# SPARQL Query Execution in Networks of Web Browsers

Arnaud Grall[1,2], Hala Skaf-Molli[1], and Pascal Molli[1]

[1] LS2N – University of Nantes, France
{arnaud.grall,hala.skaf, pascal.molli}@univ-nantes.fr
[2] GFI Informatique - IS/CIE, Nantes, France {arnaud.grall@gfi.fr}

**Abstract.** Decentralizing the Web means that users gain the ability to store their data wherever they want, users can store their data in their web browsers. Web browsers allow decentralized applications to be deployed in one click and enable interaction with end-users. However, querying data stored in a large network of web browsers remains challenging. A network of web browsers gathers a very large number of browsers hosting small data. The network is highly dynamic, many web browsers are now running on mobile devices with limited resources, this raises issues on energy consumption. In this paper, we propose Snob, a query execution model for SPARQL query over RDF data hosted in a network of browsers. The execution of a query in a browser is similar to the execution of a federated SPARQL query over remote data sources. In Snob, direct neighbours in the network are the data sources and results received from neighbours are stored locally as intermediate results. As data sources are renewed every shuffling, the query continues to progress and could produce new results after each shuffling without congesting the network. To speed up query execution, browsers processing similar queries are connected through a semantic overlay network. Experimentation shows that the number of answers produced by queries grows with the number of executed queries in the network while the number of exchanged messages is always bounded per user.

**Keywords:** Decentralized Data Management, Decentralized Applications, SPARQL Query Processing, Web Browsers

## 1 Introduction

Decentralizing the applications and the data is a powerful paradigm to provide more control to users on their digital life [3]. A decentralized application runs on resources owned by multiple authorities and provides a service even in presence of failures of any authority. Data are decentralized if authorities are free to store their data wherever they want. In this paper, we focus on how a decentralized

[3] This work will be published as part of the book "Emerging Topics in Semantic Technologies. ISWC 2018 Satellite Events. E. Demidova, A.J. Zaveri, E. Simperl (Eds.), ISBN: 978-3-89838-736-1, 2018, AKA Verlag Berlin."

application running in web browsers is able to provide a SPARQL service over data stored in browsers.

We focus on browsers as they are certainly the most deployed programming environment in the world. Decentralized applications can be deployed in one-click, they are in direct contact with end-users, they are able to capture their location, their history of browsing or their perceptions of the real world. Browsers already integrate APIs to store data locally and are often used by web applications to provide off-line functionalities. However, *querying data over a large-scale network of browsers is challenging.*

Many works address the problems of data management in Peer-to-Peer networks [20]. However, decentralizing data in browsers raises some particular issues. First, a network of browsers gathers a very large number of browsers with few data hosted in browsers. It raises the problem of completeness when executing a query over a very large number of relevant sources as pointed out in [11]. Second, a large number of browsers are now running on mobile phones with LTE connections. The dynamicity of such networks is very high compared to existing P2P networks. Saving bandwidth and battery are clearly two major issues that are not considered by many existing P2P data management systems. This raises the problem of processing queries with a fair usage of resources. Third, connections between browsers rely on the WebRTC[4] standard that does not have routing. As a browser has no address, indexing resources is useless because a browser hosting relevant data cannot be contacted without flooding the network. This raises the problem of processing queries when only direct neighbours can be contacted.

In this paper, we propose Snob, an approach for executing SPARQL queries over a network of browsersWe assume that browsers are connected in an unstructured network based on periodic peer-sampling [18,22], *i.e.*, each browser is connected to a bounded random subset of the network that is renewed periodically during a shuffling. A query running in a browser can be seen as a federated SPARQL query where neighbours are the data sources. As data sources are renewed every shuffling, the query continues to progress after each shuffling. This ensures that the number of messages exchanged with neighbours is constant and every query makes progress at each shuffling. However, as progression can be slow, we build a second overlay network where browsers are connected to a fixed number of browsers that process similar queries. The new semantic overlay allows browsers to share common intermediate results.

This paper presents the following contributions:

1. Snob a SPARQL query execution model over data hosted in a network of browsers. Snob ensures a fair usage of resources by communicating only with direct neighbours at each shuffling.
2. A semantic overlay network based on query containment.
3. Experimentation shows that the completeness of queries grows with the number of running queries in the network. The semantic overlay network is able to speed up completeness when fewer queries are running simultaneously.

[4] `https://www.w3.org/TR/webrtc/`

The section 2 describes the related works. Section 3 describes our proposal. Section 4 presents preliminaries experimental results. Section 5 concludes contributions and presents future works.

## 2 Related Works

In our previous work [16], we presented how a distributed semantic web application can be deployed in a network of browsers. We detailed some use-cases and pointed some open issues. In this paper, we will go further, we will present how it is possible to execute SPARQL queries on data hosted in browsers.

The Solid project [15] promotes the vision of a decentralized web for social Web applications. In Solid, a user stores their data in a personal online datastore (pod). Therefore, application developers can write application relying on the API of a pod. When an application is deployed in the user browser, the application requests the location of the pod to access data. Consequently, the user controls where their data are stored. However, Solid does not describe how to run a query over a large-scale network of pods.

WebDamLog [3] promotes the vision of a decentralized social network. WebDamLog allows several sites to collaborate thanks to an original distributed datalog engine. WebDamLog relies on the delegation of datalog rules to connected sites to make local datalog program runnable. Compared to Solid, WebDamLog offers a way to write distributed applications, but is not designed to scale to very large networks.

Mastodon [5] or Diaspora [6] are representative of distributed social networks [5]. Compared to Solid, a pod can gather thousands of users. As anyone can create a new pod and join the network of existing pods, the system is "decentralized". However, they do not allow to run expressive queries over the network of pods.

The term "decentralized applications" is also used in the crypto-currencies communities such as bitcoins or ethereum [7]. Such applications use blockchains to store data and cryptographic tokens to store values. As some blockchains rely on public P2P networks that anyone can join, the application is called "decentralized". In this case, users do not really choose where data are stored. In this paper, we focus on an approach where data placement and control are managed by users at run-time.

A network of browsers is strongly related to P2P data management [20] for RDF data [21]. Many systems rely on Distributed Hash Tables (DHT) such as P-Grid [1], GridVine [2]. DHTs are able to deliver good performances for some classes of queries such as range queries, however, they hardly process complex join queries and the maintenance of the structure is costly under high dynamicity. Many systems also rely on unstructured networks. Such approach just maintains a neighbourhood for each site and better resists to churn. As a participant does not know where data are located, she floods the network with her query. However,

[5] `https://joinmastodon.org/`
[6] `https://diasporafoundation.org/`
[7] `https://www.ethereum.org/`

this approach does not scale and hardly delivers complete results. Flooding can be avoided with super-peer maintaining routing indices as in Edutella [19]. Having super-peers in a network of browsers is possible but they represent a point of failure which is a strong limitation to massive deployment of distributed applications in browsers. Flooding can also be reduced with spanning trees as in sensor networks [7]. A spanning tree reduces the flooding to the number of nodes in the network. However, spanning trees are costly to maintain on large networks with heavy churn. The network traffic of Flooding can be significantly reduced using adapted replication strategies and random walks [14]. Flooding can be limited by using multiple overlays as in Semantic Overlay Network (SON) [6,8]. Participants are clustered in communities according to their common interests. Queries are routed to the right community to be executed. SON restricts the number of sources for a query. In this paper, we follow the SON approach to restrict the search space for queries.

## 3 The Snob Approach

In this paper, we propose Snob an approach to execute SPARQL queries over a network of browsers. Snob relies on three key ideas:

1. To address issues of dynamicity, we build an unstructured network based on periodic peer-sampling [18,22], *i.e.*, each browser shuffles its local view with the local view of its neighbors. Compared to structured networks [9], unstructured network tolerates high churn and high expressiveness of queries [20][chapter 16].
2. Basic approaches for executing queries on unstructured networks rely on flooding. However, flooding is incompatible with a fair usage of resources. To overcome this problem, in Snob a browser evaluates its queries as federated queries using only its direct neighbours as remote data sources. It waits for the next shuffling that will bring new data sources. Such approach regulates the network traffic, *i.e.*, a browser can only send a number of messages bounded to its number of direct neighbours per shuffling.
3. To speed up query execution, we promote the sharing of intermediate results. First, sharing intermediate results replicates and aggregates data. Consequently, the probability that a query communicates with relevant intermediate results is improved [14]. Second, we build semantic overlay networks [6] that select browsers processing similar queries. Therefore, once a browser encountered a browser with a similar profile, the selected browser remains in its neighbourhood, improving the search capacity of this browser per shuffling.

### 3.1 Network model

We consider a network of web browsers based on WebRTC. A WebRTC node has no address and WebRTC has no routing. So a node cannot send messages to a particular node in the network. However, it can send messages to direct

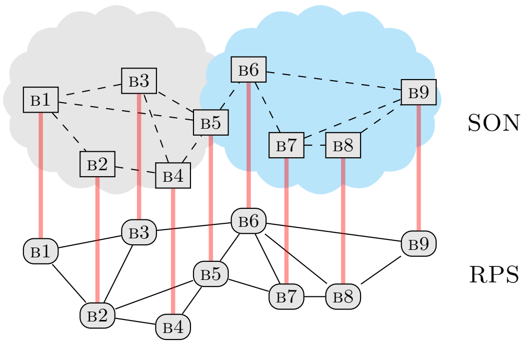


Fig. 1: B1-B9 represents browsers. The RPS network connects clients in a random graph. The semantic overlay network connects the same browsers (red link) according to their profiles. B1-B4 performs queries on Diseasome. B6-B9 perform queries on LinkedMDB. B5 performs queries on both.

neighbours. This strong constraint prevents browsers from dereferencing hosted RDF data.

We build two overlay networks as proposed in [4, 10].

1. a *Random Peer Sampling* (RPS) overlay network that maintains the membership among connected profiles. We rely on the Cyclon protocol [22] to maintain the network. Each node maintains a partial view on the entire network. The *view* contains a random subset of network nodes. Periodically, a node performs a shuffling *i.e.*, selects the oldest node from its view and they exchange parts of their views. This view is used to bootstrap and maintain the *semantic overlay network*.
2. a *Semantic Overlay Network* (SON) builds on top of RPS, it clusters browsers according to their profile. Each browser maintains a second view, this view contains the *k-best* neighbours according to the similarity of their profile with the browser profile. The maintenance of *k-best* neighbours is performed at RPS exchange time. To minimize the overhead of shuffling: (1) the profile information has to be as small as possible, (2) the similarity metric has to be computed quickly in order to prevent slowing down the query execution.

Figure 1 shows browsers, browsers $B1 - B4$ has data Diseasome, $B6 - B9$ has data on LinkedMDB and $B5$ has data on both. The RPS network ensures that all browsers are connected through a random graph, browsers profiles make the clustered network converging towards two communities. $B1 - B4$ will be highly connected because they have data over Diseasome, while $B6 - B9$ will be grouped together due their interest in LinkedMDB. $B5$ could be connected to both communities because it hosts data from both datasets.

**Algorithm 1:** Ranking Algorithm

```
Input: $P$: Profile, $Profiles$: set of Profiles, $k$: number of top ranked profiles
Output: TopK ranked profiles
1  rank($P, Profiles$):
2      $sortedProfile$ = sort($P$, $profiles$, compare)
3      return SelectTop$k$($k$, sortedProfiles)
4  compare($P$, $P_1 \in Profiles$, $P_2 \in Profile$):
5      return Scoring($P, P_2$) − Scoring($P, P_1$)
6  Scoring($P_1 \in Profiles$, $P_2 \in Profile$):
7      $score = 0$
8      foreach $TP_1 \in P_1$ do
9          foreach $TP_2 \in P_2$ do
10             if $TP_1 \equiv TP_2$ then
11                 $score = score + Weight(\equiv)$
12             end
13             else if $TP_1 \sqsubseteq TP_2$ then
14                 $score = score + Weight(\sqsubseteq)$
15             end
16             else if $TP_1 \sqsupseteq TP_2$ then
17                 $score = score + Weight(\sqsupseteq)$
18             end
19         end
20     end
21     return $score$
```

### 3.2 Browser profile

The Semantic Overlay Network clusters browsers with similar profiles. A profile is defined as *a set of triple patterns of queries that are executed by the browser.* Profiles are used for ranking browsers. We use triple patterns containment to define similarities among profiles

**Definition 1 (Triple Pattern Query Containment).** *Let $TP(D)$ denote the result of execution of query $TP$ over an RDF dataset $D$. Let $TP_1$ and $TP_2$ be two triple pattern queries. We say that $TP_1$ is contained in $TP_2$, denoted by $TP_1 \sqsubseteq TP_2$, if for any RDF dataset $D$, $TP_1(D) \subseteq TP_2(D)$. We say that $TP_1$ is equivalent to $TP_2$ denoted $TP_1 \equiv TP_2$ iff $TP_1 \sqsubseteq TP_2$ and $TP_2 \sqsubseteq TP_1$.*

Testing triple pattern containment meant finding substitution of variables in the triple patterns [17]. The complexity to find a containment is $O(1)$. For example, a containment between $TP_1$ and $TP_2$, $TP_1 \sqsubseteq TP_2$, requires to check if $TP_1$ imposes at least the same restrictions as $TP_2$ on the subject, predicate, and object positions, *i.e.*, $TP_1$ and $TP_2$ should have at most the same number of variables.

To rank neighbours, we assign weights to the containment relationship using the following order:

$$(TP_1 \equiv TP_2) \succeq (TP_1 \sqsubseteq TP_2) \succ (TP_1 \sqsupseteq TP_2) \qquad (1)$$

We assume that equivalence has the highest weight, *i.e.* the browser is considered as the best relevant data source. We assign the weight $\infty$ to $\equiv$, 2 for $\sqsubseteq$ and 1 for $\sqsupseteq$, therefore, the profile with at least one equivalence will always be preferred to other profiles.

The algorithm 1 uses the order defined in (1) to keep the $k$ best-ranked profiles of neighbours.

The *compare* function (lines 4-5) compares two different profiles by using the scoring function. The scoring function (lines 6-21) assigns weights based on the order defined in (1). Finally, the ranking function returns the top $k$ sorted profiles.

### 3.3 Query Execution

In Snob, a browser hosts a small dataset, *e.g.* thousands of triples. At a given time, a browser $B_i$ has a fixed number of neighbours: $k$ neighbours in the random peer sampling layer, and $l$ neighbours in the semantic overlay network. Typical values of $k$ and $l$ are logarithmic to the size of the whole network. Each browser exposes to its neighbours a triple pattern interface. The browser is able to process an incoming triple pattern query and return results to the triple pattern query originator. Incoming triple pattern queries are processed over local data and intermediate results. Therefore, a browser can share local data and intermediate results of running queries.

Suppose that the browser $B_i$ processes the query $Q_i$. $Q_i$ is processed *like* a federated query over its $k + l$ neighbors considered as data sources. However, there are few differences with federated query processing:

1. The federation is incomplete, *i.e.*, the federation will change at the next shuffling. Therefore, we need to keep intermediate results for continuing processing when new sources are discovered.
2. Shuffling can bring already visited data sources. However, as intermediate results may have been progressed, queries have to be re-executed.

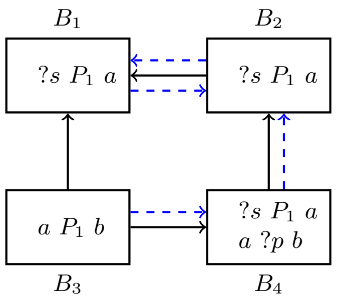
For instance, the browser $B_i$ executes $Q_i$ after each periodic shuffling window, then waits for the next shuffling for re-executing $Q_i$. During the shuffling window, the browser only answers to triple pattern queries from direct neighbours.

The Snob approach has several properties:

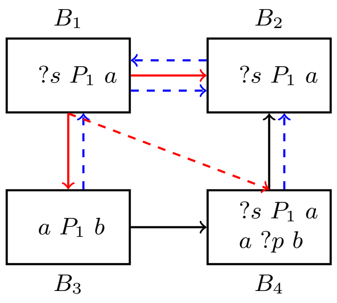
**Constant number of messages** We suppose a round corresponds to the shuffling window. Suppose the browser $B_i$ processes one query $Q_i$. In one round, $(k + l)$ messages are sent to neighbours with the list of triple patterns of the query $Q_i$. The browser waits for answers and then process its query $Q_i$ with new intermediate results. So the number of messages exchanged in the whole network of $n$ participants is bounded to $(k + l) * n$ per round. This prevents congestion of the network, saves bandwidth and energy.

**Continuous progress** Thanks to shuffling properties, the RPS layer will bring periodically to $B_i$ uniform sampling of the network, and potentially nodes that $B_i$ has never met before. So the query will progress. As we know the size of the network [8], $B_i$ is able to estimate the ratio of explored nodes with respect to the total size of the network and decides to terminate or continue the query execution. This is a form of Pay-as-you-go query processing. This approach can be compared to Link Traversal approach [12] except that we do not follow links to access data, we let the shuffling brings data to nodes. We also do not have the problem of reachability as peer sampling prevents any partition within the network.

**Automatic Clustering** As the query progress, $B_i$ meets other browsers with similar profiles, they automatically create a a community that crawls the data space thanks to shuffling. Consequently, the more browsers execute similar queries, the more they improve their crawling capacity.



(a) Network state after the first shuffling



(b) Network state after the second shuffling

| Browsers | Data |
|---|---|
| $B_1$ | $a\ P_1\ a$ |
| $B_2$ | $b\ P_1\ a$ |
| | $c\ P_1\ a$ |
| $B_3$ | $a\ P_2\ a$ |
| | $d\ P_1\ a$ |
| $B_4$ | $a\ P_1\ b$ |
| | $e\ P_1\ a$ |

(c) Initial Browsers Data

Fig. 2: Partial Snob network centred on $B_1$. Solid lines represent clients in RPS view. Dashed lines represent clients in SON view. Each client has a profile with a list of triple patterns representing its running queries.

To illustrate Snob query execution model, consider a subset of a network of browsers with a random peer sampling network (black link) and a semantic overlay network (blue link) in Fig. 2a. In this example, each browser $B_i$ has at least one query to execute. For simplicity, the example is centred on the browser $B_1$ for a period of two shuffling windows. In Fig. 2a, $B_1$ has only one neighbor $B_2$ in the SON and no neighbor in the RPS. $B_1$ executes the query $Q_1$ which has one triple pattern: $TP_1 : ?s\ P_1\ a$, therefore the profile of $B_1$ contains $TP_1$.

$B_1$ sends $TP_1$ to $B_2$. $B_2$ answers with the triples matching $TP_1$ in its local data store (cf. 2c). Once $B_1$ receives answers, $B_1$ inserts intermediate results into its local data store and executes $Q_1$ which will result in two triples. Then, $B_1$ suspends $Q_1$ execution until the next shuffling. During this time, $B_1$ only answers neighbours triple pattern queries.

[8] We use an estimation of the size of the network as proposed in [18]

After the second shuffling, in Fig 2b, $B_1$ has now more neighbors, $B_2$, $B_3$ and $B_4$. In red new the established links, solid links are for RPS neighbours and dashed ones for SON. $B_1$ sends to all neighbors $TP_1$. Neighbours answer $B_1$ request with matching triples in their local store and intermediate results collected from their neighbours. $B_1$ stores received intermediate results and re-executes $Q_1$. At this state $B_1$ has explored all browsers and the execution of $Q_1$ will produce five triples, which is the complete answers for $Q_1$ with respect to the data sources available in this network.

Other browsers executing queries follow the same cycles as $B_1$. In Fig. 2a, $B_2$ and $B_4$ also execute queries with the same triple pattern as browser $B_1$, *i.e*, $?s\ P_1\ a$. De facto, in a bigger network, browsers $B_1$, $B_2$ and $B_4$ will share their crawling capacity *i.e*, they will rely on their respective intermediate results and their different exploration of the network to answer faster their queries.

## 4 Experimental Study

The goal of the experimental study is to evaluate the effectiveness of Snob. We expect to see that the SON of Snob overcomes the RPS in terms of the number of produced answers and the number of messages is constant between rounds and always bounded to the number of edges in the network.

Table 1: Experimental Parameters

| Parameter | Values |
|---|---|
| Number of Clients | 196 |
| Executing queries | 196 - 98 - 49 |
| RPS view ($k$) | 10 - 5 |
| SON view ($l$) | 5 |
| Warmup | 10 shuffling windows |
| Shuffling Time | 10 minutes |

| Parameter | Values |
|---|---|
| Data sets | LinkedMDB - Diseasome |
| Queries | 100 queries over LinkedMDB<br>96 queries over Diseasome |
| Ranking weights | $TP_1 \equiv TP_2$: $+\infty$<br>$TP_1 \sqsubseteq TP_2$: 2<br>$TP_1 \sqsupseteq TP_2$: 1 |

### 4.1 Experimental setup

We use RDFStore-js [13] as a data store and a query engine. Each browser hosts a RDFStore-js extended with the Snob model presented in section 3. Snob source code is available at [9]. A Snob client can be a NodeJs client or pure JavaScript client running in a browser. For our experiments, we focused on a NodeJs client and completely abstracted the WebRTC communication layer. All experiments were executed on one machine with Intel(R) Xeon(R) CPU E5-2680v2@2.80GHz with 40 cores and 130GB RAM. NodeJs version is set to 8.11.1 (LTS).

[9] `https://github.com/folkvir/webrtc-dequenpeda`

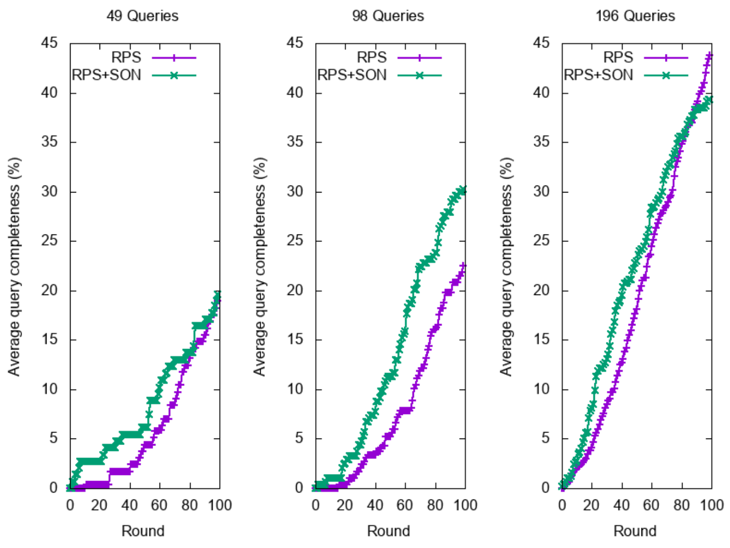


Fig. 3: Average query completeness by round for different queries number with RPS, RPS+SON network configuration. Left to right respectively 49 (quarter), 98 (half) and 196 (all) queries.

We use the real datasets: Diseasome[10] and Linked Movies Database[11] (LinkedMDB). We generate 100 queries per dataset using PATH and STAR shaped templates with two to eight triple patterns that are instantiated with random values from the dataset. We removed the queries that caused the query engine to abort execution. This results in 100 queries from LinkedMDB and 96 queries from Diseasome. For these queries, we extracted triple patterns, each triple pattern of the query is executed as a SPARQL construct query and produces a fragment that we split into two sub-fragments. Sub-fragments are randomly distributed across the clients, each client hosts at least one fragment.

We set up a network of 196 clients. Each client can execute at most one query, we vary the number of queries executed in the network to observe the impact on the network clustering. We choose one query per client (all), half of the clients executing queries (half) and a quarter of clients executing queries (quarter). The quarter is a subset of the half-selected queries. Table 1 presents the different parameters used in the experiment. We vary the value of parameters according to the objective of the experimentations as explained in the following sections. The shuffle time is fixed to 10 minutes for all experiments.

**Evaluation Metrics:** *i*) *Continuous Progress:* is the number of completed queries. *ii*) *Constant Messages number :* is the number of messages produced

[10] `https://old.datahub.io/dataset/fu-berlin-diseasome`
[11] `http://data.linkedmdb.org/`

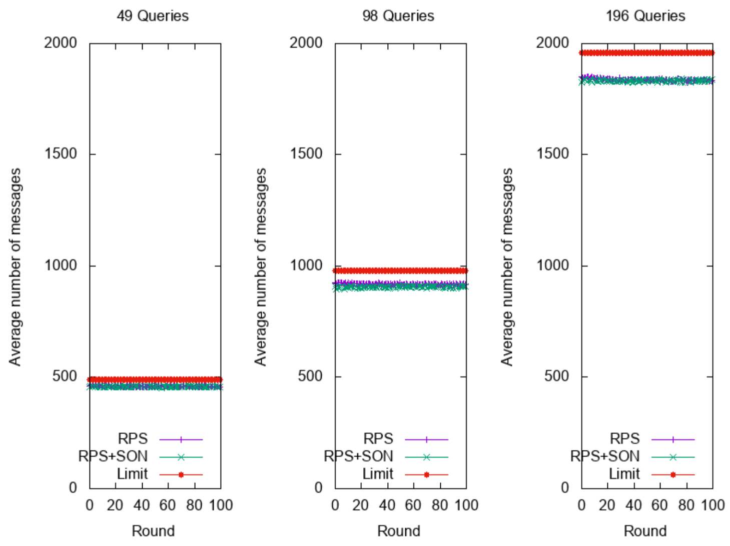


Fig. 4: Average number of messages by round for different queries number with RPS, RPS+SON network configuration. Left to right respectively 49 (quarter), 98 (half) and 196 (all) queries.

by round. *iii*) *Automatic Clustering:* is the clustering of clients running similar queries.

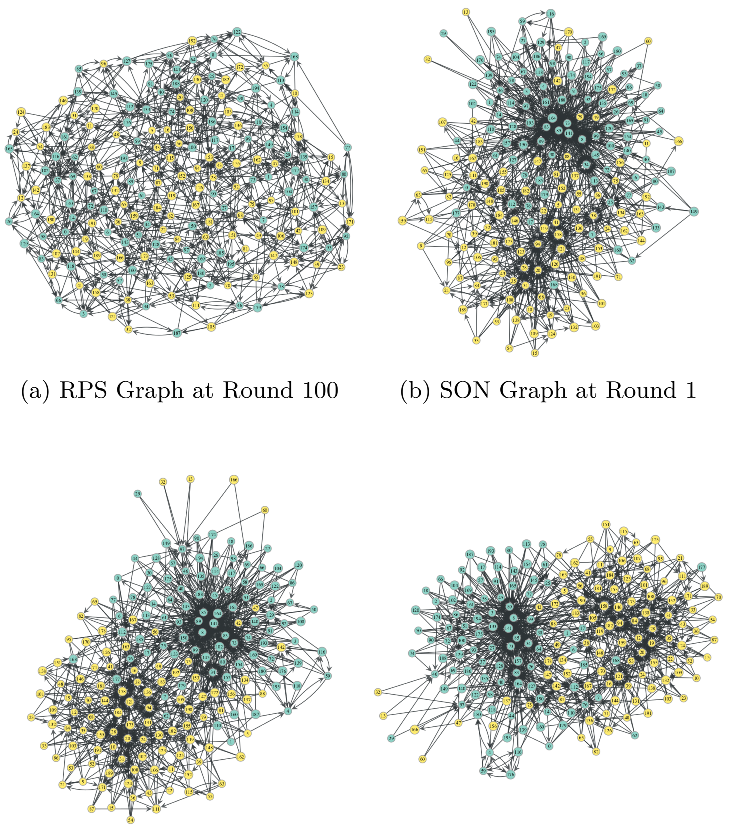
Results presented for all metrics are the average of three successive executions of 100 rounds. A round corresponds to a query execution after a periodic shuffle where the shuffling time is set to 10 minutes.

## 4.2 Experimental Results

**Continuous progression** To study the impact of the number of queries in the network on the progress of queries execution, we used different number of queries.

Fig. 3 presents the completeness of query answers per round for three different configurations. Firstly for 49 queries (quarter), secondly for 98 queries (half) and finally for 196 queries (all). A round corresponds to a query execution after a periodic shuffle where the shuffling time is set to 10 minutes. As we can see, only RPS+SON provides better query completeness.

For the **Half** and **Quarter** at round 100, the RPS produces respectively 22.48% and 18.91% of complete answers. Compared to the RPS, the RPS+SON produces respectively 30.28% and 19.60%. We notice that the RPS+SON produce a better completeness rather than just relying on the RPS. However, the gap between RPS and RPS+SON is drastically reduced for 196 queries. This is because while executing queries, intermediate results are materialized, when the number of queries increases, the number of intermediate results is naturally



(a) RPS Graph at Round 100 (b) SON Graph at Round 1

(c) SON Graph at Round 50 (d) SON Graph at Round 100

Fig. 5: The evolution of the RPS and SON networks with 196 queries (all). LinkedMDB in Yellow and Diseasome in Blue

increased. Thus, as intermediate results can be used to answer triple patterns queries, the replication of intermediate results helps to find relevant fragments faster. This is why we can sometimes notice a curve crossing between the RPS and RPS+SON curves.

**Constant message number** In all configuration, the number of messages is always bounded to $(k + l) * |queries|$ as only one triple pattern query is sent to a neighbour. For example, in Fig. 4 *all* configuration (196 queries), the average number of message at round 100 is 1834 bounded to 1960 ($(k + l) = 10$ and $|queries| = 196$). As expected, we can notice for each configuration that the average number of results is always smaller than the limit. Note here that we do not take into account messages produced by RPS and SON for their establishment and their maintenance. Only Snob messages are taken into account.

**Automatic clustering** In Fig. 5, the directed graph represents the RPS and the clustering overlay network where a node represents a client in the network and an edge represents the connection between clients. For example, an edge 1 $\rightarrow$ 2 means that the *client 1* has the *client 2* in its SON view. Fig. 5a shows clearly that the RPS network groups nodes randomly. However, Figures 5b, 5c and 5d demonstrate that Snob SON network is able to cluster clients according to their queries similarities, the clusters are clearly distinguished at round 100 where a greater value of rounds promotes the clustering.

## 5 Conclusion

In this paper, we described how a decentralized application running in web browsers is able to provide a SPARQL service over data stored in browsers. We proposed Snob, a query execution model for SPARQL query over RDF data hosted in a network of browsers. The execution of a query in a browser is similar to the execution of a federated SPARQL query over remote data sources. In Snob, direct neighbours in the network are the data sources and results received from neighbours are stored locally as intermediate results. As data sources are renewed every shuffling, the query continues to progress and could produce new results after each shuffling without congesting the network. To speed up query execution, browsers processing similar queries are connected through a semantic overlay network. Experimentation shows that the number of answers produced by queries grows with the number of executed queries in the network while the number of exchanged messages is always bounded per user.

This work opens several perspectives. First, data exchange between browsers has to be optimized when a data source is revisited to optimize the traffic. Second, intermediate results could be streamed on the semantic overlay network. Finally, a DHT service could be implemented in browsers to speed up the discovery of similar queries in the network.

## 6 Acknowledgements

This work was partially funded by the French ANR project O'Browser (ANR-16-CE25-0005-01). Mr. Grall is funded by the GFI Informatique compagny (145 Boulevard Victor Hugo, 93400, Saint Ouen, France).

## References

1. Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., Schmidt, R.: P-grid: a self-organizing structured p2p system. ACM SIGMOD Record 32(3), 29–33 (2003)

2. Aberer, K., Cudré-Mauroux, P., Hauswirth, M., Van Pelt, T.: Gridvine: Building internet-scale semantic overlay networks. In: International semantic web conference. pp. 107–121. Springer (2004)
3. Abiteboul, S., Bienvenu, M., Galland, A., Antoine, É.: A rule-based language for web data management. In: Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. pp. 293–304. ACM (2011)
4. Bertier, M., Frey, D., Guerraoui, R., Kermarrec, A., Leroy, V.: The gossple anonymous social network. In: 11th International Middleware Conference 'Middleware 2010) - ACM/IFIP/USENIX. LNCS, vol. 6452, pp. 191–211. Springer (2010), `http://dx.doi.org/10.1007/978-3-642-16955-7_10`
5. Buchegger, S., Schiöberg, D., Vu, L.H., Datta, A.: Peerson: P2p social networking: early experiences and insights. In: Proceedings of the Second ACM EuroSys Workshop on Social Network Systems. pp. 46–52. ACM (2009)
6. Crespo, A., Garcia-Molina, H.: Semantic overlay networks for p2p systems. In: International Workshop on Agents and P2P Computing. pp. 1–13. Springer (2004)
7. Diallo, O., Rodrigues, J.J., Sene, M., Lloret, J.: Distributed database management techniques for wireless sensor networks. IEEE Transactions on Parallel and Distributed Systems 26(2), 604–620 (2015)
8. Doulkeridis, C., Vlachou, A., Nørvåg, K., Vazirgiannis, M.: Distributed semantic overlay networks. In: Handbook of Peer-to-Peer Networking, pp. 463–494. Springer (2010)
9. Filali, I., Bongiovanni, F., Huet, F., Baude, F.: A survey of structured p2p systems for rdf data storage and retrieval. In: Transactions on large-scale data-and knowledge-centered systems III, pp. 20–55. Springer (2011)
10. Folz, P., Skaf-Molli, H., Molli, P.: CyCLaDEs: a decentralized cache for Linked Data Fragments. In: ESWC: Extended Semantic Web Conference (2016)
11. Grubenmann, T., Bernstein, A., Moor, D., Seuken, S.: Challenges of source selection in the wod. In: International Semantic Web Conference. pp. 313–328. Springer (2017)
12. Hartig, O.: Squin: a traversal based query execution system for the web of linked data. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. pp. 1081–1084. ACM (2013)
13. Hernández, A.G., GARCıA, M.: A javascript rdf store and application library for linked data client applications. In: Devtracks of the, WWW2012, conference. Lyon, France. Citeseer (2012)
14. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and replication in unstructured peer-to-peer networks. In: Proceedings of the 16th international conference on Supercomputing. pp. 84–95. ACM (2002)
15. Mansour, E., Sambra, A.V., Hawke, S., Zereba, M., Capadisli, S., Ghanem, A., Aboulnaga, A., Berners-Lee, T.: A demonstration of the solid platform for social web applications. In: Proceedings of the 25th International Conference Companion on World Wide Web. pp. 223–226. International World Wide Web Conferences Steering Committee (2016)
16. Molli, P., Skaf-Molli, H.: Semantic web in the fog of browsers. In: Proceedings of the Workshop on Decentralizing the Semantic Web 2017 co-located with 16th International Semantic Web Conference (ISWC 2017) (2017)
17. Montoya, G., Skaf-Molli, H., Molli, P., Vidal, M.E.: Federated SPARQL Queries Processing with Replicated Fragments. In: The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference. Bethlehem, United States (Oct 2015), `http://hal.univ-nantes.fr/hal-01169601`

18. Nédelec, B., Tanke, J., Frey, D., Molli, P., Mostéfaoui, A.: An adaptive peer-sampling protocol for building networks of browsers. World Wide Web pp. 1–33 (2017)
19. Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., Risch, T.: Edutella: a p2p networking infrastructure based on rdf. In: Proceedings of the 11th international conference on World Wide Web. pp. 604–615. ACM (2002)
20. Özsu, M.T., Valduriez, P.: Principles of distributed database systems. Springer (2011)
21. Staab, S., Stuckenschmidt, H.: Semantic Web and peer-to-peer. Springer (2006)
22. Voulgaris, S., Gavidia, D., Van Steen, M.: CYCLON: inexpensive membership management for unstructured P2P overlays. Journal of Network and Systems Management 13(2), 197–217 (2005)