Low-bandwidth Semantic Web

Onno Valkering *

Vrije Universiteit Amsterdam, the Netherlands
o.a.b.valkering@student.vu.nl

Abstract. Many ICT applications and services, including those from the Semantic Web, rely on the Web for the exchange of data. This includes expensive server and network infrastructure. Most rural areas of developing countries are not reached by the Web and its possibilities, while at the same time the ability to share knowledge has been identified as a key enabler for development. To make widespread knowledge sharing possible in these rural areas, the notion of the Web has to be downscaled based on the specific low-resource infrastructure in place. In this paper, we introduce SPARQL over SMS, a solution for exchanging RDF data in which HTTP is substituted by SMS to enable Web-like exchange of data over cellular networks. We motivate this through two use cases under development in West-Africa, which are also used for the validation of the approach. We present the design and implementation of the solution, along with a data compression method that combines generic compression strategies and strategies that use Semantic Web specific features to reduce the size of RDF before it is transferred over the low-bandwidth cellular network.

1 Introduction

The Semantic Web by design builds on, and relies on, the Web infrastructure for data exchange. This includes sophisticated server and network infrastructure which are unavailable in many rural areas of developing countries. These areas are not reached by the web and its possibilities while at the same time the ability to share knowledge has been identified as a key enabler for development. To make knowledge sharing possible in rural developing areas, the notion of the Web has to be *downscaled* based on the specific low-resource infrastructure in place [6].

Data sharing solutions, such as those based on Semantic Web and Linked Data technology, should not only be accessible to those with abundant resources and reliable infrastructures, but also in low-resource environments. Its language-agnostic nature and flexible graph models make Semantic Web technologies especially useful for data sharing in these locations. In [3] we show that locally produced market data, stored as RDF, is produced through, and used in, a

* This master thesis is supervised by Victor de Boer and Stefan Schlobach and is an extended version of [13]. Gossa Lô and Romy Blankendaal contributed to the case motivation in section 2.1

voice-interface accessible for low-literate users in their preferred language. Unfortunately, the initial success-story ended here, as low-tech options for semantic data exchange did not exist. More recently, we developed the Kasadaka[1], a low-resource prototyping and computing platform which uses semantic technologies specialized at developing multi-modal user interfaces. While this widens the applicability and possible use-cases, a core problem remains; the lack of infrastructure for Semantic Web sharing of information in most parts of the world.

The main challenge lies in the unavailability of network connections. Especially in many rural areas of developing countries, internet connections are either missing or extremely unreliable. For example, Internet penetration is estimated to be 28.6% of the population in Africa as a whole, with some countries reaching considerably less of their population: 19.6% in Ghana and 7.0% in Mali where two of our use cases presented here are situated[2]. These numbers include both urban and rural areas and in the latter, internet penetration is virtually non-existent.

By design, the Semantic Web is built on top of the Internet (TCP/IP) and World Wide Web infrastructure (including HTTP) and as such when no Internet is available, the conventional Semantic Web is unusable. However, we can design solutions to implement Semantic Web-like data sharing using alternative networking capabilities available in low-resource environments.

In this paper, we present a specific downscaling solution for exchanging (RDF) data in which HTTP is substituted by SMS[3] to enable Web-like exchange of data over cellular networks. We show the viability of this solution in two different ways:

1. Technological: we identify three main technological problems when using an SMS protocol as semantic data transfer protocol, message size, the asynchronous nature of the protocol and how to deal with pagination issues. Our solution is validated w.r.t. each of those problems with a variety of methods, which includes a large-scale empirical comparison of compression size.
2. Societal: Using two use-cases from Sub-Saharan Africa (one from Ghana, one from Mali) we will show how SMS-based Semantic Web can practically address the knowledge sharing needs of rural communities.

While in this paper, we present a practical solution to low-bandwidth knowledge sharing, our investigation will also be more generally useful to understand how Semantic Web principles and practices can be separated from the infrastructure layers that often are assumed to be prerequisites.

This paper is structured as follows: in Section 2 we present two motivating use cases that call for the use of Semantic technologies in low-resource environments based on the Kasadaka. In Section 3 we present our technical solution to use SMS messages to replace HTTP for transferring Semantic Web data using the

[1] `http://www.kasadaka.com`

[2] Internet Penetration Africa November 2015`http://www.internetworldstats.com/` Internet World Statistics, Miniwatts Marketing Group.

[3] `https://en.wikipedia.org/wiki/Short_Message_Service`

GSM network. In Section 4 we identify some major challenges and evaluate how our solution addresses these challenges, in particular focusing on various data compression methods to understand an enable the use of this low-bandwidth channel. In Section 5 we finally validate our technical solution w.r.t. our use-cases. We conclude the paper with relating our work with other work and a discussion.

2 Information Sharing in the Absence of the Web

As early as 2011 we pointed to some negative effects of the Semantic Web's reliance on Web infrastructure [6], which effectively made this technology inaccessible for a majority of the world population. Through a number of research projects in Sub-Saharan Africa we have since then identified numerous use-cases that rely on knowledge sharing. For many of those cases we created hardware platforms, mostly built on existing infrastructure and processes, which addressed the respective information needs without use of semantic technology.

The recent Kasadaka project aims at providing information to people living in rural communities at a larger scale (in terms of use-cases). Kasadaka is a generic platform, which enables voice- and SMS-based communication over GSM. It can be deployed in these communities and be owned and maintained by local stakeholders. The platform can host different information sharing services, all accessible through simple interfaces. As users are often low-literate and/or speak local languages, Kasadaka allows for icon-based visual interfaces and voice interfaces accessible through simple mobile phones. It also allows to connect to users through their own devices using an often available and reliable network.

We regularly ran into conceptual and technical problems for which the Semantic Web community has already provided robust methods. While the usage of a triple store for data hosting has shown some benefits, most real use-cases require data and knowledge sharing across communities and devices. In the following we describe two use-cases for which we have been developing information sharing solutions. Both have been co-developed with local stakeholders in rural West-Africa, both are impossible to fully realize with current technology.

2.1 The DigiVet Case

Inaccessibility of knowledge influences the socio-economic development in rural areas in Ghana. Poor infrastructure and lack of education are two factors that increase communication gaps between experts and rural laymen. One of the information needs that the farmers have is on animal health and in particular on diagnosing animals. DigiVet is a voice-based veterinary information service that aims to support subsistence farmers in making the decision whether or not to visit a veterinarian, while bringing them into contact with each other. Animal diseases spread within and between villages, and can often be cured merely with the intervention of a veterinarian. The problem that arises in these rural areas is that local expertise is often lacking and poor infrastructures (poor roads and

lack of electricity) prevent access to information and sharing of knowledge. Some farmers indicated that they would like to have information on animal diseases, disease patterns, diagnosis and symptoms, to enable them to take preventive action and preclude cattle loss. This goes to show that equipping them with locally available veterinary information could decrease animal mortality rates.

The current version of DigiVet includes a simple interface in which farmers can click through a set of symptom related questions on a touch screen connected to the Kasadaka. It is based around a knowledge base[4] developed by interviewing veterinarians working in rural Northern Ghana. Based on these interviews, the objective of the system is to provide a diagnosis whether or not a farmer should see a veterinarian. This is done in order to prevent farmers from curing the animals themselves and in order to stimulate them to more proactively visit a veterinarian. DigiVet is a diagnostic system that heavily relies on an underlying knowledge-based system, and on semantic information transfer between village farmers and veterinarians at large physical distances. While the Kasadaka is a suitable platform for the diagnostic system, there is currently no technology in these communities to cater for the necessary semantic data exchange.

2.2 The RadioMarché Case

This use case was initially described in [2]. It deals with a Market Information System designed to gather and distribute information about offerings of specific produce on local markets in the Tominian region of Mali. To allow low-literate stakeholders to retrieve market information using simple mobile phones in the absence of internet connections, a voice-accessible service was built. The service, RadioMarché, can be called by local farmers in their own language to retrieve this information. Community radio hosts do the same to retrieve and broadcast local offerings live on the radio. The system was developed and deployed in 2012 and during that time a number of offerings were stored [7]. In [3], we describe how the data was ported to the RDF data model and a Semantic Web-enabled version was developed. It describes the added benefit of linking market data to external data sources and using those links for visualization and improved data analyses. We also describe how this linked market data can be connected to voice-labels in multiple languages and therefore directly used by a voice-application.

The use case envisions most market information to be locally produced and retrieved and the system to be designed optimally for that local context. However, for some products, bulk buyers are interested in retrieving products from more than one region. Also, other actors such as Non-Governmental Organizations (NGOs) are interested in deriving statistics on quantities and types of products offered and sold. In those cases, market information needs to be transferred between two or more local instances of RadioMarché.

[4] `https://github.com/biktorrr/digivetkb`

3 A Platform for Semantic Web in an SMS

Our goal is to make the use of Semantic Web applications possible in areas lacking an infrastructure to support Web-like exchange of data. The intent is not to create an isolated network that mimics Semantic Web practices, but rather to develop a mechanism that is capable of transferring Semantic Web messages, i.e. retrieving and manipulating RDF data, across different kind of networks infrastructures. We want to achieve this without imposing additional network-specific operations.

3.1 SPARQL over SMS

The rural areas in West-Africa targeted by our cases only have cellular networks available for digital communication. Applications deployed in these areas can make use of SMS for machine-to-machine (M2M) transfer of data, instead of HTTP as part of a Web-based network. Noteworthy practical differences between SMS-based networks and Web-based networks are:

- SMS-based network agents are identified by phone numbers instead of URLs;
- The size of an SMS message is limited up to 160 bytes[5];
- SMS implements a one-way messaging pattern, whereas HTTP implements a request-response messaging pattern.

To transfer messages between Web-based networks and SMS-based networks, a conversion mechanism is required. This mechanism, in addition to the above-mentioned differences, must take into account these case-specific requirements:

- the number of messages sent should be as low as possible, in view of costs;
- it should be possible to run on low-resource, affordable hardware, such as a Raspberry Pi computer[6].

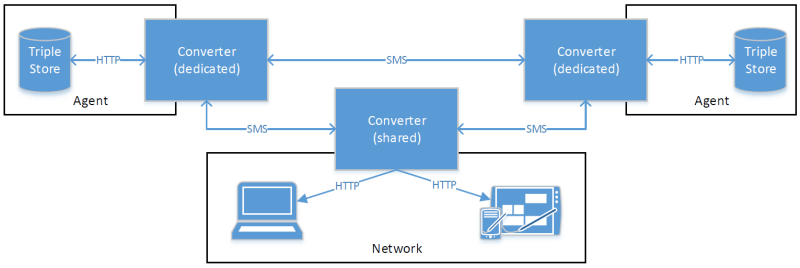
Although the costs per transferred byte are relatively high for SMS messages, still some benefits apply. For instance, the infrastructure is already in place and has a global reach which even includes some rural areas of development countries. Furthermore, the required hardware to be able to send SMS messages are affordable and widespread available.

The initial implementation of the described mechanism, which we call *SPARQL over SMS*, has been developed around SPARQL. By supporting the CONSTRUCT and INSERT/DELETE DATA query forms a basic usage of Semantic Web for machine-to-machine communication can be realized. We select this subset of SPARQL operations since these involve simple data transfer either from the client to a server or vice-versa in which the transferred data takes the form of RDF triples. SPARQL SELECT query responses take the form of result tables of arbitrary sizes and as such are harder to optimize generically.

[5] Based on the encoding used: 8-bit supports 140 characters, 7-bit up to 160 characters.
[6] `https://www.raspberrypi.org`

An overview of SPARQL over SMS is illustrated in Figure 1. The *converter* is a key component of SPARQL over SMS. A converter instance is responsible for the conversion between a standard HTTP SPARQL request and an SMS-optimized equivalent. Different options for sending and receiving SMS messages are supported by the converter, such as a GSM dongle or an online SMS service. This allows the converter to be deployed in various scenarios. For example, a converter deployed in a data-center could be used to share data with a triple store running on low-resource hardware, e.g. Kasadaka, deployed in the field.

Fig. 1. SPARQL over SMS Overview



A converter instance can be used in two modes: shared and dedicated. A dedicated converter maps a phone number directly to a particular agent. This makes it possible for the agent to both send and receive SPARQL queries. Shared converters, on the other hand, can serve multiple agents. Therefore, a phone number cannot be mapped directly to a single agent. In this case, the agents can only perform outgoing SPARQL queries but cannot be the target of incoming queries. Sending SPARQL queries typically requires an endpoint URL identifying the target receiver. To allow the targeting of an agent in an SMS-network, the converter can provide a URL representation of an arbitrary phone number. For example, the format of a SPARQL endpoint URL is: *http://{converter hostname}/agent/{phone number}/sparql*. SPARQL requests send to such an endpoint are captured by the converter and send to the phone number in the URL. On the other end, a converter receives the query and runs it on the triple store of the associated agent. The result is then send back to the initial converter which returns the result as the response to the SPARQL request in the format specified by the request.

3.2 SMS message structure and conversion

The SMS messages sent between converters follow a specific structure. Five characters of each SMS are reserved for metadata for which the basic 7-bit character set, as specific by the GSM 03.38 character set[7], is used. This includes the message type, message identifier, and position for multi-part messages. Excluding

[7] `http://www.3gpp.org/DynaReport/23038.htm`

the non-print characters there are 125 different characters left that can be used. A single character can thus express a numerical value of 1 to and including 125.

HTTP to SMS The converter will create compact SMS transfer optimized representations of HTTP SPARQL queries and results. In the case of a SPARQL query the chosen encoding routine is based on the SPARQL query form to be able to perform fine-grained optimizations. The compact representation is optionally split into multiple SMSes if it exceeds the character limitation of a single SMS. The position of each part will be indicated by the multi-part position character in the metadata. After the conversion, the compact representation is send over SMS to the phone number extracted from the endpoint URL used to send the SPARQL query to the converter.

SMS to HTTP When a converter receives a SMSes, HTTP representations are reconstructed. As different encoding routines might be used, the appropriate routine is based on the message type defined in the metadata. Decoding cannot guarantee a result exactly identical to the original message. The resulting message might thus not be syntactically equivalent, but it will be semantically equivalent. Multi-part messages are concatenated based on the multi-part position in the metadata. Although the actual order of incoming SMS messages cannot be enforced, the convention of sending the last positions first provides an indication of the number of SMSes the receiving converter can expect.

4 Research Challenges

In the generic design of our solution, converters are used to translate outgoing SPARQL queries/results to SMS messages and incoming SMS messages back to semantically equivalent SPARQL queries/results. However, to develop a workable solution, a number of challenges need to be addressed. In this section we outline the different challenges, namely: how to reduce the size of RDF data to allow for efficient transfer over SMS, issues around asynchronicity of communication and how to deal with unpredictable query result sizes.

4.1 Small RDF Data Compression

The RDF serialization format has a great effect on the size of an RDF file, and thereby on the amount of SMS messages needed to transfer the data. The costs associated with SMS messages restricts us to cases with small amounts of triples which, in the use cases described, is indeed the case (typically between 1 and 1000 triples). Still, to save costs associated with the sending SMS messages, we want to use the combination of RDF serialization and compression that is most efficient, in terms of transfer size, for such small RDF data sets.

Experiment Setup To identify the best serialization compression combination, we have set up a benchmark experiment based on RDF data sets provided by LOD Lab [12]. The RDF files provided by LOD Lab are crawled from multiple sources in the Linked Open Data Cloud[8], making it a realistic representation of real-world RDF data sets. Our benchmark test set consists of all 232,822 RDF files whose size is between 1 and 1000 triples. This large-scale experiment ensures that we test across many types of data sets, with all kinds of characteristics which might influence serialization and compression.

These files, originally in N-Triples format, were converted to various serialization formats (RDF/XML, Turtle, HDT and EXI). The RDF/XML and Turtle format are plain text serialization formats specifically designed for RDF data and are widely used. The binary format "Header, Dictionary, Triples" (HDT) is a data structure developed to compactly store and exchange RDF data, without sacrificing the ability to search through the data [4]. It claims to be useful in a wide range of cases, including sharing RDF over the web and resource efficiency on embedded devices. Efficient XML Interchange (EXI)[9] is a binary format designed to create compact representations of XML and has been proposed for efficient RDF exchange in constrained embedded networks [9]. For each format, including the original N-Triples format, the file size is recorded before and after applying gzip-compression. The default implementations of RDFLib[10], HDT[11] and EXIficient[12] have been used. Computing resource usage is not recorded since we focus only on the amount of SMS messages required based on the file size.

Table 1. Results of the LOD Lab compression experiment (N-Triples = 100%).

No. Triples	N-Triples +Gzip	RDF/XML	RDF/XML +Gzip	Turtle	Turtle +Gzip	HDT	HDT +Gzip	EXI	EXI +Gzip
1-10	**50.7**	103.8	77.0	102.0	70.3	495.5	180.1	57.5	65.9
11-20	**22.5**	62.0	27.1	50.5	24.2	122.2	47.0	23.3	24.9
21-30	**16.2**	58.2	18.5	48.7	16.3	79.5	31.1	16.5	17.5
31-40	28.3	69.1	30.9	62.1	28.6	86.5	40.7	**28.2**	29.1
41-50	9.8	51.2	10.2	42.3	**8.6**	38.1	14.8	9.3	9.7
51-60	17.2	59.2	17.5	50.1	15.9	50.5	22.8	**15.8**	16.3
61-70	11.8	58.5	12.4	42.4	**10.0**	43.0	17.7	11.1	11.6
71-80	8.8	54.8	8.5	40.9	**7.0**	31.6	11.2	7.5	7.8
81-90	6.7	52.0	6.3	40.6	**5.1**	25.4	9.1	5.8	6.0
91-100	8.1	54.9	7.6	40.4	**6.2**	26.9	9.7	6.8	7.0
101-200	8.8	62.0	8.3	39.2	**6.7**	24.7	10.1	7.6	7.9
201-300	4.8	50.8	3.6	39.0	**2.8**	13.4	4.0	3.6	3.6
301-400	4.8	51.5	3.3	37.7	**2.5**	11.4	3.3	3.0	3.1
401-500	4.4	51.5	2.9	37.4	**2.2**	10.4	2.7	2.6	2.7
501-600	5.0	53.8	3.4	38.7	**2.5**	8.9	3.0	2.9	3.0
601-700	4.1	51.0	2.5	35.9	**1.7**	8.5	2.2	2.3	2.4
701-800	4.5	51.1	2.7	36.2	**1.9**	8.1	2.1	2.4	2.4
801-900	4.4	51.1	2.6	36.4	**1.8**	7.9	1.9	2.3	2.3
901-1000	4.1	50.9	2.4	36.5	**1.7**	7.7	**1.7**	2.1	2.1

[8] http://lod-cloud.net

[9] https://www.w3.org/TR/2014/REC-exi-20140211/

[10] https://github.com/RDFLib/rdflib

[11] https://github.com/rdfhdt/hdt-java

[12] https://github.com/EXIficient/exificient

Results In Table 1, we list per format the average resulting file size in respect to the original N-Triples format. The best compression per bin are marked bold. The binary gzip, HDT and EXI formats include Base64 encoding overhead for sending over SMS. For the effect on the smallest of data sets, we first look at the results from the RDF files in the range from 1 up to 100 triples (81,492 in total). Grouped by number of triples in bins of 10. As expected, the size reduction compared to N-Triples becomes greater as the number of triples increases, because more syntactic redundancy can be taken rid of. HDT start out being bigger than the original file, this can be explained by the Header-component (metadata) that the HDT format introduces. With RDF files up to 30 triples, the gzip-compressed N-Triples format outperforms the other formats. Above 50 triples, the compressed Turtle format starts to outperform the compressed N-Triples format. For data sets between 30-60 triples, the uncompressed EXI format performs similar to compressed N-Triples and compressed Turtle. Applying gzip-compression to EXI hardly has any effect and even increases the file size in most cases.

When we consider the RDF files between 100-1000 triples we can see that gzip-compressed Turtle format results in the best compression of the data. In addition, it is more clearly visible that RDF/XML stagnates around 51% and Turtle around 36% of size compared to N-Triples. For the RDF files with 600+ triples a gzip-compressed HDT file drops below the size of EXI, at the cost of losing the ability to perform queries on the file due to the additional compression.

We conclude that for the smallest data sets ($\leq$40 triples), gzip-compressed N-Triples is preferable. For data sets between 40-1000 triples, the gzip-compressed Turtle serialization scores best. The reason N-Triples performs better than Turtle for the smallest data sets can be due to the added overhead of prefixes in the Turtle format[13]. In our implementation, we have chosen to dynamically select the appropriate serialization format, that is N-Triples or Turtle, based on the number of triples in the SPARQL result. For the two times that EXI performs best the difference is not large enough to switch from N-Triples or Turtle to EXI.

4.2 Shared Vocabulary RDF Data Compression

In section 4.1 we have focused on serialization and gzip-compression. These generic strategies consider only the syntactical representation of RDF. In order to reduce the size of RDF data even more, we tested two compression strategies focused on RDF-content specific aspects. We do this on the basis of RDF vocabularies that define reusable definitions for common properties and/or types. In order to use these strategies in SPARQL over SMS the sender and receiver must share the same set of vocabularies.

Experiment Setup To determine if RDF vocabularies can indeed be used to reduce the size of RDF data we conducted an experiment with 30 popular

[13] The used Turtle serializer adds the RDF, RDFS, XSD and XML prefixes by default.

vocabularies[14]. A subset of 3.577 RDF data sets from the previous LOD Lab experiment are subjected to dictionary-encoding and reasoning based on the RDF vocabularies. The experiment consists of three rounds with an increasing number of vocabularies (popularity top 10, top 20 and top 30). This way, the effect of using more, but less popular, vocabularies can be measured.

A single HDT file, containing a combination of vocabularies, is generated during each round of the experiment. The dictionary-encoding mechanism utilizes the Dictionary-component of this HDT file. During the encoding all the URIs in a RDF data set that also occur in one of the vocabularies are replaced with a placeholder containing the identifier generated by HDT for the specific URI.

The intended use of reasoning is to find semantic redundancies in RDF data sets, based on the knowledge contained in the vocabularies. The implemented reasoner searches for redundancies based on twelve RDFS entailment patterns[15] (all, except the first), as well as two OWL properties[16], namely *owl:SymmetricProperty* and *owl:inverseOf*. If the reasoner finds a redundant triple, it will be removed from the data set. The reasoner only considers explicitly defined triples from the RDF data set and vocabularies. Therefore, it is not guaranteed that the final result is always the smallest set of triples possible.

Table 2. Results of the shared vocabulary compression experiment (N-Triples = 100%).

No. Triples	Reasoning (10 vocab.)	Reasoning (20 vocab.)	Reasoning (30 vocab.)	D-Encoding (10 vocab.)	D-Encoding (20 vocab.)	D-Encoding (30 vocab.)
1-100	99.7	**94.0**	94.0	91.6	88.9	**88.6**
101-200	97.3	**89.3**	89.3	92.0	**91.1**	91.1
201-300	98.0	**90.0**	90.0	92.7	**91.7**	91.7
301-400	97.6	**89.6**	89.6	93.1	**92.1**	92.1
401-500	98.2	**89.2**	89.2	93.7	92.7	**92.5**
501-600	98.2	**91.3**	91.3	93.8	**93.2**	93.2
601-700	97.0	**88.9**	88.9	92.5	91.3	**91.2**
701-800	99.0	**86.6**	86.6	95.0	**94.2**	94.2
801-900	98.8	**88.5**	88.5	94.5	93.6	**93.5**
901-1000	99.9	**87.9**	87.9	94.9	**94.0**	94.0

Results The two compression strategies are measured independently during the three rounds of the experiment. Precondition for these compression strategies is that the subjected RDF data set must use one of the considered vocabularies. This makes that these compression strategies not always have effect, in terms of size reduction. The results listed in table 2 are the average resulting file sizes, grouped by number of triples in bins of 100. These results are based only on the RDF files where the compression strategies could be successfully applied (31% for reasoning and 96% for dictionary-encoding). The best results for each bin are, separately for reasoning and dictionary-encoding, marked bold.

[14] Based on the list available on `http://prefix.cc/popular/all`.

[15] `https://www.w3.org/TR/rdf11-mt/#rdfs-entailment`

[16] `https://www.w3.org/TR/owl-ref/#Property`

Reasoning based on the top 10 vocabularies has minimal effect, but using an additional 10 vocabularies increases the average size reduction across all file sizes. Using the top 30 vocabularies has no advantage over the top 20 vocabularies when using reasoning. The dictionary-encoding achieves better results based on the top 10 vocabularies than the reasoning. This is slightly improved when using the top 20 vocabularies. An additional, but minimal improvement can be obtained when using all the 30 vocabularies for dictionary-encoding. Based on these results we conclude that it is best to use the top 20 popular vocabularies when using the reasoning and dictionary-encoding compression strategies. The minimal improvement of using the top 30 for dictionary-encoding is not commensurate to the increase of processing duration and maintenance efforts introduced by the additional 10 vocabularies.

Table 3. Best results of the LOD Lab experiment before and after adding shared vocabulary compression (N-Triples = 100%).

No. Triples	Only serialization and compression	Added shared vocabulary compression
1-10	50.7	**44.2**
11-20	22.5	**18.9**
21-30	16.2	**13.6**
31-40	28.2	**23.5**
41-50	8.6	**8.0**
51-60	15.8	**8.7**
61-70	10.0	**6.0**
71-80	7.0	**6.4**
81-90	5.1	**4.4**
91-100	6.2	**5.7**
101-200	6.7	**5.7**
201-300	2.8	**2.7**
301-400	**2.5**	2.5
401-500	**2.2**	2.2
501-600	2.5	**2.4**
601-700	**1.7**	1.7
701-800	**1.9**	1.9
801-900	**1.8**	1.8
901-1000	**1.7**	1.7

We combined the top 20 vocabulary based reasoning and dictionary-encoding with the serialization and gzip-compression approach from section 4.1 to form the RDF compression method to be used in SPARQL over SMS. To evaluate this method, all the LOD Lab RDF files between 1 and 1000 triples, as described in section 4.1, are also subjected to this combined RDF compression method. The results are listed in table 3, along with the best results achieved from using only serialization and gzip-compression. It shows that using the reasoning and dictionary-encoding strategies, that are specific for use with RDF, are especially useful when compressing the smallest RDF data sets with 1 to 200 triples, in comparison with applying only the generic serialization and gzip-compression. As the data set size increases, the purely syntactical compression strategies become more efficient and gradually makes the vocabulary based compression strategies less beneficial. This is also visible in table 4, the added vocabulary based com-

pression provides a head-start in terms of the average number of triples that can be send per SMS. This head-start holds until more than 10 SMSes are send

Table 4. Average number of triples that can be send based on the number of SMSes.

Nr. of SMSes	Only serialization and compression	Added shared vocabulary compression
1	0	0
2	3	3
3	6	8
4	9	16
5	21	24
6	66	84
7	84	98
8	116	126
9	175	189
10	301	301

4.3 SPARQL Compression

The compression strategies discussed in section 4.1 and 4.2 are intended for RDF data sets. However, RDF is not the only kind of data that is transferred when using SPARQL over SMS. Queries in the form of SPARQL are also transferred. To ensure cost-efficient transfer, these SPARQL queries must also be send as small as possible.

Experiment Setup We have tested two strategies to reduce the size of SPARQL queries, namely gzip-compression and applying the final RDF compression method discussed in 4.2. To be able to use the RDF compression method for this purpose, we incorporated SPIN serialization [10] to generate RDF representations of SPARQL queries. The two strategies have been performed on 500 real-world SPARQL queries extracted from the YASGUI-client logs[17]. The RDF vocabularies of SPIN are added to the default set of vocabularies for better performance. As the two compression strategies have a binary format as results, Base64 encoding is applied on the result before measuring the file size.

Results The average file size compared to the original file size is 92,5% after applying gzip-compression and 91,9% after using the combination of SPIN and RDF compression. Even if the RDF compression on average performs better than gzip-compression, it does not in the majority of the cases. Only 44% of the time the RDF compression scored best, in the other cases either gzip-compression scored best (18%) or plain SPARQL is smaller before applying any compression (38%). The latter can be explained by the overhead of Base64 encoding, required to be able to send the binary format over SMS. Based these results, we conclude that it is best to dynamically determine if the SPARQL query can be compressed and, if so, which strategy achieves the best compression rate.

[17] `http://doc.yasgui.org/`

4.4 Blending Synchronous and Asynchronous Messaging

SPARQL exchanges follow a request-response messaging pattern. When a query is sent as a request over the network, the receiver processes the request and composes a response with the query result. A single connection is used and kept open during the transfer, making it a synchronous operation. Sending SMS messages follows a one-way messaging pattern. Each message is a standalone message that not enforces a follow-up response. Therefore, the connection is terminated after a message has been delivered, this makes the sending of SMS an asynchronous operation.

With SPARQL over SMS we want to seamlessly transfer messages from Web-based networks to SMS-based networks and vice-versa. For this purpose, we need to harmonize the two different messaging patterns. The initial implementation of SPARQL over SMS keeps connections open on the Web-based network side during the data transfer over the SMS-based network. After sending a SPARQL query over SMS, the converter waits until the corresponding query result response comes in. The query result response is of a different message type, but it can be correlated to the original request by using the message identifier for correlation.

This implementation is workable, but might not be an optimal solution considering deployment in the rural areas described in our use-cases. For example, due to unreliable or temporary loss of connectivity the response message might be available after hours or even days. It is questionable if the low-resource hardware can hold open multiple connections for that period. Even if it is capable of doing so, there is a genuine risk of a sudden power outage that will result in a loss of connections, requiring the data transfer to start-over. In our specific use-cases this would result in additional, unnecessary, costs. Additional efforts are required to create an asynchronous-supporting version of SPARQL for situations when a response is not expected within a seconds- or minutes-long time span.

4.5 Unpredictable Query Result Sizes

A simple looking SPARQL query might yield an unexpectedly large result. To be thrifty with sending SMS messages, we want to restrict the amount of SMS messages that will be send. We have considered two options to achieve this, SPARQL pagination and pagination on a SMS-level. SPARQL provides the LIMIT, OFFSET and ORDER BY keywords that can be used to implement pagination, but the SPARQL query result will not include pagination information, e.g. the total number of results available. This means it is not possible to tell whether or not all results already have been retrieved. Another issue that raises is the possibility for a triple to have a very long literal object that can span multiple SMSes. Relying only on SPARQL pagination for regulating the result size is not sufficient to regulate the number of SMSes send. Pagination on the SMS-level as alternative to the SPARQL paging would only introduce the option to decide whether or not to continue receiving the SMS messages. Since a partial result from a complete

SPARQL result cannot be created: it is the whole SPARQL result or nothing. This would also alter the way SPARQL over SMS must be used compared to the SPARQL standard, due to the addition of pagination operations which can not be ignored. With our implementation the SPARQL result is directly, after compression, send through SMS. If the result does not fit in a single SMS it will split up the message, on arbitrary points, into multiple SMSes. If reaches a hard set restriction and stops sending SMSes the receiver cannot read the message properly due to missing parts: partial results are not supported. Therefore, we consider above-limit SPARQL results as an error. An HTTP 400 bad request instead of the SPARQL result is returned to the user.

5 Practical Validation

5.1 Implementation and Integration with Kasadaka

The source code and documentation of SPARQL over SMS is openly available[18]. It is developed using the Python and Java programming languages, with easy integration with other services in mind. As a result, SPARQL over SMS is capable of running on various operating systems and devices.

For validation, we import SPARQL over SMS as a module into the Kasadaka. As described in Section 2, Kasadaka is a collection of simple, and affordable hardware: most importantly, a Raspberry Pi and a GSM dongle. It optionally has a small touch-screen to allow for icon-based interaction. All hardware components are cheap and widely available. The software on Kasadaka[19] is as much as possible free and open-source software to allow for community-development and to make it as affordable as possible. The Kasadaka project builds on information acquired from workshops in Burkina Faso, Mali, Ghana and Niger, focusing on the development of ICT services that support re-greening activities in rural, often remote areas in the Sahel. Its software includes the ClioPatria semantic web platform[20], which functions as an RDF data store for various applications on Kasadaka. It also comes with connections to a voice browser and a library to develop applications in Python. SPARQL over SMS extends Kasadaka to allow sharing of its RDF data across devices.

5.2 Evaluation in Four Scenarios

For the two use case in Section 2, we are developing services using the Kasadaka. In both cases, communication between two remotely deployed Kasadaka platforms is key. This means that the capabilities for communication over GSM allow us to test our design for SPARQL over SMS with the Kasadaka platform. Our validation setup consists of two Kasadakas with each of the triple stores loaded with both the DigiVet and RadioMarché data sets. A Web-accessible triple store

[18] `https://github.com/onnovalkering/sparql-over-sms`
[19] `https://github.com/abaart/KasaDaka`
[20] `http://cliopatria.swi-prolog.org/`

with this data is available at `http://semanticweb.cs.vu.nl/radiomarche/`. We developed two scenarios per use case which corresponds to four SPARQL queries for which we test our service and determine the amount of SMSes required both with our RDF compression method and without (in this case the plain RDF/XML output as provided by the ClioPatria triple store). To improve the shared vocabulary compression strategies, the case-specific vocabularies are added to the default SPARQL over SMS set of vocabularies.

Extending the DigiVet application Combining the DigiVet application with the SPARQL over SMS solution adds new data sharing options. With the use of two scenarios, from the perspective of the veterinarian, we demonstrate how the DigiVet application can be extended using the new features. For the first scenario we consider a veterinarian interested in retrieving the type and frequencies of animal disease symptoms occurring near Walewale, Ghana from a remote instance of DigiVet running on a Kasadaka. The SPARQL query in Listing 1.1 can be used to answer this question. Sending this query requires 3 SMS messages (in this case, there is no difference in number of triples with or without compression). The result response yields 7 triples, which sending with our solution takes 3 messages (for comparison, in the raw RDF/XML format, it would take 14 SMS messages).

Listing 1.1. Digivet SPARQL query for 1st scenario

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dv: <https://w3id.org/w4ra/digivet/>

CONSTRUCT {
  ?sym dv:occurance_count ?count
}
WHERE {
  SELECT ?sym (COUNT(?sym) as ?count) WHERE {
    ?person foaf:based_near <http://sws.geonames.org/2294174/> .
    ?person dv:has_case ?case .
    ?case dv:has_symptom ?sym
  }
  GROUP BY ?sym }
```

As a second scenario for the DigiVet case we consider the need of a veterinarian to update the animal diseases knowledge base present on a DigiVet installation. As an example, the SPARQL query in Listing 1.2 can be used to add a new disease, named “Black Leg”, with associated symptoms to the DigiVet triple store. Transferring this query requires 3 SMS messages (compared to 5 SMS messages without compression).

Listing 1.2. DigiVet SPARQL query for 2nd scenario

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dv: <https://w3id.org/w4ra/digivet/>

INSERT DATA {
    dv:black_leg a dv:Disease .
    dv:black_leg rdfs:label "Black_leg"@en .
    dv:Cow dv:canCarryDisease dv:black_leg .
    dv:Sheep dv:canCarryDisease dv:black_leg .
```

```
    dv:unwillingnessToMove dv:symptom_for_disease dv:black_leg .
    dv:rapidBreathing dv:symptom_for_disease dv:black_leg .
    dv:lameness dv:symptom_for_disease dv:black_leg .
    dv:appetiteLoss dv:symptom_for_disease dv:black_leg .
    dv:fever dv:symptom_for_disease dv:black_leg .
    dv:swellingThigh dv:symptom_for_disease dv:black_leg .}
```

Extending the RadioMarche application For the RadioMarché service, we consider two scenarios that extend its data sharing functionalities. The first scenario involves the retrieval of the current offerings, including the phone number of the advertisers, in the Mafoune and Mandiakuy regions of Mali. Using a CONSTRUCT query (Listing 1.3), this information is retrieved as a RDF graph from a RadioMarché installation. Sending the query through our solution requires 3 SMS messages (one SMS less than sending the query without any special measures). The query result consisted of 152 triples in total and could be transferred using 8 SMS messages. For the latter result, when sending the result directly from the triple store, without applying RDF Compression, a total of 121 SMS messages would have been required which shows the economic impact of the compression step.

Listing 1.3. RadioMarché SPARQL query for 1st scenario

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rm: <http://purl.org/collections/w4ra/radiomarche/>

CONSTRUCT {
    ?contact rm:contact_tel ?tel .
    ?contact rm:has_offering ?offering .
    ?offering rdfs:label ?prod_name
} WHERE {
    ?offering a rm:Offering .
    ?offering rm:has_contact ?contact .
    ?offering rm:prod_name ?prod .
    ?prod rdfs:label ?prod_name .
    ?contact rm:contact_tel ?tel .
    ?contact rm:zone ?zone .
    FILTER (?zone IN (rm:zone_Mafoune, rm:zone_Mandiakuy)) }
```

As a second RadioMarché scenario, we perform an INSERT DATA query (Listing 1.4) to add product labels in additional languages. The query will create ten new triples. Using our solution it can be send using only 3 SMS messages, half of the uncompressed number.

Listing 1.4. RadioMarché SPARQL query for 2nd scenario

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rm: <http://purl.org/collections/w4ra/radiomarche/>

INSERT DATA {
    rm:product-Beurre_de_karite rdfs:label "Shea butter"@en .
    rm:product-Beurre_de_karite rdfs:label "La manteca de karit "@es .
    rm:product-Miel_liquide rdfs:label "Honey"@en .
    rm:product-Miel_liquide rdfs:label "Miel"@es .
    rm:product-Amande_de_karite rdfs:label "Shea nuts"@en .
    rm:product-Amande_de_karite rdfs:label "Nueces de karit "@es .
    rm:product-Tamarin rdfs:label "Tamarind"@en .
    rm:product-Tamarin rdfs:label "Tamarindo"@es .
    rm:product-Graine_de_nere rdfs:label "Nere seeds"@en .
    rm:product-Graine_de_nere rdfs:label "Semillas Nere"@es . }
```

Discussion Table 5 summarizes the results for all four scenarios. It shows the number of SMSes needed to transfer the query as well as the query response. This shows that for the realistic use cases, the amount of SMSes per query is limited. We also list the total costs per query by converting current SMS rates from two providers in the location of the use case to US Dollars[21]. This shows that, although expensive, the specific use case could potentially be made economically viable. The number of SMS messages required to transfer the SPARQL results confirm to the estimations in Table 4.

Table 5. Summary of the four validation scenarios.

Scenario	Location	Query type	Request Size in nr. of SMS	Request est. cost (USD)	Response Size in nr. of SMS	Response est. cost (USD)
Digivet Sc.1	Ghana	CONSTRUCT	3	0.042	3	0.042
Digivet Sc.2	Ghana	INSERT	3	0.042	n.a.	
RadioMarché Sc.1	Mali	CONSTRUCT	3	0.105	8	0.280
RadioMarché Sc.2	Mali	INSERT	3	0.105	n.a.	

6 Related Work

SMS as a data channel has been proposed in other ICT for Development (ICT4D) cases, for example in [8]. A number of Social Network Services such as Twitter, Facebook as well as the Google search engine allow for accessing those services through SMS[22]. Mostly, this deals with machine-to-human interaction and not, as in our case for machine-to-machine interaction.

Related work in semantic data exchange in low-resource network environments includes the Entity Registry System (ERS) [1], an open-source entity registry specifically designed for environments with ad-hoc and/or unreliable network connectivity. It allows for Linked Data without using the centralised components that make up the Web infrastructure. ERS has mechanisms to deal with interval-based network connectivity (e.g. a mobile truck that functions as an access point) and is resistant against packet loss. DakNet provides similar solutions where ad-hoc wireless networks are combined with asynchronous networking, also including mobile access points [11]. Whereas these solutions also implement Web-like datasharing without Web infrastructure, they focus mainly on local networks and rely on the availability of partial internet connectivity.

Another way of transferring data without Internet is through so-called Sneakernets, where data is exchanged by physically moving removable media or hard

[21] For Mali, we assume an average cost of 20CFA=0.035USD per SMS `http://www.orangemali.com/2/particuliers/28/34/les-prepayes-113.html`. For Ghana, we assume 0.055GH=0.014USD per SMS `http://support.vodafone.com.gh/customer/portal/articles/1823814-sms-`

[22] `http://www.digitaltrends.com/mobile/sms-your-way-back-to-the-web/`, accessed Apr 2016

disks. For large-scale non-immediate data transfer, this is a viable solution [5] which can be combined with solutions such as the one presented in this paper[23]

In this paper, we focus on Semantic data exchange using the SPARQL protocol. There are other opportunities for accessing RDF data over a network. Two examples are simple URI dereferencing and the use of Linked Data Fragments [14]. Compared to these methods, accessing RDF using SPARQL typically takes more computing resources on the client and server devices itself, but bandwidth can be limited. For our specific cases, saving bandwidth is a key issue, which is why we use SPARQL. It is interesting to further investigate the trade-off between computational and networking resources in these specific ICT4D cases.

7 Conclusions

With SPARQL over SMS we have shown that data sharing according to Semantic Web practices is possible even in areas without a Web infrastructure. We developed a conversion module that translates SPARQL over HTTP requests to SMS messages and decodes these messages at the other end. This is an example of downscaling the Semantic Web to the infrastructure in place, in our case SMS. Enabling the Kasadaka platform to not only perform machine-to-human communication but also machine-to-machine communications adds new possibilities for applications our solution integrates easily with conventional Web-based knowledge sharing since it does not create an isolated SMS-network but rather presents a conversion mechanism.

We investigated a number of challenges around porting SPARQL data exchange using SMS. Several RDF compression strategies are evaluated based on real-world small data sets, leading us to a dynamic compression method that combines the generic serialization and text compression strategies with the Semantic Web specific shared vocabulary compression strategies. We have shown the viability of sending small RDF data sets using SPARQL over SMS and elaborate this in four scenarios from two realistic use cases currently under development. Future work consist of further development and deployment of Kasadaka-based solutions which include SPARQL over SMS in the field and designing longer term evaluations for these and new ICT4D use cases.

The current SPARQL over SMS has several limitations and opportunities for improvement. First, the reasoning that is used to eliminate semantic redundancies is based on a limited number of RDFS and OWL patterns and is restricted in terms of the search depth. Second, the SMS transfer mechanism is not yet fitted to properly deal with unexpected faults or partial transfers. We here are looking at methods from systems such as the aforementioned ERS. Furthermore, not yet all SPARQL operations are supported. To achieve full compatibility, these will have to be implemented. Lastly, the Kasadaka implementation used to send and receive SMSes only supports 8-bit SMSes (140 characters). If 7-bit SMSes (160

[23] As a side-note, the humorously suggested "IP over Avian carriers" protocol describes Internet access using pigeons carrying USB sticks or flash cards. `https://tools.ietf.org/html/rfc1149`

character) could be used, additional data can be transferred per SMS. The intent is to conduct further tests, by deploying SPARQL over SMS in the field, to identify the effects of these limitations and to validate the solution in real-world conditions. These field tests will also include closer research into the economic viability of these solutions as discussed in [7] and look at for example, integrating mobile-based payment plans.

Although SPARQL over SMS is developed based on ICT4D cases, it is also applicable to other low-bandwidth cases. For example in the context of disaster-management or the Internet of Things. Since the technologies used are platform independent, SPARQL over SMS can be ported to other platforms that require SPARQL over short-message transfer networks.

Finally, in this paper, we present a specific case for decoupling the principles and practices of the Semantic Web from the underlying technical implementation. This shows that these principles are still valid and valuable without the availability of Internet and the Web infrastructure. The more non-Web type of networks are supported, the greater the reach of Semantic Web will be, as knowledge can be send across multiple types of networks in a standardized fashion.

References

1. Marat Charlaganov, Philippe Cudré-Mauroux, Cristian Dinu, Christophe Guéret, Martin Grund, and Teodor Macicas. The entity registry system: Implementing 5-star linked data without the web. *arXiv preprint arXiv:1308.3357*, 2013.
2. Victor de Boer, Pieter De Leenheer, Anna Bon, Nana Baah Gyan, Chris van Aart, Christophe Guéret, Wendelien Tuyp, Stephane Boyera, Mary Allen, and Hans Akkermans. Radiomarché: Distributed voice-and web-interfaced market information systems under rural conditions. In *Proceedings of the 24th international conference on Advanced Information Systems Engineering (CAiSE)*, pages 518–532. Springer, 2012.
3. Victor de Boer, Nana Baah Gyan, Anna Bon, Wendelien Tuyp, Chris van Aart, and Hans Akkermans. A dialogue with linked data: Voice-based access to market data in the sahel. *Semantic Web*, 2013.
4. Javier D Fernández, Miguel A Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. Binary rdf representation for publication and exchange (hdt). *Web Semantics: Science, Services and Agents on the World Wide Web*, 19:22–41, 2013.
5. Jim Gray, Wyman Chong, Tom Barclay, Alex Szalay, and Jan Vandenberg. Terascale sneakernet: Using inexpensive disks for backup, archiving, and data exchange. *arXiv preprint cs/0208011*, 2002.
6. Christophe Guéret, Stefan Schlobach, Victor De Boer, Anna Bon, and Hans Akkermans. Is data sharing the privilege of a few? bringing linked data to those without the web. *ISWC2011 Outrageous ideas Track, Best Paper award*, pages 1–4, 2011.
7. Nana Baah Gyan. *The Web, Speech Technologies and Rural Development in West Africa An ICT4D Approach.* PhD thesis, Vrije Universiteit Amsterdam, 3 2016.
8. Richard Heeks. Ict4d 2.0: The next phase of applying ict for international development. *Computer*, 41(6):26–33, 2008.
9. Sebastian Käbisch, Daniel Peintner, and Darko Anicic. *The Semantic Web. Latest Advances and New Domains: 12th European Semantic Web Conference, ESWC*

2015, Portoroz, Slovenia, May 31 – June 4, 2015. Proceedings, chapter Standardized and Efficient RDF Encoding for Constrained Embedded Networks, pages 437–452. Springer International Publishing, Cham, 2015.
10. Holger Knublauch, James A Hendler, and Kingsley Idehen. Spin-overview and motivation. *W3C Member Submission*, 22, 2011.
11. Alex (Sandy) Pentland, Richard Fletcher, and Amir Hasson. Daknet: Rethinking connectivity in developing nations. *Computer*, 37(1):78–83, January 2004.
12. Laurens Rietveld, Wouter Beek, and Stefan Schlobach. Lod lab: Experiments at lod scale. In *The Semantic Web-ISWC 2015*, pages 339–355. Springer, 2015.
13. Onno Valkering, Victor de Boer, Stefan Schlobach, Gossa Lô, and Romy Blankendaal. The semantic web in an sms. 2016.
14. Ruben Verborgh, Miel Vander Sande, Pieter Colpaert, Sam Coppens, Erik Mannens, and Rik Van de Walle. Web-scale querying through linked data fragments. In *LDOW*, 2014.