ABSTRACT

Making available and archiving scientific results is for the most part still considered the task of classical publishing companies, despite the fact that classical forms of publishing centered around printed narrative articles no longer seem well-suited in the digital age. In particular, there exist currently no efficient, reliable, and agreed-upon methods for publishing scientific datasets, which have become increasingly important for science. Here we propose to design scientific data publishing as a Web-based bottom-up process, without top-down control of central authorities such as publishing companies. We present a protocol and a server network to decentrally store and archive data in the form of nanopublications, an RDF-based format to represent scientific data with formal semantics. We show how this approach allows researchers to produce, publish, retrieve, address, verify, and recombine datasets and their individual nanopublications in a reliable and trustworthy manner, and we argue that this architecture could be used for the Semantic Web in general. Our evaluation of the current small network shows that this system is efficient and reliable, and we discuss how it could grow to handle the large amounts of structured data that modern science is producing and consuming.

1. INTRODUCTION

Modern science increasingly depends on datasets, which however are left out in the classical way of publishing, i.e. through narrative (printed or online) articles in journals or conference proceedings. This means that the publications that describe certain scientific findings get disconnected from the data they are based on, which can seriously impair the verifiability and reproducibility of their results. Addressing this issue raises a number of practical problems: How should one publish scientific datasets and how can one refer to them in the respective scientific publications? How can we be sure that the data will remain available in the future and how can we be sure that data we find on the Web has not been corrupted or tampered with? Moreover, how can we refer to specific entries within large datasets?

To address some of these problems, a number of scientific data repositories have appeared in recent years, such as Figshare and Dryad. Furthermore, Digital Object Identifiers (DOI) have been advocated to be used not only for articles but also for scientific data [18]. While these services certainly improve the situation of scientific data, they have nevertheless several serious drawbacks: They have centralized architectures, they give us no possibility to check whether the data have been (deliberately or accidentally) modified, and they do not support access or referencing on a more granular level than entire datasets (such as individual data entries).

The centralized nature of existing data repositories is inconsistent with the decentralized manner in which science is typically performed, and it has serious consequences with respect to reliability and trust. The organizations running these platforms might at some point go bankrupt, be acquired by investors who do not feel committed to the principles of science, or simply become unable to keep their websites up and running. The open licenses enforced by these data repositories will probably lead to the situation where each dataset is available at multiple places, but there exist no standardized (i.e. automatable) procedures of how to find these alternative locations and how to decide whether they are trustworthy or not. Even if we put aside these worst-case scenarios, such websites have typically not a perfect uptime and might be down for a few minutes or even hours every once in a while. This is certainly acceptable for most use cases involving a human user accessing the data, but it can quickly become a problem in the case of automated access that is embedded in a larger service.

Even if we have perfect trust in a given organization running a data repository, it is still possible that somebody gains access to their database and silently modifies part of the data, or that the data get corrupted during the trans-
The last 45 characters of this URI (i.e. everything after ".") is what we call the artifact code. It contains a hash value that is calculated on the RDF content it represents, such as the RDF graphs of a nanopublication. Because this hash is part of the URI, any link to such an artifact comes with the possibility to verify its content, including other trusty URI links it might contain. In this way, the "range of verifiability" extends to the entire reference tree.

Research Objects are a related proposal to establish "self-contained units of knowledge" [1, 2], and they constitute in a sense the antipode approach to nanopublications. We could call them "megapublications," because they contain much more than a typical narrative approach. Namely, resources like input and output data, workflow definitions, log files, and presentation slides. Despite this obvious contrast, Research Objects and nanopublications are in many aspects complementary rather than competing ideas. We demonstrate in this paper, however, that it is not necessary to bundle all resources of scientific studies in large packages to ensure reproducibility and trust, but we can achieve these properties with trusty URIs and a decentralized server network.

The predominant approach for making linked data available on the Web are currently SPARQL endpoints, i.e. query APIs to RDF triple stores. While off-the-shelf triple stores can nowadays handle billions of triples or more, they require a large amount of resources in the form of memory and processor time to do so. A recent study found that more than half of the publicly accessible SPARQL endpoints are available less than 95% of the time [3]. To understand the consequences, just imagine you have to program a mildly time-critical service that depends on RDF data from, say, ten different SPARQL endpoints. Assuming that each endpoint is available 95% of the time and their availabilities are independent from each other, this means at least one of them will be down during close to five months per year. The reasons for this problem are quite clear: SPARQL endpoints provide a very powerful query interface that causes heavy

3

http://example.org/r1.RASAbXdpz5DcaXCh913eI9ruBosiL5XrU3x7BBa0U70

http://search cp.un.org/dist/SADI-Simple/
Given the versatility of the nanopublication standard, it seems straightforward to represent such collections as nanopublications themselves. However, if we let a collection nanopublication contain other nanopublications, then the former would become very large for large collections and would quickly lose its property of being nano. We can solve part of that problem by applying a principle that we can call reference instead of containment: nanopublications cannot contain but only refer to other nanopublications, and trusty URIs allow us to make these reference links almost as strong as containment links. To emphasize this principle, we do not call them “collections” but “indexes.”

However, even by only containing references and not the complete nanopublications, these indexes can still become quite large. To ensure that all such index nanopublications remain nano in size, we need to put some limit on the number of references. To support sets of arbitrary size, we can allow an index to be appended by other indexes. We set 1000 nanopublication references as the (admittedly arbitrary) upper limit any single index can directly contain. A set of 100,000 nanopublications, for example, can therefore be defined by a sequence of 100 indexes, each appending the previous one and the last one standing for the entire set. In addition, to allow datasets be organized in hierarchies, we define that the references of an index can also point to sub-indexes. In this way we end up with three types of relations: an index can append another index, it can contain other indexes as sub-indexes, and it can contain nanopublications as elements. These relations defining the structure of nanopublication indexes are shown schematically in Figure 1. By requiring that all indexes have trusty URIs, we also make it impossible to establish cycles using these relations.

Below we show how this general concept of indexes can be used to define sets of new or existing nanopublications, and how such index nanopublications can be published and their nanopublications retrieved.

3.2 Nanopublication Servers

Currently, the standard way of publishing RDF data are SPARQL endpoints, on top of which specific applications can be built, as summarized by the following picture:

```plaintext
applications (analyzing/using data)
SPARQL endpoints (providing/finding/querying/analyzing data)
```

Apart from the need to move workload from the server to the client (see also Section 2), we think that intermediate layers are required to make the whole system robust and reliable. In particular, the task of providing linked data should not be coupled to the tasks of providing services for finding, querying, or analyzing the data. We propose an architecture that looks more like the following picture:

```plaintext
applications (analyzing/using data)
advanced services (querying/analyzing data)
core services (finding data entries)
nanopublication server network (providing data entries)
```
In either case, the lowest layer needs to deal with the largest amounts of data and at the same time needs to be the most stable and robust one (because everything else depends on it). This can be best achieved if we free the lowest layer from all tasks except the provision of data entries (i.e., nanopublications in our case). Below we present a concrete proposal of how to implement such a low-level nanopublication server network. Based on such an infrastructure, one can then build different kinds of services operating on a subset of the nanopublications they find in the underlying network. “Core services” could involve things like resolving backwards references (i.e., “which nanopublications refer to the given one?”) and the retrieval of the nanopublications published by a given person or containing a particular URI. Based on such core services for finding nanopublications, one could then provide “advanced services” that allow us to run queries on subsets of the data and ask for aggregated output. While the lowest layer would necessarily be accessible to everybody, some of the services on the higher level can be private or limited to a small (possibly paying) user group. We have in particular scientific data in mind, but we think that an architecture of this kind could also be used for Semantic Web content in general.

In this paper we present our implementation of such a low-level decentralized nanopublication server network with a REST API to provide and propagate nanopublications identified by trusty URIs. The nanopublication servers of such a network connect to each other to retrieve and replicate their nanopublications, and they allow users to upload new nanopublications, which are then automatically distributed through the network. Figure 2 shows a schematic depiction of this server network.

Basing the content of this network on nanopublications with trusty URIs has a number of positive consequences for its design: The first benefit is that servers can make use of the fact that nanopublications are all similar in size and always small. Secondly, servers do not have to deal with identifier management, as the nanopublications already come with trusty URIs, which are guaranteed to be unique and universal. The third and possibly most important benefit is that nanopublications with trusty URIs are immutable and verifiable. This means that servers only have to deal with adding new entries but not with updating or correcting any of them. Together, this significantly simplifies the design of the network and its synchronization protocol.

We define the following requests a server has to respond to (in the form of HTTP GET requests):

- Each server needs to return general server information containing entries such as the name and email address of its administrator and the current number of stored nanopublications.
- Given an artifact code (i.e., the final part of a trusty URI) of a known nanopublication, the server returns the given nanopublication in a format like TriG, TriX, or N-Quads (depending on content negotiation).
- A server groups nanopublications in the order they are loaded into consecutive pages of 1000 nanopublications each, and such a page can be requested by page number as a list of URIs (page 1 containing the first 1000 nanopublications; page 2 the next 1000, etc.).
- For every page (except for incomplete last pages), a gzipped package can be requested containing the respective 1000 nanopublications.
- A list of known peers can be requested in the form of the URLs of other nanopublication servers in the network.

In addition, a server can optionally support the following two actions (in the form of HTTP POST requests):

- A server may accept requests to add a given individual nanopublication to its database.
- A server may also accept requests to add the URL of a new nanopublication server to its peer list.

Administrators of such servers have the additional possibility to load nanopublications from the local file system.

The current implementation is designed to be run on “normal” Web servers alongside with other applications, with economic use of the server’s resources in terms of memory and processing time. In order to avoid overload of the server or the network connection, we restrict outgoing connections to other servers to one at a time. Of course, sufficient storage space is needed to save the nanopublications (for which we currently use MongoDB), but storage space is typically much easier and cheaper to scale up than memory or processing capacities.

### 3.3 Trusty Publishing

Let us consider two simple exemplary scenarios to illustrate and motivate the general concepts. To demonstrate the procedure and the general interface of our implementation, we show here the individual steps on the command line in a tutorial-like fashion. Of course, users should eventually be supported by graphical interfaces, but command line tools are a good starting point for developers to build such tools. To make this example completely reproducible, these are the commands to download and compile the needed code from a Bash shell:

```
$ git clone git@github.com:Nanopublication/nanopub-java.git
$ cd nanopub-java
$ mvn compile
```

[4] The source code can be found at https://github.com/tkuhn/nanopub-server
And for convenience reasons, we can add the `scripts` directory to the path variable:

```bash
$ PATH=’pwd’/scripts:$PATH
```

To publish some new data, they have to be formatted as nanopublications. We use the TriG format here and define the following RDF prefixes:

```ttl
@prefix : <http://example.org/np1#>.  
@prefix np: <http://www.nanopub.org/nschema#>.  
@prefix prov: <http://www.w3.org/ns/prov#>.  
@prefix pav: <http://purl.org/pav/>.
@prefix ex: <http://example.org/>.  
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.  
@prefix dc: <http://purl.org/dc/terms/>.
```

A nanopublication consists of three graphs plus the head graph. The latter defines the structure of the nanopublication by linking to the other graphs:

```ttl
:Head {  
: a np:Nanopublication; np:hasAssertion :assertion;  
np:hasProvenance :provenance; np:hasPublicationInfo :pubinfo. }
```

The actual claim or hypothesis of the nanopublication goes into the assertion graph:

```ttl
:assertion {  
ex:mosquito ex:transmits ex:malaria.  
}
```

The provenance and publication info graph provide meta-information about the assertion and the entire nanopublication, respectively:

```ttl
:provenance {  
:assertion prov:wasDerivedFrom ex:mypublication.  
}  
:pubinfo {  
pav:createdBy <http://orcid.org/0000-0002-1267-0234>.  
:dc:created “2014-07-09T13:54:11+01:00”^^xsd:dateTime.  
}
```

The lines above constitute a very simple but complete nanopublication. To make this example a bit more interesting, let us define two more nanopublications that have different assertions but are otherwise identical:

```ttl
@prefix : <http://example.org/np2#>.  
@prefix np: <http://www.nanopub.org/nschema#>.  
@prefix prov: <http://www.w3.org/ns/prov#>.  
@prefix ex: <http://example.org/>.  
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.  
@prefix dc: <http://purl.org/dc/terms/>.
```

This gives us the file `trustynanopub.trig`, which contains transformed versions of the three nanopublications that now have trusty URIs as identifiers, as shown by the output lines above. Looking into the file we can verify that nothing has changed with respect to the content, and now we are ready to publish them:

```bash
$ PublishNanopub.sh trusty.nanopubs.trig  
3 nanopubs published at http://np.inn.ac/
```

For each of these nanopublications, we can check their publication status with the following command (referring to the nanopublication by its URI or just its artifact code):

```bash
$ NanopubStatus.sh -r RAVbDRdC4Bja5XAIdIqWCq1I-gc7IxrD5xLrXhlr-J6c  
{  
Index URI: http://np.inn.ac/RAvbDRdC4Bja5XAIdIqWCq1I-gc7IxrD5xLrXhlr-J6c  
3 nanopubs published at http://np.inn.ac/
}
```

As another exemplary scenario, let us imagine a researcher in the biomedical domain who is interested in the protein CDKN2A and who has derived some conclusion based on the data found in existing nanopublications. Specifically, let us suppose this researcher analyzed the five nanopublications found on 3 nanopub servers.

```bash
$ GetNanopub.sh -c RAVbDRdC4Bja5XAIdIqWCq1I-gc7IxrD5xLrXhlr-J6c  
{  
Index URI: http://np.inn.ac/RAvbDRdC4Bja5XAIdIqWCq1I-gc7IxrD5xLrXhlr-J6c  
3 nanopubs published at http://np.inn.ac/
}
```

This command downloads the nanopublications of the index just created and published.

As another exemplary scenario, let us imagine a researcher in the biomedical domain who is interested in the protein CDKN2A and who has derived some conclusion based on the data found in existing nanopublications. Specifically, let us suppose this researcher analyzed the five nanopublications specified by the following artifact codes (they can be viewed online by appending the artifact code to the URI http://np.inn.ac/):
The respective nanopublications via the server an option for cases where we have hundreds or thousands as well refer to them individually, but this is obviously not following:

1 nanopub published at http://np.inn.ac/

$ PublishNanopub.sh index.cdkn2a-nanopubs.trig

The generated index is stored in the file index.cdkn2a-nanopubs.trig, and our exemplary researcher can now publish this index to let others know about it:

$ PublishNanopub.sh index.cdkn2a-nanopubs.trig

There is no need to publish the five nanopublications this index is referring to, because they are already public (this is how we got them in the first place). The index URI can now be used to refer to this new collection of existing nanopublications in an unambiguous and reliable manner. This URI can be included in the scientific publication that explains the new finding, for example with a reference like the URI can be included in the scientific publication that explains the new finding, for example with a reference like the following:

[1] Data about CDKN2A from BEL2nanopub & neXtProt. Nanopublication index http://np.inn.ac/RA1FqcPoyVCTez-LJApJ2c_wvJHo7XZ0-xx8pJBzQZo0, 6 November 2014.

In this case with just five nanopublications, one might as well refer to them individually, but this is obviously not an option for cases where we have hundreds or thousands of them. The given web link allows everybody to retrieve the respective nanopublications via the server np.inn.ac. The URL will not resolve should the server be temporarily or permanently down, but because it is a trusty URI we can retrieve the nanopublications from any other server of the network following a well-defined protocol (basically just extracting the artifact code, i.e. the last 45 characters, and appending it to the URL of another nanopublication server). This reference is therefore more reliable and more robust than links to other types of data repositories. In fact we refer to the datasets we use in this publication for evaluation purposes in exactly this way [23, 25, 26, 27, 24].

The new finding that was deduced from the given five nanopublications should ideally, of course, also be published as a nanopublication, with a reference to the given index URI in the provenance part:

@prefix : <http://example.org/myfinding#>.
@prefix nps: <http://np.inn.ac/>.
@prefix uniprot: <http://purl.uniprot.org/uniprot/>.

:pubinfo {
  :dc:created "2014-11-06T15:05:43+01:00"^^xsd:dateTime.
  :pav:createdBy "http://orcid.org/0000-0002-1267-0234".
  :dc:created "2014-11-06T15:05:43+01:00"^^xsd:dateTime.
}

We can again transform it to a trusty nanopublication (the resulting trusty URI is http://example.org/myfinding#RA5k4zVEFa-3N1JTQ4O0X_YH9ekJ7EmDbgj9R0Inlk1j4), and then publish it as above.

Some of the features of the presented command-line interface are made available through a web interface for dealing with nanopublications that is shown in Figure 3. The supported features include the generation of trusty URIs, as well as the publication and retrieval of nanopublications. The interface allows us to retrieve, for example, the nanopublication we just generated and published above, even though we used an example.org URI, which is not directly resolvable. Unless it is just about toy examples, we should of course try to use resolvable URIs, but with our decentralized network we can retrieve the data even if the original link is no longer functioning or temporarily broken.

### 4. Evaluation

As a first evaluation of our approach, we wanted to find out whether a small server network run on normal Web servers without dedicated infrastructure is able to handle the amount of nanopublications we can expect to become publicly available in the next one or two years. We think that this is an important condition to keep the network alive in the initial stage, whereas one can later — should the ideas of nanopublications and such server networks catch on — gradually add dedicated servers to the network. The number of nanopublications that are publicly available reaches at the moment the range of millions, maybe even tens of millions. We might conjecture that this number will grow to hundreds of millions in the next one or two years, maybe even reaching one billion.

The current server network, which we use for this evaluation, consists of three servers in three different countries on two continents, as shown in Figure 4. One server is in Zurich (hosted by ETH), one in Ottawa (operated from Stanford University), and one in New Haven (hosted by Yale University). The question we want to answer here is whether this small network is able to handle the number of nanopublications we can expect in the near future.

Figure 4 is a screenshot of a nanopublication monitor that we have implemented.\(^5\) Such monitors regularly check the nanopublication server network and register changes (current these checks are performed once per minute). Furthermore, they test the response times and the correct operation of the servers by requesting a random nanopublication and verifying the returned data. We have currently two such monitors running, one in Zurich and one in Ottawa, which form an important part of the evaluation.

#### 4.1 Evaluation Design

Table 1 shows the existing datasets in the nanopublication format that we use to evaluate our approach. This includes all datasets we are aware of that use trusty URIs, with a total of more than 5 million nanopublications and close to 200 million RDF triples, including nanopublication indexes that we generated for each dataset.

Each of these datasets is assigned to one of the three servers. In the beginning of the evaluation, the servers start loading these nanopublications from the local file system. The first nanopublications start spreading to the other servers. In the beginning of the evaluation, the servers start loading these nanopublications from the local file system. The first nanopublications start spreading to the other servers.

\(^5\) See https://github.com/tkuhn/bel2nanopub and http://nextprot2rdf.sourceforge.net, respectively, and Table 1

\(^6\) https://github.com/tkuhn/nanopub-monitor
Figure 3: The web interface of the nanopublication validator can load nanopublications by their trusty URI (or just their artifact code) from the nanopublication server network. It also allows users to directly publish uploaded nanopublications.

![Validator (and more) for Nanopublications](image)

Table 1: Existing datasets in the nanopublication format that were used for the evaluation.

| Dataset                  | Number of Nanopubs | Number of Triples | Initial Location for Evaluation |
|--------------------------|--------------------|-------------------|--------------------------------|
| GeneRIF/AIDA [23]        | 157                | 156,183           | New Haven                      |
| OpenBEL 1.0 [25]         | 53                 | 50,760            | New Haven                      |
| OpenBEL 20131211 [26]    | 76                 | 74,249            | New Haven                      |
| DisGeNET v2.1.0.0 [27]   | 941                | 951,325           | Zurich                         |
| neXtProt (preliminary) [24] | 4,026            | 4,078,318         | Ottawa                         |
| Total                    | 5,253              | 5,314,236         |                                |

ers, while others are still being loaded from the file system. We therefore test the reliability and capacity of the network under a heavy-load scenario, with constant streams of new nanopublications coming from different servers.

We use the two nanopublication monitors to evaluate the responsiveness of the network. To analyze the specific traffic patterns and the work load on the different servers, we furthermore use their detailed log files.

4.2 Evaluation Results

The evaluation lasted 13 hours and 21 minutes, at which point all nanopublications were replicated on all three servers, and therefore the nanopublication traffic came to an end. Figure 5 shows the type and intensity of the data flow (i.e. the transfer of nanopublications) between the three servers over the time of the evaluation. We see that the replication of nanopublications from other servers kicks in before all local nanopublications are loaded, and the two processes then continue in parallel, interfering with each other only because they have to access the same database. In contrast, the loading of nanopublications from the other two servers is never done in parallel (due to our design decision for modest use of resources) but switches from one to the other and back.

Figure 6 shows the rate at which the nanopublications were loaded at their first, second, and third server, respectively. Over the complete duration of the evaluation, the network was able to handle an average of about 400,000 new nanopublications per hour, which corresponds to more than 100 new nanopublications per second. This includes the time needed for loading each nanopublication once from the local file system (at the first server), transferring it through the network two times (to the other two servers), and for verifying it three times (once when loaded and twice when received by the other two servers). The overall capacity of the current server network is therefore at least 9.4 million new nanopublications per day or 3.4 billion per year (given that we make sure that there is sufficient storage space on each of the servers).

Figure 7 shows the response times of the three servers as measured by the two nanopublication monitors in Zurich (top) and Ottawa (bottom). The time coverage starts from the start of the evaluation until 24h later, therefore covering the entire evaluation plus an additional 10 hours and 39 minutes after its end. We see that the observed latency is mostly due to the geographical distance between the servers and the monitors. The response time was always less
than 0.25s when the server was on the same continent as the measuring monitor. In 99.86% of all cases (including those across continents) the response time was below 0.5s, and it was always below 1.1s. Not a single one of the 8636 individual HTTP requests timed out, led to an error, or received a nanopublication that could not be successfully verified. We see that the heavy load we put onto the network did not have a big impact on the response times. Except for a handful of requests for which we see spikes in the curves, one barely notices the difference between the heavy-load and zero-load scenarios.

5. DISCUSSION

We have presented here a very low-level infrastructure for data sharing, which is just one piece of the whole puzzle. Several other pieces are still missing to make the system fully functional. Their development is ongoing or future work, and they include at least the following four aspects: (1) We need a standardized method to digitally sign nanopublications (to be performed before we calculate the trusty URI, such that the latter includes the signature). (2) We need to develop “core services” (see Section 3.2) on top of the server network to allow people to find nanopublications, as well as “advanced services” to query and analyze the content of nanopublications, and exemplary applications that make use of them. (3) We need to establish standards and best practices of how to use existing ontologies (and possibly define new ones) to describe types and relations of nanopublications (e.g. referring to earlier versions, marking nanopublications as retracted, and reviewing of nanopublications). And finally, (4) we need to find a way of referring to raw data that is too big to be converted to nanopublications, possibly via BitTorrent as has been suggested elsewhere \[15\].

Apart from that, we also have to scale up the current small network. As our protocol only allows for simple key-based lookup, the time complexity for all types of requests is sublinear and therefore scales up well. The main limiting factor is disk space, which is relatively cheap and easy to add. Still, the servers will have to specialize in order to handle really large amounts of data, which can be done in a number of ways: Servers can restrict themselves to particular types of nanopublications, e.g. to specific topics or authors, and communicate this to the network (e.g. expressed in OWL); inspired by the Bitcoin system, certain servers
could only accept nanopublications whose hash starts with a given number of zero bits, which makes it costly to publish; and some servers could be specialized to new nanopublications, providing fast access but only for a restricted time, while others could take care of archiving old nanopublications, possibly on tape and with considerable delays between request and delivery.

Lastly, there could also emerge interesting synergies with novel approaches to internet networking, such as Content-Centric Networking (CCN) [10], with which — consistent with our proposal — requests are based on content rather than hosts.

6. CONCLUSION

We argue that data publishing can and should be done in a decentralized bottom-up manner, and we demonstrate this approach based on nanopublications and trusty URIs. We introduce a protocol and a small server network that is able to handle the amounts of nanopublications we can expect to become publicly available within the next few years. We show that this network is reliable and robust even under heavy load, we discuss how it could grow to handle the large amounts of structured data that we can expect to become available as nanopublications in the medium and long run.

We believe that this network can serve as a solid basis for semantic publishing, and possibly also for the Semantic Web in general. It could contribute to improve the availability and reproducibility of scientific results and put a reliable and trustworthy layer underneath the Semantic Web.

7. REFERENCES

[1] S. Bechhofer, D. De Roure, M. Gamble, C. Goble, and I. Buchan. Research objects: Towards exchange and reuse of digital knowledge. The Future of the Web for Collaborative Science, 2010.

[2] K. Belhajjame, O. Corcho, D. Garijo, J. Zhao, P. Missier, D. Newman, R. Palma, S. Bechhofer, E. Garcia, J. M. G.-P. Cuesta, et al. Workflow-centric research objects: First class citizens in scholarly discourse. 2012.

[3] C. Buil-Aranda, A. Hogan, J. Umbrich, and P.-Y. Vandenbussche. SPARQL web-querying infrastructure: Ready for action? In The Semantic Web—ISWC 2013, pages 277–293. Springer, 2013.

[4] J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In WWW ’05, pages 613–622. ACM, 2005.

[5] C. Chichester, P. Gaudet, O. Karch, P. Groth, L. Lane, A. Bairoch, B. Mons, and A. Loizou. Querying neXtProt nanopublications and their value for insights on sequence variants and tissue expression. Web Semantics: Science, Services and Agents on the World Wide Web, 2014.

[6] C. Chichester, O. Karch, P. Gaudet, L. Lane, B. Mons, and A. Bairoch. Converting neXtProt into linked data and nanopublications. Semantic Web, 2014, to appear.

[7] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In Designing Privacy Enhancing Technologies, pages 46–66. Springer, 2001.

[8] K. Fu, M. F. Kaashoek, and D. Mazières. Fast and secure distributed read-only file system. ACM Transactions on Computer Systems, 20(1):1–24, 2002.

[9] P. Groth, A. Gibson, and J. Velterop. The anatomy of a nano-publication. Information Services and Use, 30(1):51–56, 2010.

[10] V. Jacobson, D. K. Smetters, J. D. Thornton, M. Plass, N. Briggs, and R. Braynard. Networking named content. Commun. ACM, 55(1):117–124, Jan. 2012.

[11] T. Kuhn. A survey and classification of controlled natural languages. Computational Linguistics, 40(1):121–170, March 2014.

[12] T. Kuhn, P. E. Barbano, M. L. Nagy, and M. Krauthammer. Broadening the scope of nanopublications. In Proceedings of the 10th Extended
Figure 7: This diagram shows the server response times as recorded during and after the evaluation, which ended at 13 hours and 21 minutes, as indicated by the black vertical line.