# NREngine: A Graph-Based Query Engine For Network Reachability

Wenjie Li[1], Lei Zou[1], Peng Peng[2], and Zheng Qin[2]

[1] Peking Universtity, Beijing,China
[2] Hunan University, Changsha, China
{liwenjiehn,zoulei}@pku.edu.cn,{hnu16pp,zqin}@hnu.edu.cn

**Abstract.** A quick and intuitive understanding of network reachability is of great significance for network optimization and network security management. In this paper, we propose a query engine called *NREngine* for network reachability when considering the network security policies. NREngine constructs a knowledge graph based on the network security policies and designs an algorithm over the graph for the network reachability. Furthermore, for supporting a user-friendly interface, we also propose a structural query language named *NRQL* in NREngine for the network reachability query. The experimental results show that NREngine can efficiently support a variety of network reachability query services.

**Keywords:** Network Reachability · Network Security Policies · RDF · Graph Database.

## 1 Introduction

Network reachability is an important basis for network security services, which has attracted more and more attentions of the experts and scholars. Network reachability is a functional characteristic of the network, which ensures smooth communication between nodes in order for users to conveniently access the resources of the network[1].

Considering the requirement of network security or privacy protection, users usually configure various of security policies in network devices such as firewalls, routers and so on. Security policies usually restrict the users' access to the network, and its function is to control network reachability. Obviously, there needs to be a balance between ensuring normal network communication and achieving network security or privacy protection. In another word, the network reachability needs to be maintained within a suitable range. If the network reachability is more than the actual requirement, it may cause unnecessary communication, or even create opportunities for the malicious attacks; and the network reachability that is less than the actual requirement will disrupt the normal network services, and even lead to huge economic losses. Therefore, the network must have suitable reachability.

In order to measure the network reachability, there are many traditional methods to check whether the network is reachable or not by using the ping, traceroute or other tools. These methods have the following two shortcomings. First, the results are dependent on the state of the devices. If some devices in the query path get offline, the query results may be always unreachable. Second, these methods send the test data packet (such as ICMP data packet) to evaluate the network reachability, which costs a lot of network resources.

With the increase of network devices and the expansion of network scale, it will become a hot and difficult point to quantify the reachability model of the whole network. Furthermore, it has important theoretical value and application prospects to validate the network reachability through an efficient network reachability query approach, and intelligently locate the defects in security policies configuration according to the query results, and optimize the security policies configuration and network performance.

After constructing the network reachability model, we can use the graph traversal search algorithms to query the reachability. Those methods require searching the graph globally for network reachability and often have low performances. Furthermore, if the network reachability model is stored in two-dimensional database tables or files, we should reconstruct the graph when querying, which greatly reduces the performance dramatically; If the network reachability model is stored in memory, once the system is offline or downtime, it cannot save the data of the network reachability model and also are limited by the memory capacity. Therefore, in this study, we build up a knowledge graph of network reachability based on network security policies and transform the network reachability query into queries over the knowledge graphs. Then, we can maintain the knowledge graph of network reachability in graph databases, like gStore [2, 3], Jena [4], rdf4j [5] and Virtuoso [6], which can gain the high performances when evaluating queries over the knowledge graphs of network reachability.

### 1.1 Key Contributions

In this paper, we focus on the query of the network reachability. The main contributions of this paper are as follows:

1. We propose a novel model for the network reachability based on network security policies, and construct a knowledge graph of network reachability.
2. We extend the structured query language over knowledge graphs and propose a new structured query language called *NRQL* for network reachability. We design the efficient query algorithms for evaluating NRQL statements over the knowledge graph of network reachability.
3. We propose a novel query engine for the network reachability called *NREngine*, which implements all the above techniques.
4. To evaluate the effectiveness and efficiency of *NREngine*, we conduct extensive experiments.

## 2 Related Work

Recently, there are some effective works on the network reachability. Xie *et al.* have made a pioneering work, they define the network reachability and propose a method to model the static network reachability[1]. The key idea is to extract the configuration information of routers in the network and reconstruct the network into a graph in a formal language, so the network reachability can be calculated through classical problems such as closure and shortest path.

Zhang *et al.* propose a method to merge the IP addresses with the same reachability into the IP address sets[7]. When the network reachability is changed, the affected IP address sets can be reconstructed quickly by splitting or merging to update the network reachability in real-time. However, they do not provide an algorithm to answer whether an IP is reachable along a certain path to another IP. Benson *et al.* propose the concept of the policies unit[8] . The policies unit is a set of IP addresses affected by the same security policies. A policies unit may be distributed in many subnets, or there may be many different policies units in a subnet. They also do not provide an algorithm for the network reachability query.

Amir *et al.* propose a network reachability query scheme based on network configurations (mainly ACLs), and construct a network reachability query tool called "Quarnet"[9],[10]. Its ACL model still adopts FDD model, the queries need to be split by the paths of the FDD. Chen *et al.* propose the first cross-domain privacy-preserving protocol for quantifying network reachability[11]. The protocol constructs equivalent representations of the ACL rules and determines network reachability while preserving the privacy of the individual ACLs. They do not consider the other network security policies(such as route table). Hone *et al.* propose a new method to detect IP prefix hijacking based on network reachability, which is a specific application of network reachability[12].

Recently, there are some effective works on network research based on the graph. Liang *et al.* propose an improved hop-based reachability indexing scheme 3-Hop which gains faster reachability query evaluation, which has less indexing costs and better s-calabilities than state-of-the-art hop-based methods, and they propose a two-stage node filtering algorithm based on 3-Hop to answer tree pattern queries more efficiently[13]. Rao *et al.* propose a model of network reachability based on decision diagram[14]. Li *et al.* propose a verification method of network reachability based on the topology path. They transform the problem of the communication need into the verification problem of topology path reachability via SNMP and Telnet-based topology discovery and graph theory techniques[15]. Alfredo *et al.* propose a novel reachability-based theoretical framework for modeling and querying complex probabilistic graph data[16]. Hasan proposes a novel knowledge representation framework for computing sub-graph isomorphic queries in interaction network database[17].

## 3 Overview

### 3.1 Problem Definitions

The essence of network reachability query is to determine whether a certain type of network packet can reach another node from one node or from one subnet to another. Given two subnets $N_1$ and $N_2$, and two host nodes $v_1$ and $v_2$, where $v_1$ is a node of $N_1$, so $v_1 \in N_1$ holds, and $v_2$ is a node of $N_2$, so $v_2 \in N_2$ holds. The network reachability query in this paper can be divided into three categories as follows according to the query targets.

**Node to Node**. This category of query is mainly used to check the network reachability between nodes. We use $v_1 \rightarrow v_2$ to denote that $v_1$ to $v_2$ is reachable, and use $v_1 \nrightarrow v_2$ to denote that $v_1$ to $v_2$ is unreachable.

**Node to Subset**. This category of query is mainly used to check the network reachability between nodes and subsets. We use $v_1 \rightarrow N_2$ to denote that $v_1$ to $N_2$ is reachable, and use $v_1 \nrightarrow N_2$ to denote that $v_1$ to $N_2$ is unreachable. Obviously, the following formula holds.

$$v_1 \rightarrow N_2 = \{\exists v_j,\ \ v_1 \rightarrow v_j\}(v_j \in N_2)$$

**Subset to Subset**. This category of query is mainly used to check the network reachability between subnets. We use $N_1 \rightarrow N_2$ to denote that $N_1$ to $N_2$ is reachable, and use $N_1 \nrightarrow N_2$ to denote that $N_1$ to $N_2$ is unreachable. Obviously, the following formula holds.

$$N_1 \rightarrow N_2 = \{\exists v_i, v_j,\ \ v_i \rightarrow v_j\}(v_i \in N_1, v_j \in N_2)$$

We also can divide the network reachability query into two categories as follows according to the result of query.

1. **Boolean query**. The result of the query is a boolean value (such as yes or no). For example, "SMTP server 192.168.0.32 to host 192.168.0.54 is reachable?" , and the result is "yes" or "no".
2. **Node query**. The result of the query is a set of nodes that satisfy the query condition. For example, "Which hosts in subset 192.168.0.0/24 can receive the email from the SMTP server 192.168.0.32?" . The result is a set of nodes.

### 3.2 System Architecture

In this paper, we propose a query engine for network reachability based on network security policies, *NREngine*. Fig. 1 shows the system architecture of NREngine. NREngine consists of two parts as follows.

In the offline part of NREngine, we collect and organize the network security policies (including ACLs and routing table). First, we remove the network security policies where the action field value is *deny* and extend OPTree [18] for maintaining the network security policies. OPTree is a homomorphic structure of network security policies, and the redundancy policies and the conflict policies can be removed when constructing OPTree. Then, we propose a network reachability model based on the network topology and the network security policies, and construct a knowledge graph based on the network reachability model. To support efficient evaluation of the network reachability query, we maintain the knowledge graph in graph databases, like gStore [2, 3], Jena [4], rdf4j [5] and Virtuoso [6].

The online part of NREngine is mainly responsible for processing user query requests. In this part, we propose a structured query language, called *NRQL* (*N*etwork *R*eachability *Q*uery *L*anguage) for network reachability query. Users send the query requests to NREngine using NRQL statements. The key of NREngine query is to match the query conditions by using OPTree in the network reachability model. In order to adapt to *OPTree*, we propose a NRQL query parsing algorithm based on OPTree query algorithm. NREngine can provide three categories of queries: "node to node", "node to subnet" and "subnet to subnet". The results of those queries can be either "Yes" or "NO", or a set of nodes satisfying the query conditions.

Considering the high real-time requirement of network reachability query, the relatively low frequency of network security policies change, and the long time-consuming construction of the network reachability model based on OPTree, we construct or update the network reachability model through timing schedule in the offline part. In the online part, NREngine provides a real-time network reachability query service based on the high query efficiency of graph databases. In terms of system deployment, the online part and the offline part can be deployed independently, in which the offline part can be deployed in the intranet environment to isolate ordinary users, and the online part can be deployed in the extranet environment to provide network reachability query services to ordinary users by the GUI of NREngine.

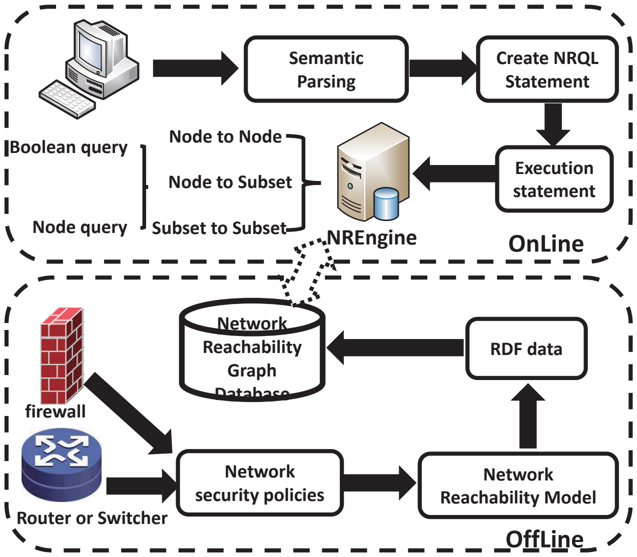


**Fig. 1.** The system architecture of NREngine

## 4 The Offline Part of NREngine

### 4.1 Network Reachability Model based on Network Security Policies

The essence of network reachability in our paper is to determine whether network data packets can be transferred from one node to another. Many factors affect whether network data packets can be transmitted from one node to another, such as the state of the devices. The devices include firewalls, routers, switchers and hosts. If a device is not online, the data packet can not be transmitted through it. However, the state of the device is an instantaneous state, which may be man-made shutdown or the device failure. By opening or repairing the fault, the state of the device can be changed, so it cannot reflect the basic state of the network.

Another key factor affecting the network data package is the network security policies. The network security policy is not an instantaneous state, and cannot be changed due to the device off-line or the device failure. Therefore, the network security policies can well reflect the basic state of the network, and we can build a model for the network reachability based on the network security policies. Noted that, because the security policies in hosts are managed by their managers and it is hard to collect the security policies, we do not consider those security policies in our paper.

First, we formally define the network reachability model. Given a subnet $N$ and there are $n$ devices $(D_1, D_2, \ldots, D_n)$ in $N$. Here, we use the graph as the basic model of the network reachability model. The network is denoted as $G = (V, E, L)$, where $V$ denotes the set of devices like firewall, router or switcher in the subnet $N$; $E$ denotes a set of the edges between vertices and $L$ is the labels of the edges based on the network security policies. According to the properties of ACL and routing table, if $v_i$ has only one outgoing edge, $v_i$ denotes a device which has packet filtering function, such as firewall. Otherwise $v_i$ denotes a device which has packet forwarding function ,such as router or switcher. Obviously, $G$ is a directed graph. Fig. 2 shows an example of the network reachability model. In Fig. 2, $v_5$, $v_6$ and $v_7$ denote the network security devices which have packet filtering function, and $v_1$, $v_2$, $v_3$ and $v_4$ denote network security devices which packet forwarding function.

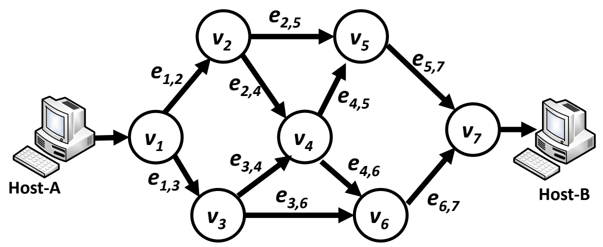


**Fig. 2.** An example of the network reachability model

The construction of network reachablity model $G = (V, E)$ for subset $N$ has two steps as follow as.

**Step 1: Creating the vertices set for devices.** We create a vertex for each network security device in subset $N$. Noted that if a network security device $D_i$ not only has a packet filtering function but also has a packet forwarding function, in other words, the action field's values of the network security policies $R_i$ in $D_i$ have three categories: *Accept*, *Deny* and *NextDevice*. we should create two virtual vertexes $D'_i$ and $D''_i$ to denote $D_i$, and $D'_i$ only has a packet filtering function, and $D''_i$ only has a packet forwarding function.

**Step 2: Linking the vertices of devices.** According to the topological structure of the network $N$ and the flow direction of data packets, the edges between nodes are constructed. Given a vertex $v_i$ with packet filtering function, and the next node is $v_j$, we construct an edge $e_{i,j}$ that from $v_i$ to $v_j$, and construct the OPTree $T_{v_i}$ based on the network security policies which in $v_i$ , we use $L(e)$ to denote the label of the edge $e$, that is, $L(e_{i,j})$= $T_{v_i}$. If $v_i$ has the packet forwarding function, then there are many next-hop nodes of $v_i$. we construct an edge $e_{i,j}$ for each next-hop node $v_j$, and use $R(i, j)$ to denote the network security policies which is in $v_i$ and the next-hop node is $v_j$, and we construct the OPTree $T_{R(i,j)}$ and $L(e_{i,j})$=$T_{R(i,j)}$.

## 4.2 Knowledge Graph of Network Reachability

After constructing the network reachability model, the next step is to efficiently evaluate the network reachability query. Existing methods of maintaining the network reachability model in two-dimensional database tables or files have low performances or are limited by memory capacity. Thus, in this study, we build up a knowledge graph of network reachability based on network security policies and transform the network reachability model into edges of knowledge graphs. Fig. 3 shows an example knowledge graph of network reachability. Then, we can maintain the knowledge graph of network reachability in graph databases, like gStore [2, 3], Jena [4], rdf4j [5] and Virtuoso [6], which can gain the high performance of evaluating network reachability queries.

The schema of vertices in our knowledge graph of network reachability is shown in Table 1, which includes three categories of vertices ("DeviceType", "Network" and "Edge"). For the devices that have the packet forwarding function, there is more than one "Edge" vertex, and for the devices that have the packet filtering function, there is only one "Edge" vertex.

| Name | Type | Remark |
|---|---|---|
| DeviceType | property | the device type of the vertex, such as firewall,router,switcher,and host |
| Network | property | the network of the vertex belongs,such as $N_1$ |
| Edge | resource | the edge of the vertex,such as $e_{1,2}$ |

**Table 1.** Schema of Vertices in Knowledge Graph of Network Reachability

The schema of edges in our knowledge graph of network reachability is shown in Table 2. The most important category of edges is "Label". The value of a "Label" edge represents the set of network security policies which is to be matched by a data packet pass through the edge. In order to improve the query efficiency, we maintain OPTree in memory to store the network security policies.

| Name | Type | Remark |
|---|---|---|
| Label | property | The label value of the edge. In this s-tudy, the value is an OPTree. |
| NextNode | resource | The vertex which the edge point to. |

**Table 2.** Schema of Edges in Knowledge Graph of Network Reachability

## 5 The Online Part of NREngine

### 5.1 Structured Query Language for Network Rechability

In this section, we extend the structured query language over knowledge graphs, SPARQL [19], for describing the user's network reachability query request over the knowledge graph of network rechability. The extended structured query language is called *NRQL* (Network Reachability Query Language).

Generally, SPARQL does not support matching operations related to the *policy matching condition*, a SPARQL statement only supports queries with limited steps. If the target of a query statement is to find the data which meet the query conditions within 3 steps, its meaning is to find data within the range of 3 edges. In network reachablility query, we do not know the number of edges between the starting node and the

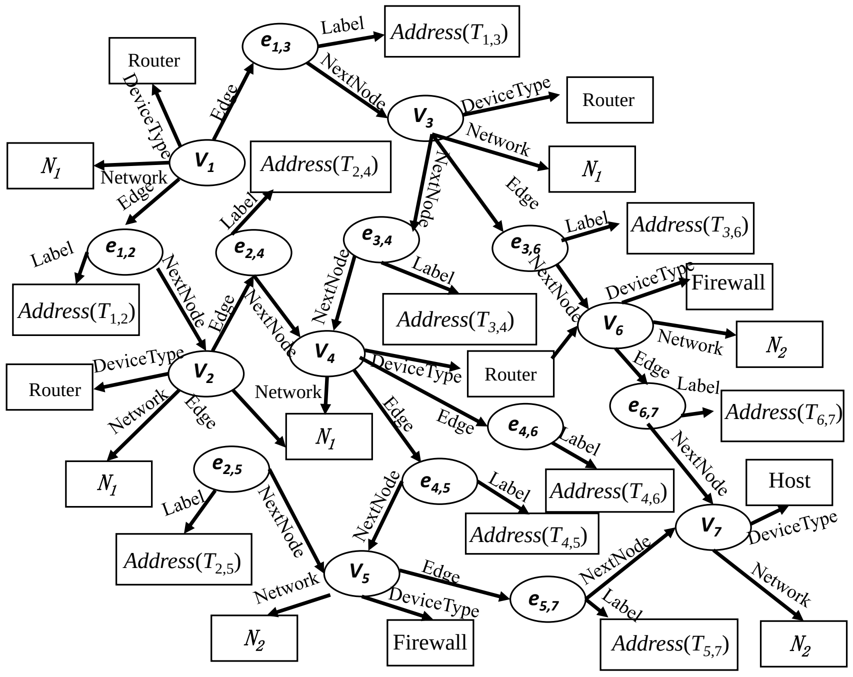


**Fig. 3.** Example Knowledge Graph of Network Reachability

target node. Therefore, we replace the starting vertex recursively and perform a one step SPARQL query in the recursive process.

A NRQL statement consists of two parts: a query target clause beginning with keyword *select* and a ? to denote a query variable. Therefore, a query target clause contains several query targets by several query variables. Fig. 4 shows an example of NRQL statement. There are two query targets in Fig. 4.

```
Select ?x,?y
Where
{
   ?x  Network ?y.
   ?x  DeviceType "Host".
   ?z  NextNode ?x.
   ?z  Label ?k.
   ?k   <nr:in> <SIP=192.168.32.0/24,SP=any, DIP=192.168.24.212,DP=23,P=tcp>.
}
```

**Fig. 4.** Example Point Query

Fig. 4 shows a node query. We use the keyword *exist* to describe the boolean query, and we only add *exist* for the query variables of the query statement for boolean query is shown in Fig. 5.

The other part of the NRQL statement is a query condition clause beginning with keyword *where* , which is wrapped with braces and contains several triple patterns. Each triple pattern is a query condition for edges in knowledge graph. Here, to support matching the data package with network security policies in the network reachability query, we define a new predicate "<nr: in >" in NRQL. This predicate means that the content of the object is regarded as a data packet, and then the process is transformed into the problem of checking whether a data packet is matched with a set of network security policies. We define this query condition clause as a *policy matching condition*. In a query condition clause, there is only one *policy matching condition*. In [18], Li *etc.* propose that using OPTree can solve this problem efficiently, so it can be transformed into finding a predicate path in OPTree, which can be solved by using OPTree search algorithm.

```
Select exist(?x)
Where
{
   ?x  Network ?y.
   ?x  DeviceType "Host".
   ?z  NextNode ?x.
   ?z  Label ?k.
   ?k   <nr:in> <SIP=192.168.32.0/24,SP=any, DIP=192.168.24.212,DP=23,P=tcp>.
}
```

**Fig. 5.** Example Boolean Query

## 5.2 Execution of NRQL Statements

After we generate NRQL query statements based on the user's query request, then we execute the statements over the knowledge graph of network reachability. Unfortunately, most graph databases for knowledge graphs, like gStore [2, 3], Jena [4], rdf4j [5] and Virtuoso [6], do not directly support the NRQL query statements, but they only support the SPARQL query statements. Therefore, when NRQL query statements are executed, additional parsing and pre-processing of NRQL query statements are needed.

```
Algorithm 1: NRQLBooleanQuery($Q, G$)
  Input: $Q$:The NRQL query statement.
  $G$:The network reachability model.
  Output: $True$:Network is reachable; $False$:Network is unreachable
1 set $v_c$=$v_{start}$;
2 $result$=$checkIsMatch$($v_c$, $Q_{select}$, $R_{kg}$, $r_{match}$);
3 if $result$==$false$ then
4     return false;
5 else
6     generate a sparql statement $ask$ which the condition clause is $R_{kg} - r_{match}$.
          /* Execute the ask by the query api of the graph
          database                                          */
7     return $graphDatabase.query(ask)$;
```

Before describing the query algorithm, we formally define the common concepts which would be used in the algorithm. We use $G = (V, E, L)$ to denote the network reachability model, and use $Q$={$Q_{select}$, $Q_{where}$} to denote a NRQL query statement, in which $Q_{select}$ denotes the query target clause and $Q_{where}$ denotes the query condition clause. According to the above description, the query condition clause contains a set of triple patterns, so $Q_{where}$=$R_{kg}$, where $R_{kg}$ denotes a set of edges in the knowledge graph of network reachability and $r_i$ denotes a triple tuple in the set of $R_{kg}$. The $r_{match}$ denotes the *policy matching condition*.

We design the query algorithm of the NRQL according to the categories of the query of network reachability and the categories of the query result. For boolean query, we design the boolen query algorithm, whose pseudo-code is shown as Algorithm 1.

```
Function Boolean checkIsMatch($v_c$, $Q_{select}$, $R_{kg}$, $r_{match}$)
if $v_c \in Q_{select}$ then
    return true;
else
    /* get the next vertexes of $v_c$ which match with $r_{match}$ */
    set $V_{next}$=$getNextVertexes(v_c, r_{match})$;
    if $V_{next} \neq \emptyset$ then
        /* match $r_{match}$, then call the recursive function */
        set $flag$=$true$;
        for $i = 0$ to $V_{next}.length$ do
            set $v_c$=$V_{next}[i]$;
            $flag$=$flag$ || $checkIsMatch(v_c, Q_{select}, R_{kg}, r_{match})$;
        if $flag$==$false$ then
            return false;
        else
            return true;
    else
        return false;
```

There are two key functions for boolean query in Algorithm 1. Function *checkIsMatch* is a recursive function. For example, in node to node query, we use $v_{start}$ to denote the start vertex, and use $v_{end}$ to denote the end vertex. Firstly, we set $v_c$=$v_{start}$ and call the function *getNextVertexes* to get the next nodes. In function *getNextVertexes*, we generate a sparql query statement to get the next vertexes by using the query api of the graph database and use $V_{next}$ to denote the vertexes which satisfy the condition clause $R_{kg}$-$r_{match}$. Secondly, we check whether each vertex $v_i$ match with $r_{match}$, if a vertex can match with $r_{match}$, we set $v_c$=$v_i$, and repeatedly execute the function *checkIsMatch* until every vertex in $V_{next}$ has been checked. Finally, if the result of function *checkIsMatch* is false, it means that there is no path that satisfies the query condition clause, the result of node to node query is false. Otherwise, we generate a SPARQL ask statement where the condition clause is $R_{kg}$-$r_{match}$. According to the properties of ask statement of SPARQL, the result is a boolen value.

The node query algorithm is similar to the boolean query algorithm, except that the result is a set of nodes, and the pseudo-code of the algorithm is shown in Algorithm 2.

In Algorithm 2, we finally generate a SPARQL query statement to get the vertexes which satisfy with $R_{kg}$-$r_{match}$.

**Function** Vertexes getNextVertexes($v_c, r_{match}$)

```
   /* generate the query statement based on the SPARQL */
1  set sparql=' select ?address,?z where { < $v_c$ > edge ?y. ?y label ?address. ?y NextNode
     ?z.}';
2  $json$=$graphDatabase.query(sparql)$;
3  if $json$==$null$ then
4      return $\emptyset$;
5  else
6      set $V_{list}$=[];
7      set $edgeList$=$json.list$;
8      for $i$ = 1 to $edgeList$ do
           /* get the Object of the OPTree according the memory
              address of the OPTree */
9          set $T$=$Address(edgeList[i].address)$;
           /* use the search algorithm of the OPTree */
10         set $path$=$T.search(r_{match})$;
11         if $path \neq null$ then
               /* It means that the edge of the vertex $V_c$ match
                  the $r_{match}$ when there is a path */
12             $V_{list}$.add($edgeList[i].endNode$);
13         else
14             continue;
15 return $V_{list}$;
```

**Algorithm 2: NRQLNodeQuery**($Q, G$)

```
   Input: $Q$:The NRQL query statement for the network reachability query.
   $G$:The network reachability model.
   Output: the node set which satisfies the query condition
1  set $v_c$=$v_{start}$;
2  $result$=$checkIsMatch(v_c, Q_{select}, R_{kg}, r_{match})$;
3  if $result$==$false$ then
4      return null;
5  else
6      generate a sparql statement select which the query target clause is $Q_{select}$ and the
         query condition clause is $R_{kg} - r_{match}$.
       /* Excute select by using the query api of the graph
          database. */
7      return $graphDatabase.query(select)$;
```

## 5.3 Analysis

For Algorithm 1, we find the key process of boolean query is to match all the output edges of the starting node with the *policy matching condition*. Because the *gStore* has a high level of query efficiency, the query time of *gStore* can be ignored. This checking process is equivalent to the searching process of OPTree. The time complexity of OPTree search algorithm as $O(m\dot{l}og)$, assuming that the number of output edges of each node is $k$, and there are $q$ nodes between the starting node and the target nodes of the query. The best case is that there is not a output edge $e$ of the starting node which can match the *policy matching condition*, then we only need to execute the checking process for the starting node once. Thus the time complexity is $O(km\dot{\log}n)$. The worst case is that we need to execute the checking process for every node between the starting node and the target node. The time complexity is $O(kqm\dot{\log}n)$. Therefore, the time complexity of the boolean query algorithm is $O(kpm\dot{\log}n)(1 \leq p \leq q)$.

Noted that the node query algorithm is similar to the boolean query algorithm, therefore, the time complexity of the node query algorithm is $O(kpm\dot{\log}n)(1 \leq p \leq q)$.

## 6 Experiments

In this section, we perform our experiments and evaluate the effectiveness and efficiency of NREngine.

### 6.1 Setting

In this paper, we propose a knowledge graph-based query engine for network reachability, and construct the network reachability model based on the network topology and the network security policies. Therefore, in our experiments, we should generate three categories of datasets as follow.

1).**Datasets of Network Topology**. The data set includes the devices and the edges between the devices. The devices include the host, router, switcher and firewalls. The size of the devices in the generated network topology ranges from 10 to 100 with the step length of 10, and the ratio of firewalls in those devices is 30%, the ratio of routers or switchers in those devices is 70%, and the size of the forwarding ports in routers or switchers is no more than 4. Noted that in our experiment, the size of subnets is 3, and each of subnet, we generate 2 hosts.

In order to be closer to the actual situation, we generate the random size of the forwarding ports for each router and switcher, so the size of the edges in the network reachability model is uncertain. However, the size of the edges in the network reachability model can intuitively reflect the complexity of the network. Therefore, We use the size of the edges in the network reachability model as the metrics in our experiment. Table 3 shows the generated data set of network topology in our experiments.

2).**Datasets of Network Security Policies**.We use the network security policies generation tool *ClassBench* proposed in [20], which is widely used in policies generation to generate the network security policy sets of the devices in the network.The size of the generated network security policy sets in each device range from 100 to 1000 with the step length is 100, note that each field of a network security policy generated by *ClassBench* is represented as a range, we need to transform the range value to one or more prefixes based on the properties of OPTree.

3).**Datasets of NRQL Query Statements**. We generate three categories of NRQL query statement: node to node query, node to subnet query and subnet to subnet query. We generate 100 NRQL query statements for each category. Noted that the start node is different from the end node in the generated NRQL query statements.

| Scale of Devices | Devices | | Edges |
|---|---|---|---|
| | Routers or Switchers | Firewalls | |
| 10 | 7 | 3 | 89 |
| 20 | 14 | 6 | 503 |
| 30 | 21 | 9 | 912 |
| 40 | 28 | 12 | 1293 |
| 50 | 35 | 15 | 1723 |
| 60 | 42 | 18 | 2207 |
| 70 | 49 | 21 | 2498 |
| 80 | 56 | 24 | 2974 |
| 90 | 63 | 27 | 3319 |
| 100 | 70 | 30 | 3812 |

**Table 3.** Datasets

We perform our experiments on PC running Centos7.2 with 32GB memory and 4 cores of Intel(R) Xeon(R) processor(3.3GHz) and implement our prototype system using Java. The graph database used for maintaining the knowledge graph of network reachability is *gStore* [2, 3].

For the offline part of NREngine, we generate the knowledge graph of network reachability and construct OPTrees for the labels of edges. We measure the execution time and memory usage of the knowledge graph.

For the online part of NREngine, we perform two categories of the network reachability query: the boolean query and the node query. To evaluate the efficiency of N-REngine, we measure the query time in our experiments. In our experiments, we execute the three categories network reachability query: node to node query, node to subnet query and subnet to subnet query that use the same query condition clauses and only change the query target clauses.

In order to make the experimental results more accurate, we execute the all NRQL query statements, and then measure their average query time.

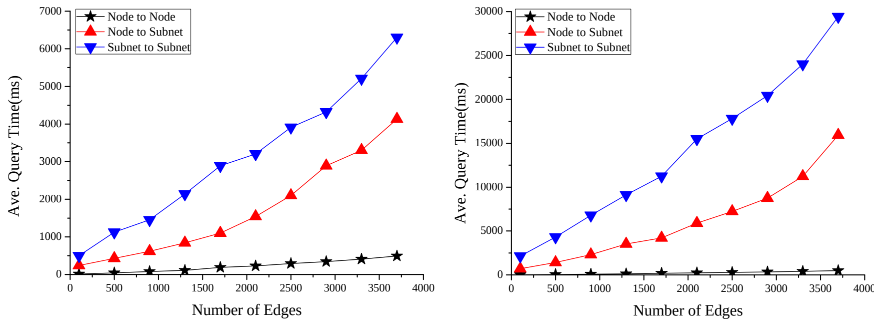
## 6.2 Experiments Results of Offline Part

For the offline part of NREngine, the results of experiments are shown in Table 4. Noted that the number of edges is the size of edges in network reachability model. On the one hand, the experimental results show that we can build a knowledge graph of network reachability with 3700 edges in less than three hours, and the memory usage is less than 3GB. Because of the knowledge graph construction is an offline process, so the time cost and the space cost are acceptable.

On the other hand, the experimental results show that the time and the memory usage of OPTree construction are more than that of knowledge graph generation. The reason is that OPTree Construction is a time-consuming operation, and it includes path checking and path merging, and network reachability model can be quickly converted into knowledge graph based on pattern matching.

| Number of Edges | Knowledge Graph Generation | | OPTree Construction | |
|---|---|---|---|---|
| | Ave. Time(s) | Ave. Memory Size(MB) | Ave. Time(min) | Ave. Memory Size(MB) |
| 100 | 12.56 | 12.42 | 3.51 | 40.42 |
| 500 | 18.52 | 20.42 | 11.52 | 218.32 |
| 900 | 23.41 | 25.62 | 21.42 | 398.23 |
| 1300 | 26.43 | 31.24 | 35.23 | 612.42 |
| 1700 | 31.24 | 35.62 | 51.52 | 812.25 |
| 2100 | 51.42 | 39.42 | 72.51 | 978.07 |
| 2500 | 69.41 | 42.54 | 98.53 | 1234.23 |
| 2900 | 87.09 | 48.23 | 119.64 | 1592.31 |
| 3300 | 97.14 | 55.21 | 138.52 | 1892.18 |
| 3700 | 112.52 | 60.87 | 150.42 | 2132.52 |

**Table 4.** Results on Knowledge Graph of Network Reachability Construction



(a) The results of boolean query (b) The results of node query

**Fig. 6.** The Results of Experiments

## 6.3 Experiments Results of Online Part

For the online part of NREngine, the results of experiments are shown in Fig. 6. The experimental results show that on the one hand the query time of network reachability is from milliseconds to seconds with the increase in the size of edges in network reachability model. On the other hand, the efficiency of node to node query is the highest, followed by node to subnet query, and the efficiency of subnet to subnet query is the lowest. The reason is that we need to traverse every node in the subnet until we find a node that satisfies the query conditions for node to subnet query and subnet to subnet query. Fig. 6(a) shows the experimental result of the boolean query, and Fig. 6(b) shows the experimental result. The experimental results show that the query time of node query is similar to that of boolean query in node to node query, and the query time of node query is much longer than that of boolean query in node to subnet query and subnet to subnet query. The reason is that we need find all nodes that satisfy the query conditions in the node query, and we just find one node that satisfies the query in the boolean query.

## 7 Conclusion

In this study, we propose a model of the network reachability based on the network security policies, and propose a query engine called "NREngine" for network reachablity. In order to improve the efficiency of network reachability query, some techniques are used to construct the network as a knowledge graph of network reachability and maintain the knowledge graph in graph databases. On this basis, the knowledge graph of network reachability is proposed for network reachability query. To describe user's network reachability query requests, we propose a structured query language, which is called NRQL, and design the the query algorithms for NRQL. The experimental results indicate that NREngine is effective and efficient.

**Acknowledgements.** This work is partially supported by The National Key Research and Development Program of China under grant 2018YFB1003504, the National Natural Science Foundation of China under Grant (No. U20A20174, 61772191), Science and Technology Key Projects of Hunan Province (2019WK2072, 2018TP3001, 2018TP2023, 2015TP1004), and ChangSha Science and Technology Project (kq2006029).

## References

1. Geoffrey G. Xie, Jibin Zhan, David A. Maltz, Hui Zhang, and Albert G. Greenberg. On static reachability analysis of ip networks. In *IEEE INFOCOM*, pages 2170–2183, 2005.
2. Lei Zou, Jinghui Mo, and Lei Chen. gStore: Answering SPARQL Queries via Subgraph Matching. *VLDB Endowment*, 4(8):482–493, 2011.
3. Lei Zou, M. Tamer Özsu, Lei Chen, Xuchuan Shen, Ruizhe Huang, and Dongyan Zhao. gStore: A Graph-based SPARQL Query Engine. *VLDB Journal*, 23(4):565–590, 2014.
4. Brian McBride. Jena: Implementing the RDF Model and Syntax Specification. In *SemWeb*, 2001.
5. Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In *ISWC*, pages 54–68, 2002.
6. Orri Erling and Ivan Mikhailov. Virtuoso: RDF Support in a Native RDBMS. In *Semantic Web Information Management*, pages 501–519. 2009.
7. Bo Zhang, T S Eugene Ng, and Guohui Wang. Reachability monitoring and verification in enterprise networks. In *ACM SIGCOMM*, pages 459–460, 2008.
8. Theophilus Benson, Aditya Akella, and David A Maltz. Mining policies from enterprise network configuration. In *The 9th ACM SIGCOMM conference on Internet measurement conference*, pages 136–142, 2009.
9. Amir R. Khakpour and Alex X. Liu. Quantifying and querying network reachability. In *the 29th International Conference on Distributed Computing Systems (ICDCS)*, pages 817–826, 2010.
10. Amir R Khakpour and Alex X Liu. Quantifying and verifying reachability for access controlled networks. *IEEE/ACM Transactions on Networking(TON)*, 21(2):551–565, 2013.
11. Fei Chen, Bruhadeshwar Bezawada, and Alex X. Liu. Privacy-preserving quantification of cross-domain network reachability. *IEEE/ACM Transactions on Networking(TON)*, 23(3):946–958, 2015.
12. Hong, Seong Cheol, H. Ju, and W. K. Hong. Network reachability-based ip prefix hijacking detection. *International Journal of Network Management*, 23(1):1–15, 2013.

13. Ronghua Liang, Hai Zhuge, Xiaorui Jiang, Qiang Zeng, and Xiaofei He. Scaling hop-based reachability indexing for fast graph pattern query processing. *IEEE Transactions on Knowledge and Data Engineering(TKDE)*, 26(11):2803–2817, 2014.
14. Zheng Chan Rao and Tian Yin Pu. Decision diagram-based modeling of network reachability. *Applied Mechanics and Materials*, 513:1779–1782, 2014.
15. Yazhuo Li, Yang Luo, Zhao Wei, Chunhe Xia, and Xiaoyan Liang. A verification method of enterprise network reachability based on topology path. In *the 2013 Ninth International Conference on Computational Intelligence and Security*, pages 624–629, 2013.
16. Alfredo Cuzzocrea and Paolo Serafino. A reachability-based theoretical framework for modeling and querying complex probabilistic graph data. In *IEEE International Conference on Systems*, pages 1177–1184, 2012.
17. Hasan Jamil. A novel knowledge representation framework for computing sub-graph isomorphic queries in interaction network databases. In *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, pages 131–138, 2009.
18. Wenjie Li, Zheng Qin, Keqin Li, Hui Yin, and Lu Ou. A novel approach to rule placement in software-defined networks based on optree. *IEEE ACCESS*, 7(1):8689–8700, 2019.
19. Marcelo Arenas, Claudio Gutiérrez, and Jorge Pérez. On the Semantics of SPARQL. In *Semantic Web Information Management*, pages 281–307. 2009.
20. David E. Taylor and Jonathan S. Turner. Classbench: A packet classification benchmark. *IEEE/ACM Transactions on Networking*, 15(3):499–511, 2007.