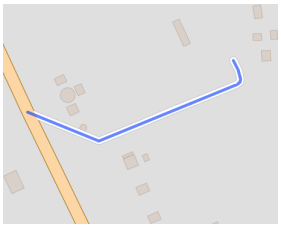
If you are not familiar with a version control system: please think of an undo functionality over unlimited time and an unlimited number of steps. The complexity comes from the fact that a global undo is not good enough: We have billions of interdepending objects, and for each undo operation we only want to touch as few objects as possible.

In the OpenStreetMap data model, even the quite basic operation of getting the geometry of a way in the past requires knowledge of timestamps of multiple versions of the way and of multiple versions of each node the way refers to.

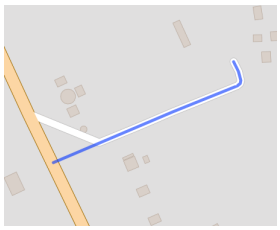
As an example, we examine the geometry of way 620642231:

```
<way id="620642231" version="1" timestamp="2018-08-28T14:44:41Z">
  <nd ref="5863607477"/>
  <nd ref="5863607476"/>
  <nd ref="5863607475"/>
  <nd ref="5863607474"/>
  <nd ref="5863607473"/>
  <nd ref="5863607468"/>
  <nd ref="5863607472"/>
  <nd ref="5863607471"/>
  <nd ref="5863607470"/>
  <nd ref="5863607469"/>
  <tag k="highway" v="residential"/>
</way>
```

The way has now, at 2018-08-29T19:20:00Z, this shape:



But at its creation it had a different shape:



Please note that both geometries come from version 1 of the way. There is no indication that the way has changed at all and even less when. It is only with the help of the timestamps of the nodes that we can reconstruct how many geometries for this way have existed, which shapes they had, and which geometry had applied to the version created by the user announced as creator.

```
<node id="5863607468" lat="6.6399282" lon="2.5312543"
      version="1" timestamp="2018-08-28T14:44:40Z"/>
```

```
<node id="5863607469" lat="6.6401413" lon="2.5312221"
    version="1" timestamp="2018-08-28T14:44:40Z"/>
<node id="5863607470" lat="6.6400521" lon="2.5312677"
    version="1" timestamp="2018-08-28T14:44:40Z"/>
<node id="5863607471" lat="6.6399748" lon="2.5312851"
    version="1" timestamp="2018-08-28T14:44:41Z"/>
<node id="5863607472" lat="6.6399495" lon="2.5312798"
    version="1" timestamp="2018-08-28T14:44:41Z"/>
<node id="5863607473" lat="6.6398003" lon="2.5309391"
    version="1" timestamp="2018-08-28T14:44:41Z"/>
<node id="5863607474" lat="6.6396977" lon="2.5306803"
    version="1" timestamp="2018-08-28T14:44:41Z"/>
<node id="5863607475" lat="6.6395632" lon="2.5303638"
    version="1" timestamp="2018-08-28T14:44:41Z"/>
<node id="5863607476" lat="6.6394273" lon="2.5300272"
    version="1" timestamp="2018-08-28T14:44:41Z"/>
<node id="5863607477" lat="6.6392582" lon="2.5296503"
    version="1" timestamp="2018-08-28T14:44:41Z"/>
<node id="5863607477" lat="6.6392488" lon="2.5296208"
    version="2" timestamp="2018-08-28T15:30:11Z"/>
<node id="5863607477" lat="6.6396831" lon="2.5293942"
    version="3" timestamp="2018-08-29T17:15:43Z"/>
```

The latter three lines are responsible for the variation of the geometry. At the creation, the way has existed alongside version 1 of node 5863607477. Since 15:30:11, version 2 of the node has been responsible for the way. And since 17:15:43 of the next day, we have our now current state.

A tool that wants to show which highways have changed between Aug 29th midnight and Aug 30th midnight needs to know whether version 2 of node 5863607477 has started before or after midnight of Aug 29th to decide whether the way has changed in that timespan or not.

If there were a second node that changed its position then we even would not know without timestamps which geometry existed at all, because we do not know which combination of versions have existed at the same time and which have not.

This is not a corner case. In a couple of samples the number of ways that changed without upcounting their version has been between 20% and 30% of all ways changed in that period. Likewise, properties of relations as whether they form a proper area or they are contiguous depend also on the referred ways and nodes and thus on their timestamps.

One functionality that by its very design thus needs timestamps is an answer to the question what has changed in the last 24 hours (or, last hour or any other time period), possibly filtered by further criteria.

This backfires in particular on tools like *Show me the Way* [5]: That tool now demonstrates the openness and beatiful simplicity of OpenStreetMap by that you can see almost-live editing activity directly from a URL and could not do so any longer.

Similarly, the almost-instant rendering is a huge mapper incentive, because mappers are motived from the near-instant feedback. The almost-instant rendering works based on minute diffs, but it only can because these convey times-

tamps at least on minute graularity anyway.

If one needed a user login to see the (up-to date) map on osm.org or to watch Showmetheway then we would present the project as much less open than it used to be and we would for sure lose a significant number of users.

This generalizes to the other cases mentioned above: What now is based on simply going to a URL (including possibly indexed by a search engine where applicable) will then at least require a login.

One immediately repercussion is that directing users to routinely surf through the OpenStreetMap universe logged in collides with privacy considerations to avoid amassing user profile data on osm.org. In particular, this may prevent users from either using privacy protection tools or from using OpenStreetMap, regardless whether the data is actually stored or evaluated on osm.org.

Even worse, the established security frameworks are unfit for the settings of the rendering stack or for multi-site architectures like Showmetheway (see the separate report about OpenStreetMap and OAuth).

Doing quality assurance by fixing states of the data (like a "tested version" in JOSM [1]), referring to remarkable old data states, and proving that the OpenStreetMap had contained certain data in the past all rely on the possibility to access old data states via a suitable parametrized URL, regardless whether the data is selected by object version or timestamps.

There are solutions for such scenarios, but all of them involve more difficult workflows or endorse bad practices like entering osm.org user credentials on other places than osm.org (e.g. JOSM inevitably processes credentials internally and does screen scraping). It will shy away less technically savvy users because of the more complex workflow and has a chilling effect on third parties (that then have most of the work to set up authentication workflows instead of working with OpenStreetMap data).

In a similar way, finding and fixing bugs gets more complicated. The typical arcane bug means that a combination of systems does not what it is supposed to do. Adding an authetication component on three or four compnents means that the order of time to find the cause raised by an order of magnitude.

By contrast, a rogue user that wants to create a user profile can do so by opening a one-time user account, ignoring the click-through contract, and downloading and processing a planet or a planet with history file.

Also, there are sufficiently other clues in the data as shown in the next section that a user with legal department and technical skills could claim to have gained timestamps up to about the minute from legal non-login sources by a technique similar to parallel construction [4].

2 What other channels convey similar information?

I have found two different independend sources beside timestamps for about the information that can be derived from timestamps. One are the minute diffs, the other are object ids. Be aware that this does not preclude the possibility of other side channels.

The minute replication diffs [2] are a core feature of OpenStreetMap; their basic working model is as followed. OpenStreetMap is though as a database of

(for this purpose) nodes, ways, and relations. A relatively tiny fraction of these elements change per minute.

The purpose of OpenStreetMap is that you can have you own copy of all nodes, ways, and relations. But it would start to age already during the time you download the data. To avoid this, minute diffs assume that you have already obtained a copy of all nodes, ways, and relations. It then tells you only which elements have changed. Applying the change of one minute to a mirrored database takes much less than a minute. That way, you can have once per minute a database that is only a minute behind the main database.

In particular, the almost-instant rendering stack has no alternative than to use minute diffs. Getting much more coarse would disturb the instant feedback loop that so many mappers see as motivating.

The minute diffs are distinct files, and there is nothing that bars a user from storing them. But the elements in this diff file have by the very purpose of the file a timestamp from that minute. There are some limitations from the technological side of the database that make this cohesion less strict; minute timestamps can deviate from the proper minute of their diff by a couple of minutes. But most elements do match their minute. Thus the minute diffs allow to track editing activity on OpenStreetMap with minute granularity.

As we have already seen that the rogue user is fine with the login requirement, the risk assessment is about the contract-abiding institutional user. Based solely on the content of minute diffs, it is possible to prove that a certain type of feature in a certain region has been edited during a certain time. Thus, it is highly likely that the activity hours of each individual user with nontrivial habits could be traced back from this data. Similarly, harassing based on waging an edit war would also work fine from the minute diffs.

Hence, the minute diffs alone are sufficient for all kind of data misuse known so far. Having timestamps does not add any significant potential for misuse.

For the sake of completeness, the ids are investigated. Each of the object types nodes, ways, and relations have their own id counter. Ids are assigned strictly ascending in time order with no gaps. Hence, by adding a node every now and then, one can predict by linear interpolation with exactitude in the order of minutes the timestamp of any node of version 1. In other words: every edit that includes adding one or more node can be traced back by using the estimated timestamp of that node id. I have no statistics, but I am confident that the majority of edits add a node.

Similarly, but with less precision because the numbers are smaller, the creation timestamp of ways and relations can be reconstructed.

Again, knowing even only the ids and versions of the objects from a static file is already enough to predict for the majority of edits their timestamp. Not perfectly, but good enough for the above mentioned tracking activities.

3 Conclusion

Hiding timestamps behind a login requirement impedes many legit and vital uses, because meaningful authentication does not fit at all into many of the workflows. But hiding timestamps can be both bluntly circumvented with a one-time account or sophisticatedly circumvented by reconstructing timestamps from minute diffs or element ids.

References

[1] https://josm.openstreetmap.de/

[2] https://wiki.openstreetmap.org/wiki/Planet.osm/diffs

[3] https://openhistoricalmap.org

[4] https://en.wikipedia.org/wiki/Parallel_construction

[5] https://osmlab.github.io/show-me-the-way/