Kuhn T, Chichester C, Krauthammer M, Queralt-Rosinach N, Verborgh R, Giannakopoulos G, Ngonga Ngomo A, Viglianti R, Dumontier M. 2016. Decentralized provenance-aware publishing with nanopublications. PeerJ Computer Science 2:e78 https://doi.org/10.7717/peerj-cs.78

Decentralized provenance-aware publishing with nanopublications

Tobias Kuhn, Christine Chichester, Michael Krauthammer, Núria Queralt-Rosinach, Ruben Verborgh, George Giannakopoulos, Axel-Cyrille Ngonga Ngomo, Raffaele Viglianti, Michel Dumontier

Publication and archival of scientific results is still commonly considered the responsability of classical publishing companies. Classical forms of publishing, however, which center 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. In this article, 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. Based on a novel combination of existing concepts and technologies, we present a server network to decentrally store and archive data in the form of nanopublications, an RDF-based format to represent scientific data. We show how this approach allows researchers to publish, retrieve, verify, and recombine datasets of nanopublications in a reliable and trustworthy manner, and we argue that this architecture could be used as a low-level data publication layer to serve the Semantic Web in general. Our evaluation of the current network shows that this system is efficient and reliable.

PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016

Decentralized Provenance-Aware Publishing with Nanopublications

Tobias Kuhn[1], Christine Chichester[2], Michael Krauthammer[3], Núria Queralt-Rosinach[4], Ruben Verborgh[5], George Giannakopoulos[6], Axel-Cyrille Ngonga Ngomo[7], Raffaele Viglianti[8], and Michel Dumontier[9]

[1]Department of Computer Science, VU University Amsterdam, De Boelelaan 1081, 1081 HV Amsterdam, Netherlands, t.kuhn@vu.nl *(corresponding author)*
[2]Nestle Institute of Health Sciences, Lausanne, Switzerland
[3]Yale University School of Medicine, New Haven, CT, USA
[4]Universitat Pompeu Fabra, Barcelona, Spain
[5]Multimedia Lab, Ghent University – iMinds, Ghent, Belgium
[6]NCSR Demokritos, Athens, Greece
[7]AKSW Research Group, University of Leipzig, Germany
[8]University of Maryland, College Park, MD, USA
[9]Stanford Center for Biomedical Informatics Research, Stanford University, CA, USA

ABSTRACT

Publication and archival of scientific results is still commonly considered the responsability of classical publishing companies. Classical forms of publishing, however, which center 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. In this article, 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. Based on a novel combination of existing concepts and technologies, we present a server network to decentrally store and archive data in the form of nanopublications, an RDF-based format to represent scientific data. We show how this approach allows researchers to publish, retrieve, verify, and recombine datasets of nanopublications in a reliable and trustworthy manner, and we argue that this architecture could be used as a low-level data publication layer to serve the Semantic Web in general. Our evaluation of the current network shows that this system is efficient and reliable.

Keywords: Data publishing, Semantic Web, Linked Data, provenance, nanopublications

1 INTRODUCTION

Modern science increasingly depends on *datasets*, which are however 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 describing scientific findings become 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 have not been corrupted or tampered with? Moreover, how can we refer to specific entries or subsets from large datasets, for instance, to support a specific argument or hypothesis?

To address some of these problems, a number of scientific data repositories have appeared, such as Figshare and Dryad.[1] Furthermore, Digital Object Identifiers (DOI) have been advocated to be used not only for articles but also for scientific data (Paskin, 2005). While these approaches certainly improve the situation of scientific data, in particular when combined with Semantic Web techniques, they have nevertheless a number of drawbacks: They have *centralized* architectures, they give us no possibility to

[1]http://figshare.com, http://datadryad.org

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). We argue that the centralized nature of existing data repositories is inconsistent with the decentralized manner in which science is typically performed, and that 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 for other reasons become unable to keep their websites up and running. Even though the open licenses enforced by these data repositories will probably ensure that the datasets remain available at different places, there exist no standardized (i.e. automatable) procedures to find these alternative locations and to decide whether they are trustworthy or not.

Even if we put aside these worst-case scenarios, 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 data from these websites, but it can quickly become a problem in the case of automated access embedded in a larger service. Furthermore, it is possible that somebody gains access to the repository's database and silently modifies part of the data, or that the data get corrupted during the transfer from the server to the client. We can therefore never perfectly trust any data we get, which significantly complicates the work of scientists and impedes the potential of fully automatic analyses. Lastly, existing forms of data publishing have for the most part only one level at which data is addressed and accessed: the level of entire datasets (sometimes split into a small number of tables). It is in these cases not possible to refer to individual data entries or subsets in a way that is standardized and retains the relevant metadata and provenance information. To illustrate this problem, let us assume that we conduct an analysis using, say, 1000 individual data entries from each of three very large datasets (containing, say, millions of data entries each). How can we now refer to exactly these 3000 entries to justify whatever conclusion we draw from them? The best thing we can currently do is to republish these 3000 data entries as a new dataset and to refer to the large datasets as their origin. Apart from the practical disadvantages of being forced to republish data just to refer to subsets of larger datasets, other scientists need to either (blindly) trust us or go through the tedious process of semi-automatically verifying that each of these entries indeed appears in one of the large datasets. Instead of republishing the data, we could also try to describe the used subsets, e.g. in the form of SPARQL queries in the case of RDF data, but this doesn't make it less tedious, keeping in mind that older versions of datasets are typically not provided by public APIs such as SPARQL endpoints.

Below, we present an approach to tackle these problems, which builds upon existing Semantic Web technologies, in particular RDF and nanopublications, and adheres to accepted Web principles, such as decentralization and REST APIs. Specifically, our research question is: Can we create a decentralized, reliable, trustworthy, and scalable system for publishing, retrieving, and archiving datasets in the form of sets of nanopublications based on existing Web standards and infrastructure?

This article is an extended and revised version of a previous conference paper (Kuhn et al., 2015).

2 BACKGROUND

Nanopublications (Groth et al., 2010) are a relatively recent proposal for improving the efficiency of finding, connecting, and curating scientific findings in a manner that takes attribution, quality levels, and provenance into account. While narrative articles would still have their place in the academic landscape, small formal data snippets in the form of nanopublications should take their central position in scholarly communication (Mons et al., 2011). Most importantly, nanopublications can be automatically interpreted and aggregated and they allow for fine-grained citation metrics on the level of individual claims. A nanopublication is defined as a small data container consisting of three parts: an assertion part containing the main content in the form of an atomic piece of formally represented data (e.g. an observed effect of a drug on a disease); a provenance part that describes how this piece of data came about (e.g. how it was measured); and a publication info part that gives meta-information about the nanopublication as a whole (e.g. when it was created). The representation of a nanopublication with its three parts is based on the RDF language with named graphs (Carroll et al., 2005). In other words, the nanopublication approach boils down to the ideas of subdividing scientific results into atomic assertions, representing these assertions in RDF, attaching provenance information in RDF on the level of individual assertions, and treating each of these tiny entities as an individual publication. Nanopublications have been applied to a number of domains, so far mostly from the life sciences including pharmacology (Williams et al., 2012; Banda et al., 2015), genomics (Patrinos et al., 2012), and proteomics (Chichester et al., 2015). An increasing number of

PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016

datasets formatted as nanopublications are openly available, including neXtProt (Chichester et al., 2014) and DisGeNET (Queralt-Rosinach et al., 2015), and the nanopublication concept has been combined with and integrated into existing frameworks for data discovery and integration, such as CKAN (McCusker et al., 2013).

Interestingly, the concept of nanopublications has also been taken up in the humanities, namely in philosophy,[2] musicology (Freedman, 2014), and history/archaeology (Golden and Shaw, 2016). A humanities dataset of facts is arguably more interpretive than a scientific dataset; relying, as it does, on the scholarly interpretation of primary sources. Because of this condition, "facts" in humanities datasets (such as prosopographies) have often been called "factoids" (Bradley, 2005), as they have to account for a degree of uncertainty. Nanopublications, with their support for granular context and provenance descriptions, offer a novel paradigm for publishing such factoids, by providing methods for representing metadata about responsibilities and by enabling discussions and revisions beyond any single humanities project.

Research Objects are a related approach to establish "self-contained units of knowledge" (Belhajjame et al., 2012), and they constitute in a sense the antipode approach to nanopublications. We could call them *mega*-publications, as they contain much more than a typical narrative publication, namely resources like input and output data, workflow definitions, log files, and presentation slides. We demonstrate in this paper, however, that bundling all resources of scientific studies in large packages is not a necessity to ensure the availability of the involved resources and their robust interlinking, but we can achieve that also with cryptographic identifiers and a decentralized architecture.

SPARQL is an important and popular technology to access and publish Linked Data, and it is both a language to query RDF datasets (Harris and Seaborne, 2013) and a protocol to execute such queries on a remote server over HTTP (Feigenbaum et al., 2013). Servers that provide the SPARQL protocol, referred to as "SPARQL endpoints", are a technique for making Linked Data available on the Web in a flexible manner. While off-the-shelf triple stores can nowadays handle billions of triples or more, they potentially require a significant amount of resources in the form of memory and processor time to execute queries, at least if the full expressive power of the SPARQL language is supported. A recent study found that more than half of the publicly accessible SPARQL endpoints are available less than 95% of the time (Buil-Aranda et al., 2013), posing a major problem to services depending on them, in particular to those that depend on several endpoints at the same time. To understand the consequences, imagine one has 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 load in terms of memory and computing power on the side of the server. Clients can request answers to very specific and complex queries they can freely define, all without paying a cent for the service. This contrasts with almost all other HTTP interfaces, in which the server imposes (in comparison to SPARQL) a highly limited interface, where the computational costs per request are minimal.

To solve these and other problems, more light-weight interfaces were suggested, such as the read-write Linked Data Platform interface (Speicher et al., 2015), the Triple Pattern Fragments interface (Verborgh et al., 2014), as well as infrastructures to implement them, such as CumulusRDF (Ladwig and Harth, 2011). These interfaces deliberately allow less expressive requests, such that the maximal cost of each individual request can be bounded more strongly. More complex queries then need to be evaluated by clients, which decompose them in simpler subqueries that the interface supports (Verborgh et al., 2014). While this constitutes a scalability improvement (at the cost of, for instance, slower queries), it does not necessarily lead to perfect uptimes, as servers can be down for other reasons than excessive workload. We propose here to go one step further by relying on a *decentralized* network and by supporting only identifier-based lookup of nanopublications. Such limited interfaces normally have the drawback that traversal-based querying does not allow for the efficient and complete evaluation of certain types of queries (Hartig, 2013), but this is not a problem with the multi-layer architecture we propose below, because querying is only performed at a higher level where these limitations do not apply.

A well-known solution to the problem of individual servers being unreliable is the application of a decentralized architecture where the data is replicated on multiple servers. A number of such approaches

[2] http://emto-nanopub.referata.com/wiki/EMTO_Nanopub

PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016

related to data publishing have been proposed, for example in the form of distributed file systems based on cryptographic methods for data that are public (Fu et al., 2002) or private (Clarke et al., 2001). In contrast to the design principles of the Semantic Web, these approaches implement their own internet protocols and follow the hierarchical organization of file systems. Other approaches build upon the existing BitTorrent protocol and apply it to data publishing (Markman and Zavras, 2014; Cohen and Lo, 2014), and there is interesting work on repurposing the proof-of-work tasks of Bitcoin for data preservation (Miller et al., 2014). There exist furthermore a number of approaches to applying peer-to-peer networks for RDF data (Filali et al., 2011), but they do not allow for the kind of permanent and provenance-aware publishing that we propose below. Moreover, only for the centralized and closed-world setting of database systems, approaches exist that allow for robust and granular references to subsets of dynamic datasets (Proell and Rauber, 2014).

The approach that we present below is based on previous work, in which we proposed *trusty URIs* to make nanopublications and their entire reference trees verifiable and immutable by the use of cryptographic hash values (Kuhn and Dumontier, 2014, 2015). This is an example of such a trusty URI:

```
http://example.org/r1.RA5AbXdpz5DcaYXCh9l3eI9ruBosiL5XDU3rxBbBaUO70
```

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.

Furthermore, we argued in previous work that the assertion of a nanopublication need not be fully formalized, but we can allow for informal or underspecified assertions (Kuhn et al., 2013), to deal with the fact that the creation of accurate semantic representations can be too challenging or too time-consuming for many scenarios and types of users. This is particularly the case for domains that lack ontologies and standardized terminologies with sufficient coverage. These structured but informal statements are supposed to provide a middle ground for the situations where fully formal statements are not feasible. We proposed a controlled natural language (Kuhn, 2014) for these informal statements, which we called AIDA (standing for the introduced restriction on English sentences to be atomic, independent, declarative, and absolute), and we had shown before that controlled natural language can also serve in the fully formalized case as a user-friendly syntax for representing scientific facts (Kuhn et al., 2006). We also sketched how "science bots" could autonomously produce and publish nanopublications, and how algorithms could thereby be tightly linked to their generated data (Kuhn, 2015b), which requires the existence of a reliable and trustworthy publishing system, such as the one we present here.

3 APPROACH

Our approach builds upon the existing concept of nanopublications and our previously introduced method of trusty URIs. It is a proposal of a reliable implementation of accepted Semantic Web principles, in particular of what has become known as the *follow-your-nose* principle: Looking up a URI should return relevant data and links to other URIs, which allows one (i.e. humans as well as machines) to discover things by navigating through this data space (Berners-Lee, 2006). We argue that approaches following this principle can only be reliable and efficient if we have some sort of guarantee that the resolution of any single identifier will succeed within a short time frame in one way or another, and that the processing of the received representation will only take up a small amount of time and resources. This requires that (1) RDF representations are made available on several distributed servers, so the chance that they all happen to be inaccessible at the same time is negligible, and that (2) these representations are reasonably small, so that downloading them is a matter of fractions of a second, and so that one has to process only a reasonable amount of data to decide which links to follow. We address the first requirement by proposing a distributed server network and the second one by building upon the concept of nanopublications. Below we explain the general architecture, the functioning and the interaction of the nanopublication servers, and the concept of nanopublication indexes.

3.1 Architecture

There are currently at least three possible architectures for Semantic Web applications (and mixtures thereof), as shown in a simplified manner in Figure 1. The first option is the use of plain HTTP GET

PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016

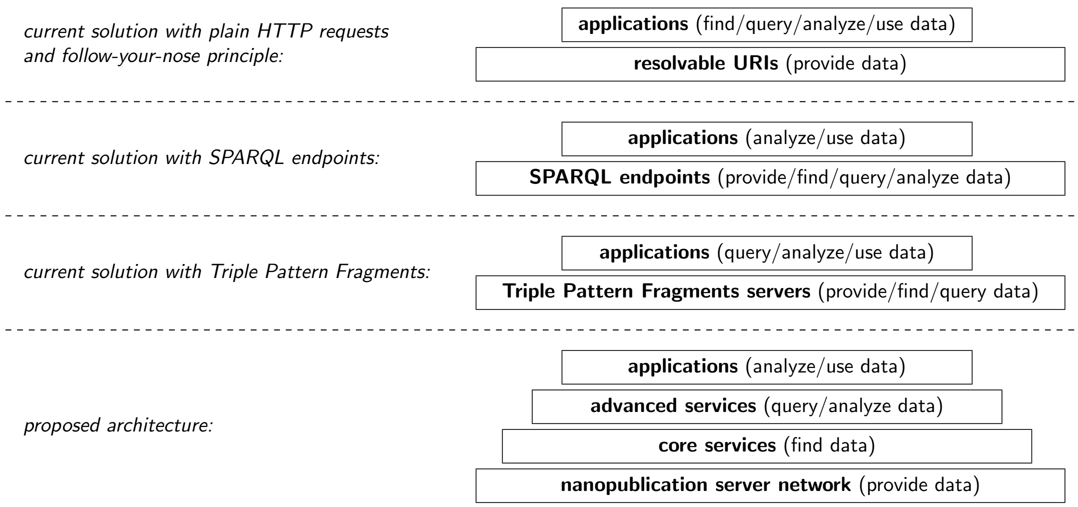


Figure 1. Illustration of current architectures of Semantic Web applications and our proposed approach

requests to dereference a URI. Applying the follow-your-nose principle, resolvable URIs provide the data based on which the application performs the tasks of finding relevant resources, running queries, analyzing and aggregating the results, and using them for the purpose of the application. This approach aligns very well with the principles and the architecture of the Web, but the traversal-based querying it entails comes with limitations on efficiency and completeness (Hartig, 2013). If SPARQL endpoints are used, as a second option, most of the workload is shifted from the application to the server via the expressive power of the SPARQL query language. As explained above, this puts servers at risk of being overloaded. With a third option such as Triple Pattern Fragments, servers provide only limited query features and clients perform the reminder of the query execution. This leads to reduced server costs, at the expense of longer query times.

We can observe that all these current solutions are based on two-layer architectures, and have moreover no inherent replication mechanisms. A single point of failure can cause applications to be unable to complete their tasks: A single URI that does not resolve or a single server that does not respond can break the entire process. We argue here that we need distributed and decentralized services to allow for robust and reliable applications that consume Linked Data. In principle, this can be achieved for any of these two-layer architectures by simply setting up several identical servers that mirror the same content, but there is no standardized and generally accepted way of how to communicate these mirror servers and how to decide on the client side whether a supposed mirror server is trustworthy. Even putting aside these difficulties, two-layer architectures have further conceptual limitations. The most low-level task of providing Linked Data is essential for all other tasks at higher levels, and therefore needs to be the most stable and robust one. We argue that this can be best achieved if we free this lowest layer from all tasks except the provision and archiving of data entries (nanopublications in our case) and decouple it from the tasks of providing services for finding, querying, or analyzing the data. This makes us advocate a multi-layer architecture, a possible realization of which is shown at the bottom of Figure 1.

Below we present a concrete proposal of such a low-level data provision infrastructure in the form of a 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. These higher layers can of course make use of existing techniques such as SPARQL endpoints and Triple Pattern Fragments or even classical relational databases, and they can cache large portions of the data from the layers below (as nanopublications are immutable, they are easy to cache). For example, an advanced service could allow users to query the latest versions of several drug-related datasets, by keeping a local triple store

PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016

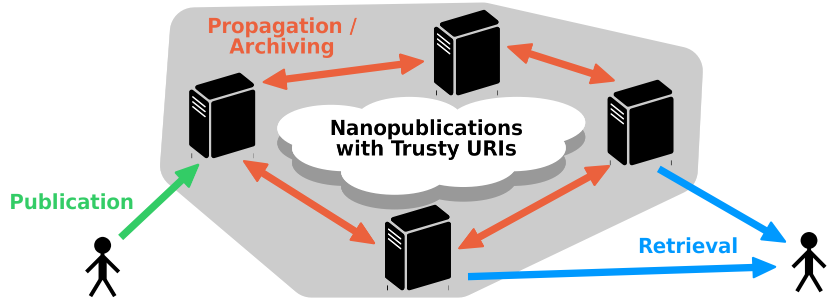


Figure 2. Schematic representation of the decentralized server architecture. Nanopublications can be uploaded to a server (or loaded from the local file system by the server administrator), and they are then propagated to the other servers of the network. They can then be retrieved from any of the servers, or from multiple servers simultaneously, even if the original server is not accessible.

and providing users with a SPARQL interface. Such a service would regularly check for new data in the server network on the given topic, and replace outdated nanopublications in its triple store with new ones. A query request to this service, however, would *not* involve an immediate query to the underlying server network, in the same way that a query to the Google search engine does *not* trigger a new crawl of the Web.

While the lowest layer would necessarily be accessible to everybody, some of the services on the higher level could 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.

3.2 Nanopublication Servers

As a concrete proposal of a low-level data provision layer, as explained above, we present here a decentralized nanopublication server network with a REST API to provide and propagate nanopublications identified by trusty URIs.[3] The nanopublication servers of such a network connect to each other to retrieve and (partly) 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 the fact that nanopublications are all similar in size and always small makes it easy to estimate how much time is needed to process an entity (such as validating its hash) and how much space to store it (e.g. as a serialized RDF string in a database). Moreover it ensures that these processing times remain mostly in the fraction-of-a-second range, guaranteeing quick responses, and that these entities are never too large to be analyzed in memory. The second benefit is that 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* them, which eliminates the hard problems of concurrency control and data integrity in distributed systems. Together, these aspects significantly simplify the design of such a network and its synchronization protocol, and make it reliable and efficient even with limited resources.

Specifically, a nanopublication server of the current network has the following components:

- A **key-value store** of its nanopublications (with the artifact code from the trusty URI as the key)
- A long list of all stored nanopublications, in the order they were loaded at the given server. We call this list the server's **journal**, and it consists of a journal identifier and the sequence of nanopublication identifiers, subdivided into pages of a fixed size. (1000 elements is the default: page 1 containing the first 1000 nanopublications; page 2 the next 1000, etc.)

[3] Source code repository: https://github.com/tkuhn/nanopub-server

PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016

- A **cache of gzipped packages** containing all nanopublications for a given journal page
- **Pattern definitions** in the form of a *URI pattern* and a *hash pattern*, which define the surface features of the nanopublications stored on the given server
- A **list of known peers**, i.e. the URLs of other nanopublication servers
- **Information about each known peer**, including the journal identifier and the total number of nanopublications at the time it was last visited

The server network can be seen as an unstructured peer-to-peer network, where each node can freely decide which other nodes to connect to and which nanopublications to replicate.

The URI pattern and the hash pattern of a server define the surface features of the nanopublications that this server cares about. We called them *surface features*, because they can be determined by only looking at the URI of a nanopublication. For example, the URI pattern '`http://rdf.disgenet.org/`' states that the given server is only interested in nanopublications whose URIs start with the given sequence of characters. Additionally, a server can declare a hash pattern like '`AA AB`' to state that it is only interested in nanopublications whose hash in the trusty URI start with one of specified character sequences (separated by blank spaces). As hashes are represented in Base64 notation, this particular hash pattern would let a server replicate about 0.05% of all nanopublications. Nanopublication servers are thereby given the opportunity to declare which subset of nanopublications they replicate, and need to connect only to those other servers whose subsets overlap. To decide on whether a nanopublication belongs to a specified subset or not, the server only has to apply string matching at two given starting points of the nanopublication URI (i.e. the first position and position 43 from the end — as the hashes of the current version of trusty URIs are 43 bytes long), which is computationally cheap.

Based on the components introduced above, the servers respond to the following request (in the form of HTTP GET):

- Each server needs to return general **server information**, including the journal identifier and the number of stored nanopublications, the server's URI pattern and hash pattern, whether the server accepts POST requests for new nanopublications or servers (see below), and informative entries such as the name and email address of the maintainer and a general description. Additionally, some server-specific limits can be specified: the maximum number of triples per nanopublication (the default is 1200), the maximum size of a nanopublication (the default is 1 MB), and the maximum number of nanopublications to be stored on the given server (unlimited by default).
- Given an artifact code (i.e. the final part of a trusty URI) of a nanopublication that is stored by the server, it returns the given **nanopublication** in a format like TriG, TriX, N-Quads, or JSON-LD (depending on content negotiation).
- A **journal page** can be requested by page number as a list of trusty URIs.
- For every journal page (except for incomplete last pages), a gzipped **package** can be requested containing the respective nanopublications.
- The **list of known peers** can be requested as a list of URLs.

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.

Server administrators have the additional possibility to load nanopublications from the local file system, which can be used to publish large amounts of nanopublications, for which individual POST requests are not feasible.

Together, the server components and their possible interactions outlined above allow for efficient decentralized distribution of published nanopublications. Specifically, current nanopublication servers

PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016

follow the following procedure. Every server s keeps its own list of known peer P_s. For each peer p on that list that has previously been visited, the server additionally keeps the number of nanopublications on that peer server n'_p and its journal identifier j'_p, as recorded during the last visit. At a regular interval, every peer server p on the list of known peers is visited by server s:

1. The latest server information is retrieved from p, which includes its list of known peers P_p, the number of stored nanopublications n_p, the journal identifier j_p, the server's URI pattern U_p, and its hash pattern H_p.

2. All entries in P_p that are not yet on the visiting server's own list of known peers P_s are added to P_s.

3. If the visiting server's URL is not in P_p, the visiting server s makes itself known to server p with a POST request (if this is supported by p).

4. If the subset defined by the server's own URI/hash patterns U_s and H_s does not overlap with the subset defined by U_p and H_p, then there won't be any nanopublications on the peer server that this server is interested in, and we jump to step 9.

5. The server will start at position n to look for new nanopublications at server p: n is set to the total number of nanopublications of the last visit n'_p, or to 0 if there was no last visit (nanopublication counting starts at 0).

6. If the retrieved journal identifier j_p is different from j'_p (meaning that the server has been reset since the last visit), n is set to 0.

7. If $n = n_p$, meaning that there are no new nanopublications since the last visit, the server jumps to step 9.

8. All journal pages p starting from the one containing n until the end of the journal are downloaded one by one (considering the size of journal pages, which is by default 1000 nanopublications):

   (a) All nanopublication identifiers in p (excluding those before n) are checked with respect to whether (A) they are covered by the visiting server's patterns U_s and H_s and (B) they are not already contained in the local store. A list l is created of all nanopublication identifiers of the given page that satisfy both, (A) and (B).

   (b) If the number of new nanopublications $|l|$ exceeds a certain threshold (currently set to 5), the nanopublications of p are downloaded as a gzipped package. Otherwise, the new nanopublications (if any) are requested individually.

   (c) The retrieved nanopublications that are in list l are validated using their trusty URIs, and all *valid* nanopublications are loaded to the server's nanopublication store and their identifiers are added to the end of the server's own journal. (Invalid nanopublications are ignored.)

9. The journal identifier j_p and the total number of nanopublications n_p for server p are remembered for the next visit, replacing the values of j'_p and n'_p.

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. The current system and its protocol are not set in stone but, if successful, will have to evolve in the future — in particular with respect to network topology and partial replication — to accommodate a network of possibly thousands of servers and billions of nanopublications.

PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016

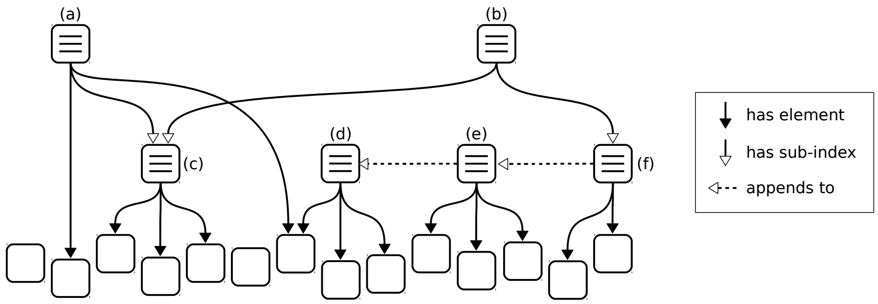


Figure 3. Schematic example of nanopublication indexes

3.3 Nanopublication Indexes

To make the infrastructure described above practically useful, we have to introduce the concept of indexes. One of the core ideas behind nanopublications is that each of them is a tiny atomic piece of data. This implies that analyses will mostly involve more than just one nanopublication and typically a large number of them. Similarly, most processes will generate more than just one nanopublication, possibly thousands or even millions of them. Therefore, we need to be able to group nanopublications and to identify and use large collections of them.

Given the versatility of the nanopublication standard, it seems straightforward to represent such collections as nanopublications themselves. However, if we let such "collection nanopublications" contain other nanopublications, then the former would become very large for large collections and would quickly lose their 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 call them *indexes* and not collections.

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, and to support sets of arbitrary size, we can allow indexes to be appended by other indexes. We set 1000 nanopublication references as the upper limit any single index can directly contain. This limit is admittedly arbitrary, but it seems to be a reasonable compromise between ensuring that nanopublications remain small on the one hand and limiting the number of nanopublications needed to define large indexes on the other. A set of 100,000 nanopublications, for example, can therefore be defined by a sequence of 100 indexes, where the first one stands for the first 1000 nanopublications, the second one appends to the first and adds another 1000 nanopublications (thereby representing 2000 of them), and so on up to the last index, which appends to the second to last and thereby stands for the entire set. In addition, to allow datasets to 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 to* 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 3. Index (a) in the shown example contains five nanopublications, three of them via sub-index (c). The latter is also part of index (b), which additionally contains eight nanopublications via sub-index (f). Two of these eight nanopublications belong directly to (f), whereas the remaining six come from appending to index (e). Index (e) in turn gets half of its nanopublications by appending to index (d). We see that some nanopublications may not be referenced by any index at all, while others may belong to several indexes at the same time.

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.

3.4 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, using the `np` command from the `nanopub-java`

PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016

library (Kuhn, 2015a). 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 (requiring Git and Maven):

```
$ git clone https://github.com/Nanopublication/nanopub-java.git
$ cd nanopub-java
$ mvn package
```

And for convenience reasons, we can add the *bin* directory to the path variable:

```
$ PATH=`pwd`/bin:$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:

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

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

```
: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:

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

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

```
: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:

```
@prefix : <http://example.org/np2#>.
...
  ex:Gene1 ex:isRelatedTo ex:malaria.
...

@prefix : <http://example.org/np3#>.
...
  ex:Gene2 ex:isRelatedTo ex:malaria.
...
```

We save these nanopublications in a file `nanopubs.trig`, and before we can publish them, we have to assign them trusty URIs:

```
$ np mktrusty -v nanopubs.trig
Nanopub URI: http://example.org/np1#RAQoZlp22LHIvtYqHCosPbUtX8yeGs1Y5AfqcjMneLQ2I
Nanopub URI: http://example.org/np2#RAT5swlSLyMbuD03KzJsYHVV2oM1wRhluRxMrvpkZCDUQ
Nanopub URI: http://example.org/np3#RAkvUpysi9Ql3itlc6-iIJMG7YSt3-PI8dAJXcmafU71s
```


PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016

This gives us the file `trusty.nanopubs.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:

```
$ np publish 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):

```
$ np status -a RAQoZlp22LHIvtYqHCosPbUtX8yeGs1Y5AfqcjMneLQ2I
URL: http://np.inn.ac/RAQoZlp22LHIvtYqHCosPbUtX8yeGs1Y5AfqcjMneLQ2I
Found on 1 nanopub server.
```

This is what you see immediately after publication. Only one server knows about the new nanopublication. Some minutes later, however, the same command leads to something like this:

```
$ np status -a RAQoZlp22LHIvtYqHCosPbUtX8yeGs1Y5AfqcjMneLQ2I
URL: http://np.inn.ac/RAQoZlp22LHIvtYqHCosPbUtX8yeGs1Y5AfqcjMneLQ2I
URL: http://ristretto.med.yale.edu:8080/nanopub-server/RAQoZlp22LHIvtYqHCosPbU...
URL: http://nanopubs.stanford.edu/nanopub-server/RAQoZlp22LHIvtYqHCosPbUtX8yeG...
URL: http://nanopubs.semanticscience.org:8082/RAQoZlp22LHIvtYqHCosPbUtX8yeGs1Y...
URL: http://rdf.disgenet.org/nanopub-server/RAQoZlp22LHIvtYqHCosPbUtX8yeGs1Y5A...
URL: http://app.tkuhn.eculture.labs.vu.nl/nanopub-server-2/RAQoZlp22LHIvtYqHCo...
URL: http://nanopubs.restdesc.org/RAQoZlp22LHIvtYqHCosPbUtX8yeGs1Y5AfqcjMneLQ2I
URL: http://nanopub.backend1.scify.org/nanopub-server/RAQoZlp22LHIvtYqHCosPbUt...
URL: http://nanopub.exynize.com/RAQoZlp22LHIvtYqHCosPbUtX8yeGs1Y5AfqcjMneLQ2I
Found on 9 nanopub servers.
```

Next, we can make an index pointing to these three nanopublications:

```
$ np mkindex -o index.nanopubs.trig trusty.nanopubs.trig
Index URI: http://np.inn.ac/RAXsXUhY8iDbfDdY6sm64hRFPr7eAwYXRlSsqQAz1LE14
```

This creates a local file `index.nanopubs.trig` containing the index, identified by the URI shown above. As this index is itself a nanopublication, we can publish it in the same way:

```
$ np publish index.nanopubs.trig
1 nanopub published at http://np.inn.ac/
```

Once published, we can check the status of this index and its contained nanopublications:

```
$ np status -r RAXsXUhY8iDbfDdY6sm64hRFPr7eAwYXRlSsqQAz1LE14
1 index nanopub; 3 content nanopubs
```

Again, after just a few minutes this nanopublication will be distributed in the network and available on multiple servers. From this point on, everybody can conveniently and reliably retrieve the given set of nanopublications. The only thing one needs to know is the artifact code of the trusty URI of the index:

```
$ np get -c RAXsXUhY8iDbfDdY6sm64hRFPr7eAwYXRlSsqQAz1LE14
```

This command downloads the nanopublications of the index we 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 URL `http://np.inn.ac/` or the URL of any other nanopublication server):

```
RAEoxLTy4pEJYbZwA9FuBJ6ogSquJobFitoFMbUmkBJh0
RAoMW0xMemwKEjCNWLFt8CgRmg_TGjfVSsh15hGfEmcz4
RA3BH_GncwEK_UXFGTvHcMVZ1hW775eupAccDdho5Tiow
RA3HvJ69nO0mD5d4m4u-Oc4bpXlxIWYN6L3wvB9jntTXk
RASx-fnzWJzluqRDe6GVMWFEyWLok8S6nTNkyElwapwno
```

These nanopublications about the same protein come from two different sources: The first one is from the BEL2nanopub dataset, whereas the remaining four are from neXtProt.[4] These nanopublications

[4] See https://github.com/tkuhn/bel2nanopub and http://nextprot2rdf.sourceforge.net, respectively, and Table 1

PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016

can be downloaded as above with the `np get` command and stored in a file, which we name here `cdkn2a-nanopubs.trig`.

In order to be able to refer to such a collection of nanopublications with a single identifier, a new index is needed that contains just these five nanopublications. This time we give the index a title (which is optional):

```
$ np mkindex -t "Data about CDKN2A from BEL2nanopub & neXtProt" \
  -o index.cdkn2a-nanopubs.trig cdkn2a-nanopubs.trig
Index URI: http://np.inn.ac/RA6jrrPL2NxxFWlo6HFWas1ufp0OdZzS_XKwQDXpJg3CY
```

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:

```
$ np publish index.cdkn2a-nanopubs.trig
1 nanopub published at http://np.inn.ac/
```

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 following:

> [1] Data about CDKN2A from BEL2nanopub & neXtProt. Nanopublication index http://np.inn.ac/RA6jrrPL2NxxFWlo6HFWas1ufp0OdZzS_XKwQDXpJg3CY, 14 April 2015.

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 much 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, as described below in Section 4, in exactly this way (NP Index `RAY_lQruua`, 2015; NP Index `RACy0I4f_w`, 2015; NP Index `RAR5dwELYL`, 2015; NP Index `RAXy332hxq`, 2015; NP Index `RAVEKRW0m6`, 2015; NP Index `RAXFlG04YM`, 2015; NP Index `RA7SuQ0e66`, 2015).

The new finding that was deduced from the given five nanopublications can, 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/>.
...
:assertion {
  uniprot:P42771 a ex:proteinWithPropertyX.
}
:provenance {
  :assertion prov:wasInfluencedBy
     nps:RA6jrrPL2NxxFWlo6HFWas1ufp0OdZzS_XKwQDXpJg3CY.
}
:pubinfo {
  : pav:createdBy <http://orcid.org/0000-0002-1267-0234>.
  : dc:created "2015-04-14T08:05:43+01:00"^^xsd:dateTime.
}
```

We can again transform it to a trusty nanopublication, 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 4. 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.

Figure 4. The web interface of the nanopublication validator can load nanopublications by their trusty URI (or just their artifact code) from the anopublication server network. It also allows users to directly publish uploaded nanopublications.

4 EVALUATION

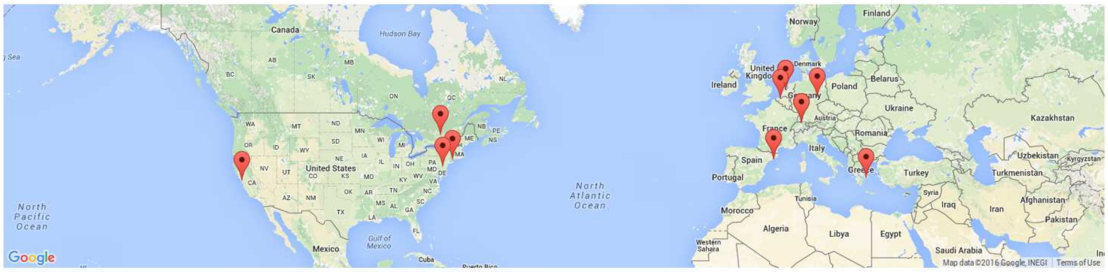
To evaluate our approach, we want to find out whether a small server network run on normal Web servers, without dedicated infrastructure, is able to handle the amount of nanopublications we can expect to become publicly available in the next few years. At the time the evaluation was performed, the server network consisted of three servers in Zurich, New Haven, and Ottawa. Seven new sites in Amsterdam, Stanford, Barcelona, Ghent, Athens, Leipzig, and Haverford have joined the network since. The current network of 15 server instances on 10 sites (in 8 countries) is shown in Figure 5, which is a screenshot of a nanopublication monitor that we have implemented[5]. Such monitors regularly check the nanopublication server network, register changes (currently once per minute), and test the response times and the correct operation of the servers by requesting a random nanopublication and verifying the returned data.

4.1 Evaluation Design

Table 1 shows seven existing nanopublication datasets, five of which we used for the first part of the evaluation (the other two were not yet available at the time this evaluation was conducted). These five datasets consist of a total of more than 5 million nanopublications and close to 200 million RDF triples, including nanopublication indexes that we generated for each dataset. The total size of these five datasets when stored as uncompressed TriG files amounts to 15.6 GB. Each of the datasets is assigned to one of the three servers, where it is loaded from the local file systems. The first nanopublications start spreading to the other servers, while others are still being loaded from the file system. We therefore test the reliability and capacity of the network under constant streams of new nanopublications coming from different servers, and we use two nanopublication monitors (in Zurich and Ottawa) to evaluate the responsiveness of the network.

In the second part of the evaluation we expose a server to heavy load from clients to test its retrieval capacity. For this we use a service called Load Impact[6] to let up to 100 clients access a nanopublication server in parallel. We test the server in Zurich over a time of five minutes under the load from a linearly

[5] https://github.com/tkuhn/nanopub-monitor
[6] https://loadimpact.com



URL	Status	OK Ratio	Resp Time (Dist)	Last Seen OK	NP Count	IP Address	Server Location	Version	URI / Hash Pattern
http://ristretto.med.yale.edu:8080/nanopub-server/	OK	99.98538%	195 ms (6212 km)	2016-02-05 08:06:23	6370147	128.36.40.86	New Haven, United States	0.2	/
http://np.inn.ac/	OK	99.99707%	29 ms (0 km)	2016-02-05 08:06:27	6370147	129.132.255.27	Zurich, Switzerland	0.3	/
http://npx1.inn.ac/	OK	100.0%	6 ms (0 km)	2016-02-05 08:06:26	398634	129.132.255.27	Zurich, Switzerland	0.3	/ A B C D
http://app.tkuhn.eculture.labs.vu.nl/nanopub-server-1/	OK	99.970764%	33 ms (615 km)	2016-02-05 08:06:26	1593807	130.37.193.11	Amsterdam, Netherlands	0.3	/ A B C D E F G H I J K L M N O P
http://app.tkuhn.eculture.labs.vu.nl/nanopub-server-2/	OK	99.98538%	54 ms (615 km)	2016-02-05 08:06:24	1592080	130.37.193.11	Amsterdam, Netherlands	0.3	/ Q R S T U V W X Y Z a b c d e f
http://app.tkuhn.eculture.labs.vu.nl/nanopub-server-3/	OK	99.988304%	32 ms (615 km)	2016-02-05 08:06:24	1592191	130.37.193.11	Amsterdam, Netherlands	0.3	/ g h i j k l m n o p q r s t u v
http://app.tkuhn.eculture.labs.vu.nl/nanopub-server-4/	OK	99.97953%	34 ms (615 km)	2016-02-05 08:06:27	1592069	130.37.193.11	Amsterdam, Netherlands	0.3	/ w x y z 0 1 2 3 4 5 6 7 8 9 - _
http://nanopubs.semanticscience.org:8082/	OK	99.94445%	215 ms (6131 km)	2016-02-05 08:06:22	6370147	134.117.221.11	Ottawa, Canada	0.4	/
http://nanopub.exynize.com/	OK	100.0%	49 ms (516 km)	2016-02-05 08:06:23	6370147	139.18.2.164	Leipzig, Germany	0.4	/
http://nanopub.backend1.scify.org/nanopub-server/	OK	99.9775%	185 ms (1616 km)	2016-02-05 08:06:27	6370147	143.233.226.41	Athens, Greece	0.4	/
http://nanopubs.restdesc.org/	OK	99.99707%	66 ms (539 km)	2016-02-05 08:06:22	6370147	157.193.213.74	Ghent, Belgium	0.3	/
http://digitalduchemin.org/np-mirror/	OK	99.0991%	196 ms (6460 km)	2016-02-05 08:06:21	199421	165.82.124.16	Haverford, United States	0.4	/ E F
http://nanopubs.stanford.edu/nanopub-server/	OK	100.0%	375 ms (9394 km)	2016-02-05 08:06:25	6370147	171.67.213.57	Stanford, United States	0.2	/
http://rdf.disgenet.org/nanopub-server-disgenet/	OK	99.994156%	57 ms (835 km)	2016-02-05 08:06:26	1959788	84.89.134.134	Barcelona, Spain	0.3	http://rdf.disgenet.org/ /
http://rdf.disgenet.org/nanopub-server/	OK	99.988304%	90 ms (835 km)	2016-02-05 08:06:24	6370147	84.89.134.134	Barcelona, Spain	0.2	/

Figure 5. This screenshot of the nanopublication monitor interface (http://npmonitor.inn.ac) showing the current server network. It currently consists of 15 server instances on 10 physical servers in Zurich, New Haven, Ottawa, Amsterdam, Stanford, Barcelona, Ghent, Athens, Leipzig, and Haverford.

increasing number of clients (from 0 to 100) located in Dublin. These clients are programmed to request a randomly chosen journal page, then to go though the entries of that page one by one, requesting the respective nanopublication with a probability of 10%, and starting over again with a different page. As a comparison, we run a second session, for which we load the same data into a Virtuoso SPARQL endpoint on the same server in Zurich (with 16 GB of memory given to Virtuoso and two 2.40 GHz Intel Xeon processors). Then, we perform exactly the same stress test on the SPARQL endpoint, requesting the nanopublications in the form of SPARQL queries instead of requests to the nanopublication server interface. This comparison is admittedly not a fair one, as SPARQL endpoints are much more powerful and are not tailor-made for the retrieval of nanopublications, but they provide nevertheless a valuable and well-established reference point to evaluate the performance of our system.

4.2 Evaluation Results

The first part of 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 6 shows the rate at which the nanopublications were loaded at their first, second, and third server, respectively. 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). Figure 7 shows the response times of the three servers as measured by the two nanopublication monitors in Zurich (top) and Ottawa (bottom) during the time of the evaluation. 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 0.21 seconds when the server was on the same continent as the measuring monitor. In 99.77% of all cases (including those across continents) the response time was below 0.5 seconds, and it was always below 1.1 seconds. Not a single one of the 4802 individual HTTP requests timed out, led to an error, or received a nanopublication that could not be successfully verified.

Figure 8 shows the result of the second part of the evaluation. The nanopublication server was able to handle 113,178 requests in total (i.e. an average of 377 requests per second) with an average response time of 0.12 seconds. In contrast, the SPARQL endpoint answering the same kind of requests needed 100 times

PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016

dataset *name and citation*	# nanopublications *index*	*content*	# triples *index*	*content*	used for evaluation
GeneRIF/AIDA (NP Index `RAY_lQruua`, 2015)	157	156,026	157,909	2,340,390	✓
OpenBEL 1.0 (NP Index `RACy0I4f_w`, 2015)	53	50,707	51,448	1,502,574	✓
OpenBEL 20131211 (NP Index `RAR5dwELYL`, 2015)	76	74,173	75,236	2,186,874	✓
DisGeNET v2.1.0.0 (NP Index `RAXy332hxq`, 2015)	941	940,034	951,325	31,961,156	✓
DisGeNET v3.0.0.0 (NP Index `RAVEKRW0m6`, 2015)	1,019	1,018,735	1,030,962	34,636,990	
neXtProt (NP Index `RAXFlG04YM`, 2015)	4,026	4,025,981	4,078,318	156,263,513	✓
LIDDI (NP Index `RA7SuQ0e66`, 2015)	99	98,085	99,272	2,051,959	
total	6,371	6,363,741	6,444,470	230,943,456	5

Table 1. Existing datasets in the nanopublication format that were used for the first part of the evaluation.

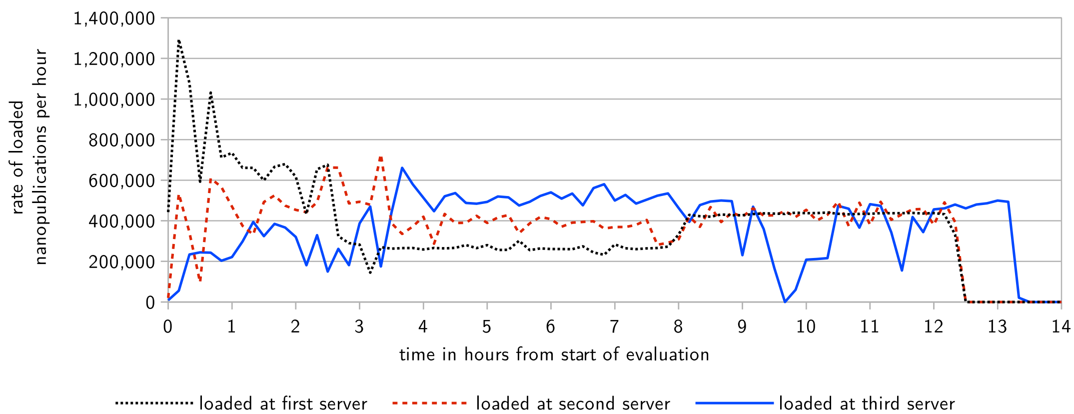


Figure 6. This diagram shows the rate at which nanopublications are loaded at their first, second, and third server, respectively, over the time of the evaluation. At the first server, nanopublications are loaded from the local file system, whereas at the second and third server they are retrieved via the server network.

longer to process them (13 seconds on average), consequently handled about 100 times fewer requests (1267), and started to hit the timeout of 60 seconds for some requests when more than 40 client accessed it in parallel. In the case of the nanopublication server, the majority of the requests were answered within less than 0.1 seconds for up to around 50 parallel clients, and this value remained below 0.17 seconds all the way up to 100 clients. As the round-trip network latency alone between Ireland and Zurich amounts to around 0.03 to 0.04 seconds, further improvements can be achieved for a denser network due to the reduced distance to the nearest server.

The first part of the evaluation shows that the overall replication capacity of the current server network is around 9.4 million new nanopublications per day or 3.4 billion per year. The results of the second part show that the load on a server when measured as response times is barely noticeable for up to 50 parallel clients, and therefore the network can easily handle $50 \cdot x$ parallel client connections or more, where x is the number of independent physical servers in the network (currently $x = 10$). The second part thereby also shows that the restriction of avoiding parallel outgoing connections for the replication between servers is actually a very conservative measure that could be relaxed, if needed, to allow for a higher replication capacity.

PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016

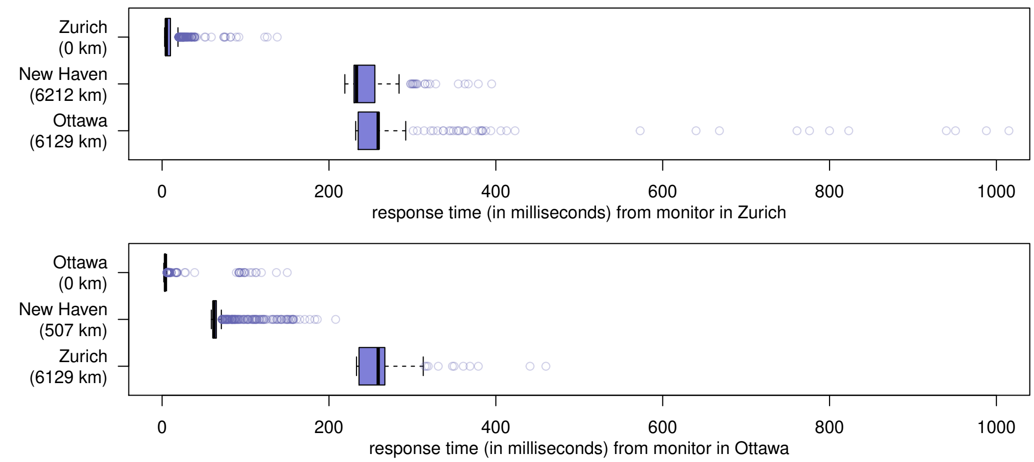


Figure 7. Server response times under heavy load, recorded by the monitors during the first evaluation

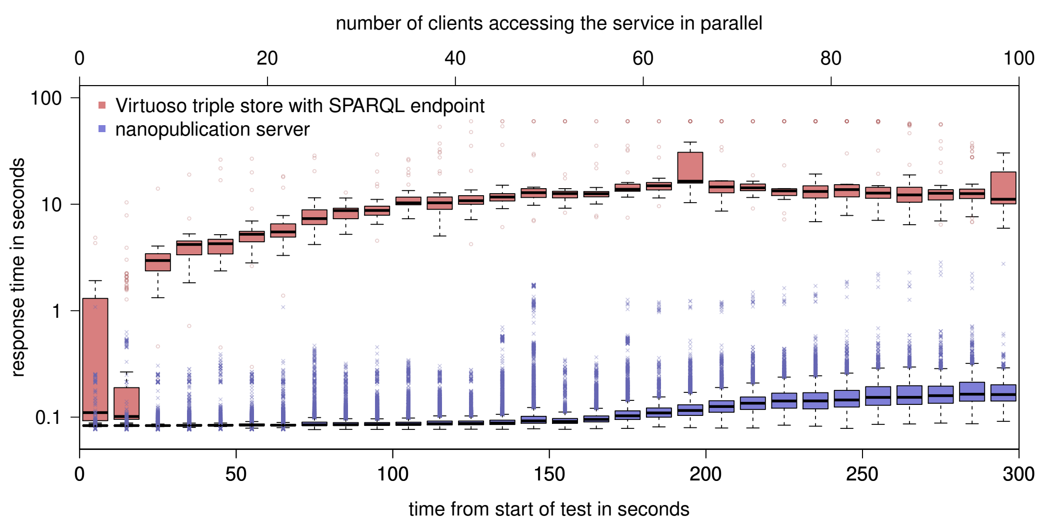


Figure 8. Results of the evaluation of the retrieval capacity of a nanopublication server as compared to a general SPARQL endpoint (note the logarithmic y-axis)

5 DISCUSSION AND CONCLUSION

We have presented here a low-level infrastructure for data sharing, which is just one piece of a bigger ecosystem to be established. The implementation of components that rely on this low-level data sharing infrastructure is ongoing and future work. This includes the development of "core services" (see Section 3.1) on top of the server network to allow people to find nanopublications and "advanced services" to query and analyze the content of nanopublications. In addition, we need to establish standards and best practices of how to use existing ontologies (and to define new ones where necessary) to describe properties and relations of nanopublications, such as referring to earlier versions, marking nanopublications as retracted, and reviewing of nanopublications.

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 even more, i.e. replicate only a part of all nanopublications, in order to handle really large amounts of data. In addition to the current surface feature definitions via URI and hash patterns, a number of additional ways of specializing are possible in the future: Servers can restrict themselves to

PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016

particular types of nanopublications, e.g. to specific topics or authors, and communicate this to the network in a similar way as they do it now with URI and hash patterns; 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 (Jacobson et al., 2012), with which — consistent with our proposal — requests are based on content rather than hosts.

We argue that data publishing and archiving can and should be done in a decentralized manner. We believe that the presented server 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.

REFERENCES

Banda, J. M., Kuhn, T., Shah, N. H., and Dumontier, M. (2015). Provenance-centered dataset of drug-drug interactions. In *Proceedings of the 14th International Semantic Web Conference (ISWC 2015)*, pages 293–300. Springer.

Belhajjame, K., Corcho, O., Garijo, D., Zhao, J., Missier, P., Newman, D., Palma, R., Bechhofer, S., Garcıa, E., Cuesta, J. M. G.-P., et al. (2012). Workflow-centric research objects: First class citizens in scholarly discourse. In *Proceedings of SePublica 2012*. CEUR-WS.

Berners-Lee, T. (2006). Linked data — design issues. http://www.w3.org/DesignIssues/LinkedData.html.

Bradley, J. (2005). Documents and data: Modelling materials for humanities research in xml and relational databases. *Literary and linguistic computing*, 20(1):133–151.

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

Carroll, J., Bizer, C., Hayes, P., and Stickler, P. (2005). Named graphs, provenance and trust. In *Proceedings of WWW '05*, pages 613–622. ACM.

Chichester, C., Gaudet, P., Karch, O., Groth, P., Lane, L., Bairoch, A., Mons, B., and Loizou, A. (2014). 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*.

Chichester, C., Karch, O., Gaudet, P., Lane, L., Mons, B., and Bairoch, A. (2015). Converting neXtProt into linked data and nanopublications. *Semantic Web*.

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

Cohen, J. P. and Lo, H. Z. (2014). Academic torrents: A community-maintained distributed repository. In *Proceedings of XSEDE '14*, page 2. ACM.

Feigenbaum, L., Williams, G. T., Clark, K. G., and Torres, E. (2013). SPARQL 1.1 protocol. Recommendation, W3C.

Filali, I., Bongiovanni, F., Huet, F., and Baude, F. (2011). A survey of structured P2P systems for RDF data storage and retrieval. In *Transactions on large-scale data- and knowledge-centered systems III*, pages 20–55. Springer.

Freedman, R. (2014). The renaissance chanson goes digital: digitalduchemin. org. *Early Music*, 42(4):567–578.

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

Golden, P. and Shaw, R. (2016). Nanopublication beyond the sciences. *PeerJ Computer Science*.

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

Harris, S. and Seaborne, A. (2013). SPARQL 1.1 query language. Recommendation, W3C.

Hartig, O. (2013). An overview on execution strategies for Linked Data queries. *Datenbank-Spektrum*, 13(2):89–99.

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

PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016

Kuhn, T. (2014). A survey and classification of controlled natural languages. *Computational Linguistics*, 40(1):121–170.
Kuhn, T. (2015a). nanopub-java: A Java library for nanopublications. In *Proceedings of the 5th Workshop on Linked Science (LISC 2015)*.
Kuhn, T. (2015b). Science bots: A model for the future of scientific computation? In *WWW 2015 Companion Proceedings*, pages 1061–1062. ACM.
Kuhn, T., Barbano, P. E., Nagy, M. L., and Krauthammer, M. (2013). Broadening the scope of nanopublications. In *Proceedings of ESWC 2013*, pages 487–501. Springer.
Kuhn, T., Chichester, C., Krauthammer, M., and Dumontier, M. (2015). Publishing without publishers: a decentralized approach to dissemination, retrieval, and archiving of data. In *Proceedings of the 14th International Semantic Web Conference (ISWC 2015)*, Lecture Notes in Computer Science. Springer.
Kuhn, T. and Dumontier, M. (2014). Trusty URIs: Verifiable, immutable, and permanent digital artifacts for linked data. In *Proceedings of ESWC 2014*, pages 395–410. Springer.
Kuhn, T. and Dumontier, M. (2015). Making digital artifacts on the web verifiable and reliable. *IEEE Transactions on Knowledge and Data Engineering*, 27(9).
Kuhn, T., Royer, L., Fuchs, N. E., and Schroeder, M. (2006). Improving text mining with controlled natural language: A case study for protein interations. In *Proceedings DILS'06*. Springer.
Ladwig, G. and Harth, A. (2011). CumulusRDF: linked data management on nested key-value stores. In *Proceedings of SSWS 2011*.
Markman, C. and Zavras, C. (2014). BitTorrent and libraries: Cooperative data publishing, management and discovery. *D-Lib Magazine*, 20(3):5.
McCusker, J. P., Lebo, T., Krauthammer, M., and McGuinness, D. L. (2013). Next generation cancer data discovery, access, and integration using prizms and nanopublications. In *Proceedings of DILS 2013*, pages 105–112. Springer.
Miller, A., Juels, A., Shi, E., Parno, B., and Katz, J. (2014). Permacoin: Repurposing Bitcoin work for data preservation. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 475–490. IEEE.
Mons, B., van Haagen, H., Chichester, C., den Dunnen, J. T., van Ommen, G., van Mulligen, E., Singh, B., Hooft, R., Roos, M., Hammond, J., et al. (2011). The value of data. *Nature genetics*, 43(4):281–283.
NP Index `RA7SuQ0e66` (2015). Linked Drug-Drug Interactions (LIDDI). Nanopublication index http://np.inn.ac/RA7SuQ0e661LJdKpt5EOS2DKykf1ht9LFmNaZtFSDMrXg.
NP Index `RACy0I4f_w` (2015). Nanopubs converted from OpenBEL's Small and Large Corpus 1.0. Nanopublication index http://np.inn.ac/RACy0I4f_wr62O17BhnD5EkJU6Glf-wp0oPbDbyve7P6o.
NP Index `RAR5dwELYL` (2015). Nanopubs converted from OpenBEL's Small and Large Corpus 20131211. Nanopublication index http://np.inn.ac/RAR5dwELYLKGSfrOclnWhjOj-2nGZN_8BW1JjxwFZINHw.
NP Index `RAVEKRW0m6` (2015). Nanopubs extracted from DisGeNET v3.0.0.0. Nanopublication index http://np.inn.ac/RAVEKRW0m6Ly_PjmhcxCZMR5fYIlzzqjOWt1CgcwD_77c.
NP Index `RAXFlG04YM` (2015). Nanopubs converted from neXtProt protein data (preliminary). Nanopublication index http://np.inn.ac/RAXFlG04YMi1A5su7oF6emA8mSp6HwyS3mFTVYreDeZRg.
NP Index `RAXy332hxq` (2015). Nanopubs extracted from DisGeNET v2.1.0.0. Nanopublication index http://np.inn.ac/RAXy332hxqHPKpmvPc-wqJA7kgWiWa-QA0DIpr29LIG0Q.
NP Index `RAY_lQruua` (2015). AIDA Nanopubs extracted from GeneRIF. Nanopublication index http://np.inn.ac/RAY_lQruuagCYtAcKAPptkY7EpITwZeUilGHsWGm9ZWNI.
Paskin, N. (2005). Digital object identifiers for scientific data. *Data Science Journal*, 4:12–20.
Patrinos, G. P., Cooper, D. N., van Mulligen, E., Gkantouna, V., Tzimas, G., Tatum, Z., Schultes, E., Roos, M., and Mons, B. (2012). Microattribution and nanopublication as means to incentivize the placement of human genome variation data into the public domain. *Human mutation*, 33(11):1503–1512.
Proell, S. and Rauber, A. (2014). A scalable framework for dynamic data citation of arbitrary structured data. In *3rd International Conference on Data Management Technologies and Applications (DATA2014)*.
Queralt-Rosinach, N., Kuhn, T., Chichester, C., Dumontier, M., Sanz, F., and Furlong, L. I. (2015). Publishing DisGeNET as nanopublications. *Semantic Web — Interoperability, Usability, Applicability*.
Speicher, S., Arwe, J., and Malhotra, A. (2015). Linked data platform 1.0. Recommendation, W3C.
Verborgh, R., Hartig, O., De Meester, B., Haesendonck, G., De Vocht, L., Vander Sande, M., Cyganiak,

PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016

R., Colpaert, P., Mannens, E., and Van de Walle, R. (2014). Querying datasets on the Web with high availability. In Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C., Vrandečić, D., Groth, P., Noy, N., Janowicz, K., and Goble, C., editors, *Proceedings of the 13th International Semantic Web Conference*, volume 8796 of *Lecture Notes in Computer Science*, pages 180–196. Springer.

Williams, A. J., Harland, L., Groth, P., Pettifer, S., Chichester, C., Willighagen, E. L., Evelo, C. T., Blomberg, N., Ecker, G., Goble, C., et al. (2012). Open PHACTS: semantic interoperability for drug discovery. *Drug discovery today*, 17(21):1188–1198.

PeerJ PrePrints | https://doi.org/10.7287/peerj.preprints.1760v1 | CC-BY 4.0 Open Access | rec: 19 Feb 2016, publ: 19 Feb 2016