CMPE160 - Introduction to Object Oriented Programming

Project 3 - Domain Name System (DNS) Implementation

Deadline: 05.06.2020 - 23:55

1. Introduction

In this project, you are going to implement a basic Domain Name System (DNS) mechanism. Even though used extensively on the web, most of the time we are not aware of using it. DNS is a distributed and hierarchical naming system for the devices connected to the Internet. It has many functionalities, but generally its main objective can be expressed as transforming the domain names into corresponding IP addresses for accessing requested services.

A domain name, such as boun.edu.tr, is a string that is used for addressing issues. Since it is hard to memorize complicated IP addresses in the format of A.B.C.D (each part of an IP address is a decimal representation of a 8-bit number), the web sites or network domains are accessed through easily-memorizable domain names. However, the computers/terminals connected to the Internet are capable of communicating via only IP addresses. Therefore, the domain names should be initially resolved and its IP address should be obtained for communication. In other words, DNS works as a mapping mechanism, which provides the corresponding IP addresses for a requested domain name.

Here is a simple example to depict the operations of a DNS, which is also illustrated in Figure 1. A computer connected to the Internet wants to access the main page of www.boun.edu.tr and the user sends a request via a web browser. This request is initially forwarded to the DNS mechanism. After finding the IP address of the requested web site, it is sent to the user. Then, the browser requests the main page of www.boun.edu.tr after obtaining its IP address.

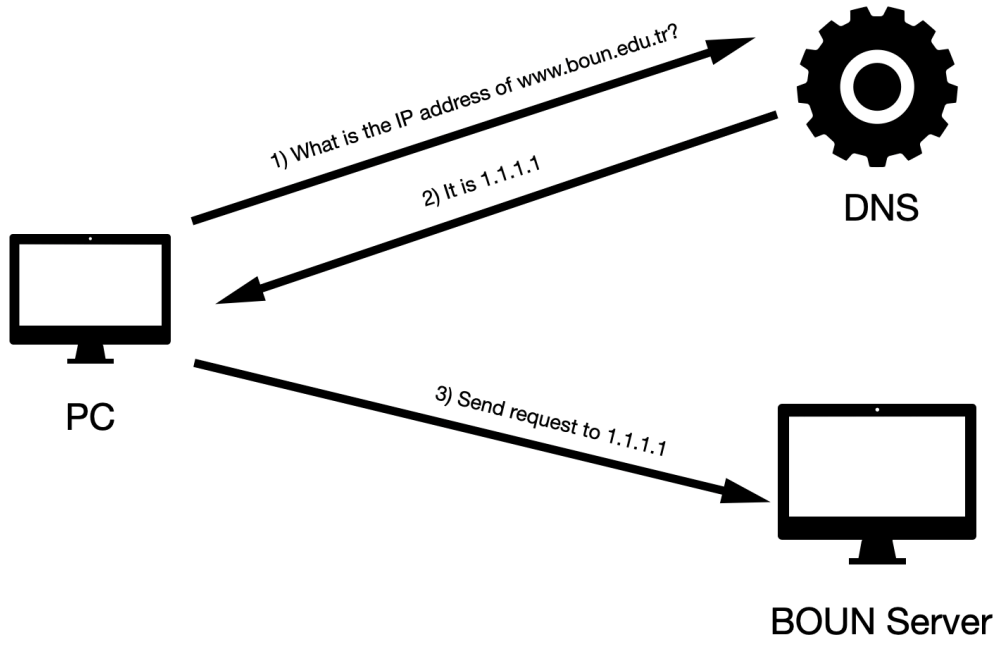

Figure 1. Simple illustration of DNS operations.

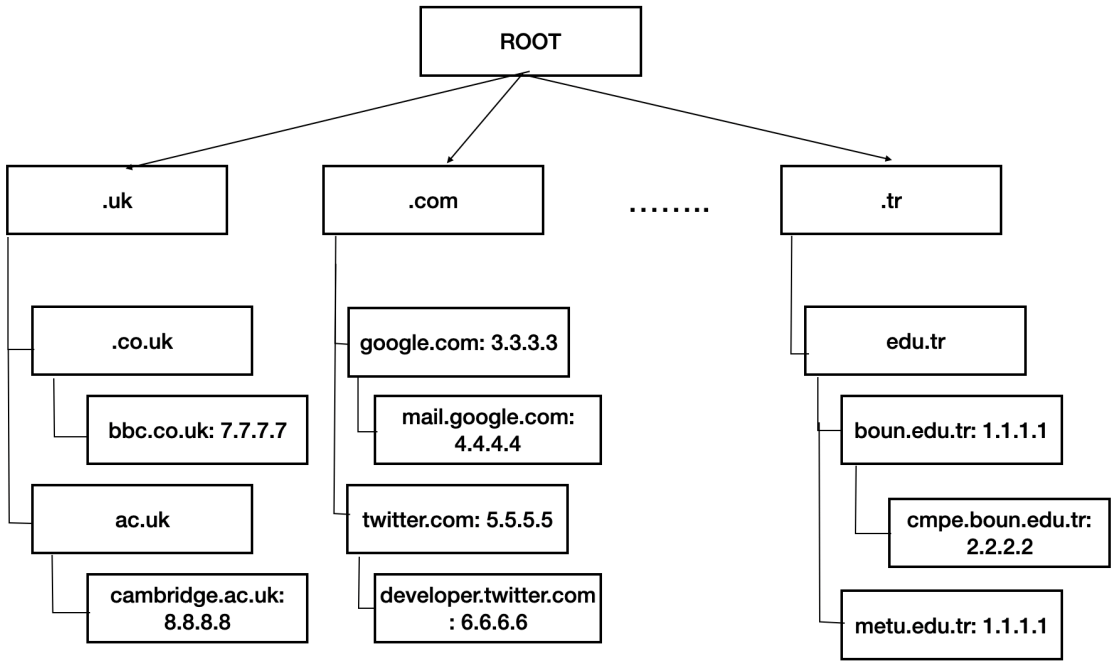

Figure 2. An example DNS structure

In fact, the original implementation, functionalities and structure of DNS are more complex, but in this project we are going to implement a simple version of it. It is represented as a tree structure, where the domains are organized in this tree accordingly. An example structure of overall DNS mechanism is shown in Figure 2. In this figure, it can be observed that the domain names are organized as subdomains of the DNS root. If a corresponding node represents a valid domain name, such as twitter.com, its IP address can be returned to the computer requesting this domain name. On the other hand, if a node does not include a valid domain name, such as .ac.uk, it does not have any IP address and it just functions as an intermediate node to the valid domain names.

Continuing from the previous example, after fetching the IP address of the BOUN main web server, the user computer stores this information in its local cache, so that it does not send another request to the DNS if BOUN web page is requested once more. However, local cache has a limited size, so that it can only store a finite number of domain name - IP address mapping. There are different algorithms to manage the cache replacement operations, and this project follows a simple one that is explained in the next section.

2. Implementation Details

The project consists of a single package named `question`, and all the classes should be implemented in this class. Actually, you are provided with the empty class files, so you need to complete the implementation of these classes.

2.1. DnsNode.java

This class represents each of the nodes in the DNS tree structure. The following fields should be defined for this class:

- `Map<String, DnsNode> childNodeList`: This field is utilized for maintaining the tree structure. The keys are represented as strings, and the values are DNS nodes. For example, assuming that the current node is .uk, the key “ac” refers to the node “.ac.uk" and the key “co” refers to the node of ".co.uk". Each key is a subdomain name and the value it is mapped represents a child node in the tree structure.

- `boolean validDomain`: It shows whether the current node is a valid domain name or just a subdomain that cannot have any IP address.
- `Set<String> ipAddresses`: This set is for storing the IP addresses as a list of a domain name. Please keep in mind that a full domain name may have more than one server, or IP address. For example, google.com has many servers worldwide, any subsequent requests for this domain will probably return different IP addresses to provide the load balancing.

The constructor of this class creates a DNS node with an empty `childNodeList` and `ipAddresses`. Besides, the `validDomain` should be initialized as "false".

After adding the first IP address to the list of a node, its `validDomain` should be set to true, since it represents a valid domain name with a corresponding IP address.

2.2. Client.java

The class representing the client side consists of simple implementation of the cache mechanism mentioned above. Within this class, a nested `private class CachedContent` should be implemented. The `CachedContent` class has the following fields:

- `String domainName`
- `String ipAddress`
- `int hitNo`

After obtaining an IP address for the requested web domain name, it is stored in the cache by using the necessary information. Assuming that cache can store only 10 different domain-IP information in a local device. In order to store 11th one, an old record should be removed. Our cache replacement algorithm removes the cache that is least used. This information is provided through "`int hitNo`", which is incremented by one when the local device uses this information to access a web server without consulting DNS. The record with the minimum `hitNo` is removed to open a room for a new record.

The `Client` class on the other hand, has the following fields:

- `DnsTree root`: This field is for accessing the tree structure. There may be different DNS instances, and so the client should possess the necessary information about the main DNS as soon as it is connected to the Internet.
- `String ipAddress`: Like web servers, each of the clients is assigned with an IP address. This is just for identifying the clients, it is not used for the DNS operations in this project.
- `CachedContent[] cacheList`: The cache of a client is presented with a finite-size array.

Additionally, you need to implement the following methods with the exact format provided:

- `Client(String ipAddress, DnsTree root)`: Constructor method. It initializes the cache list with the size of 10 default.
- `String sendRequest(String domainName)`: It returns the IP address of the requested domain name. If it is available in the cache, it directly returns the corresponding IP address in the cache record. If there are multiple records belonging to the same domain name, the one with the minimum index should be returned. The hitNo should be incremented by 1 if it is fetched from the cache. If it is not available in the cache, then a request should be sent to the

DNS. It is possible that a domain name has multiple IP addresses. Thus, in order to decide on the IP address to return for a specific request, you need to use the Round Robin algorithm. For example, assume that there are 3 different IP addresses of a domain name, namely IP1, IP2 and IP3. The very first request for this domain should be returned with IP1. The second request, independent of the client that is requesting, should be provided with IP2, third request with IP3. Then, the fourth request should return IP1 again, IP2 for the fifth, and so on.

- `void addToCache(String domainName, String ipAddress)`: After obtaining an IP address through sendRequest method, it should be added to the cache by this method. This method should meet the requirements of the cache replacement algorithm described above. If there are two different records with the same domain name, they should be stored as separate records. A cache record is considered as the same with another record only if both domain and IP addresses are identical.

2.3. DnsTree.java

This class represents the main DNS structure. The only field it should have is "`DnsNode root`" which is the root of the tree. Correspondingly, the constructor of this class only creates and initializes the root node.

You should implement the following methods in this class:

- `void insertRecord(String domainName, String ipAddress)`: This method inserts a new record for a given domain name. If a corresponding node is not available in the tree, it should be created and validDomain should be marked as true. Otherwise, the IP address list of the node is updated with this method.
- `boolean removeRecord(String domainName)`: It removes the node with the given domainName from the tree. If successfully removed, return true, otherwise, return false.
- `boolean removeRecord(String domainName, String ipAddress)`: It removes the given ipAddress of a DNS node with the given domainName. If successfully removed, return true, otherwise, return false.
- `String queryDomain(String domainName)`: It queries a domain name within the DNS, and returns the next IP address of the `domainName`, following the Round Robin mechanism. If there is no such domain name available in the tree, this method should return null.
- `Map<String, Set<String>> getAllRecords()`: It should return all the valid domain names in the DNS mechanism with at least 1 IP address. The return type is a Map object whose keys represent the valid domain names, and value is the set of IP addresses of a particular key (domain name).

3. Important Remarks

- The deadline of the project is 05.06.2020 - 23:55. No late submissions are allowed.
- Use appropriate access modifiers, keywords (super, final, static etc.) whenever it is necessary.
- Code documentation is important. You need to document your code in Javadoc style including class implementations, method definitions and field declarations.
- The will be automatically graded, therefore it is important that you follow the rules specified in this document exactly.