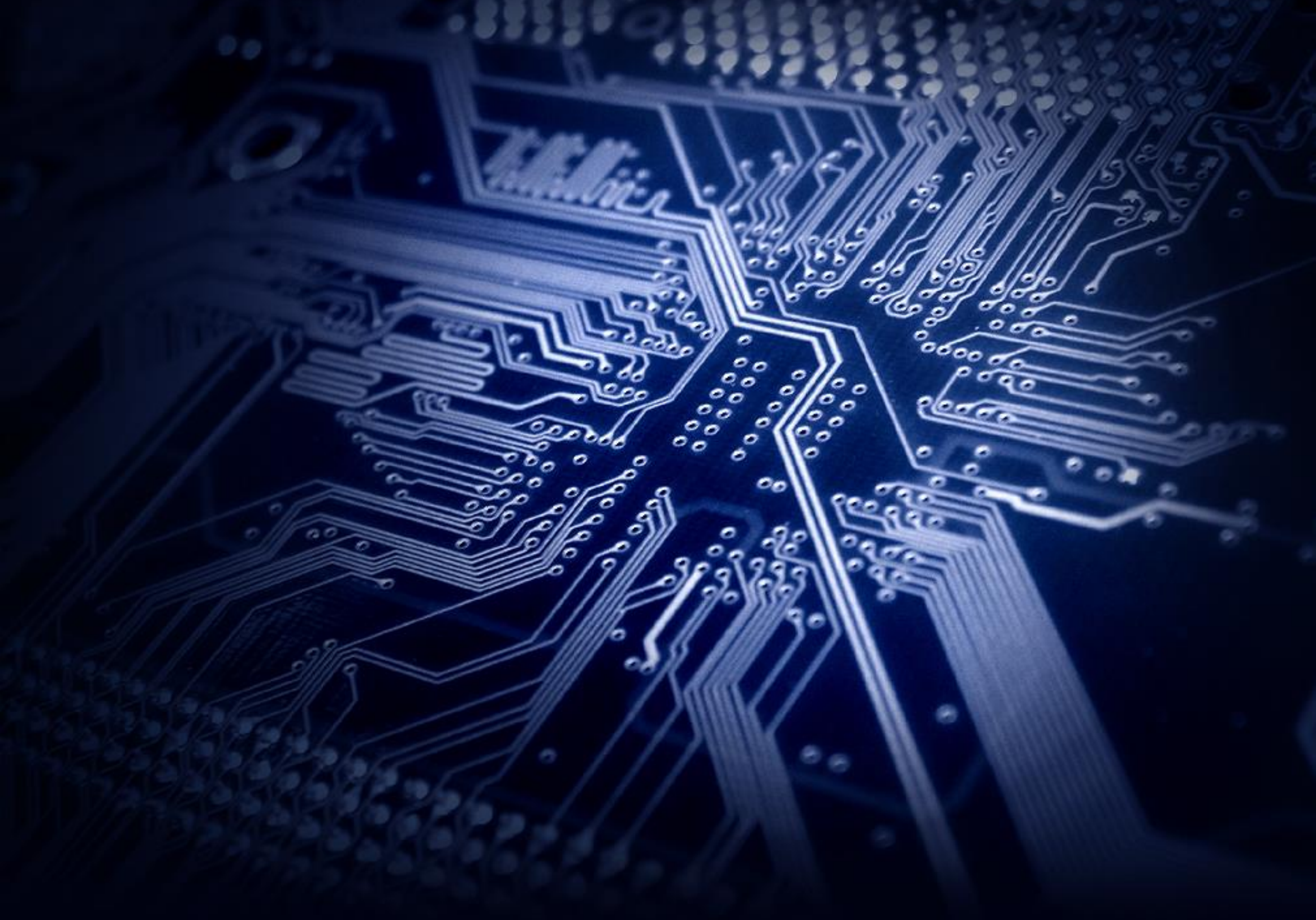

MSBD5017 Blockchain Technology

Decentralized Domain Name System

Report

Group 16
LIANG, Zhuoxian (Business)
NIU, Yutong (Technical)

CONTENT PAGE

1. BACKGROUND

1.1 Domain Name System (DNS)

The domain name system works like a phone book which maps the alphabetic domain names to numeric Internet Protocol addresses. The IP addresses are necessary for locating a webpage. For example, when a user wants to reach the Facebook website, he or she types in "https://www.facebook.com" for the URL, then the DNS would look for the corresponding IP address for the Facebook website which is "69.63.176.13". A domain name is much easier to remember and read by human than the numeric IP address. In order to get an IP address for your website, you need to register an available IP address and find a companies to host the address.

1.2 The Importance of DNS

The domain name system is one of the most important components for the Internet, which ensure smooth and convenient browsing for different webpages. Every device that connected to the Internet has a unique IP address, which is for locating that device and hence communications between devices. Without a DNS translating the domain name into IP address, the devices are unable to communicate with each other and thus you would hardly to reach any webpage on the Internet.

1.3 DNS Attacks

DNS is being attacked frequently since bringing down the DNS often means bringing down the whole services of the companies because users need the DNS to access the services. Therefore, denying DNS is the most direct and easy way to attack the services. According to the 2021 Global DNS Threat Report conducted by Continuity Central, the most common types of DNS attacks faced by the financial industry are listed below.

1.3.1 DNS Spoofing (55 percent of financial institutions)

For DNS spoofing, the attackers try to alter the DNS records to redirect the users to a fraudulent website. It gives a great threat that the confidential information of the users such as account, password, identification maybe stolen if the users enter their personal information into the fraudulent website.

1.3.2 DNS-based Malware (42 percent)

Malware may be installed by hijacking DNS queries. The malwares are also often used to steal the critical and confidential information from the users or even the companies.

1.3.3 Distributed Denial-of-service (DDoS) Attacks (35 percent)

DDoS attackers attempt to overwhelm the targeted server using a large flood of Internet traffic, trying to jam the traffic of the server. As a result, the real users would be prevented from reaching the server normally under the attack.

1.4 Damages Caused by DNS Attacks

According to the 2021 Global DNS Threat Report conducted by Continuity Central during the COVID-19 pandemic, damages caused by each DNS attack across industries are about 950,000 US dollars on average. An average of 7.6 attacks were happened for the companies across industries during that period, and it takes an average of 5.62 hours to mitigate an attack.

For the financial industry, the damages are even more severe. Damages caused by each DNS attack in financial industry are about 1.1 million US dollars on average. An average of 8.3 attacks were happened for the companies in financial industry during that period, and it takes an average of 6.12 hours to mitigate an attack.

It is important and urgent to relieve the damages caused by these DNS attacks because it is not only a great burden for some companies to absorb these costs by themselves, but it also imposes great risks for the leaks of user and company information which may cause more unexpected indirect costs in the future.

2. DNS SERVICE MARKET

2.1 Current Market Size

According to the Managed DNS Services Market – Global Industry Analysis and Forecast Report by Maximize Market Research PVT. LTD., The market size for DNS service is around 350 million US dollars. The largest market is mainly located in North and South America. In 2020, The main service type is premium or advance DNS, but the DDoS protected DNS and Geo DNS services also account for significant proportions.

2.2 Market Potential

Due to the growing trend of digitization across different industries, especially since the COVID-19 pandemic, the anticipated growth in the DNS service market is also substantial, which is estimated to 18 percent per year. And up to year 2027, the market size is expected to be about 1120 million US dollars, which is more than triple of the current size [Figure 1].

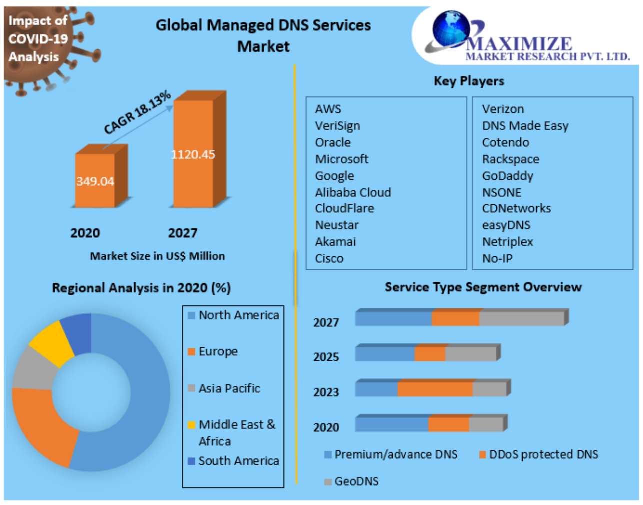

Figure 1 Global DNS service market

Provided with the potential growing number of DNS attacks annually in the future, which could be up to 15.4 million of attacks in year 2023 [Figure 2], and the growing number of daily DNS queries on Google public DNS [Figure 3]. More secured DNS are needed to meet the increasing demands.

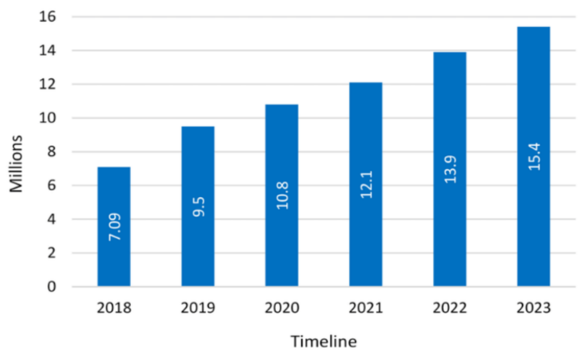

Figure 2 Number of DNS attacks by year

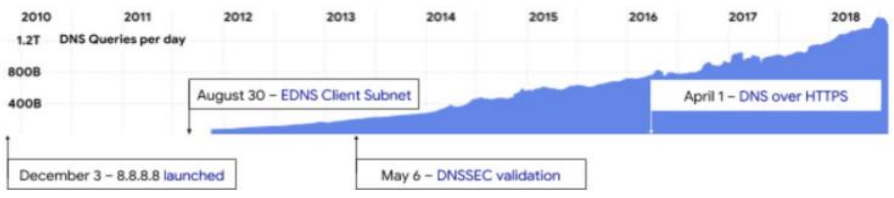

Figure 3 Number of DNS queries on Google public DNS per day

3. PROBLEMS OF CENTRALIZED DNS

There are multiple problems of the centralized DNS to be address, but the major concerns are often the problem of single point of failure, censorship issue and privacy issue.

3.1 Single point of failure

A single point of failure is a non-redundant part of a system, which would cause the entire system to stop working of it fails. Centralized DNS is an example since the services rely on the system would be unavailable if the DNS is offline. It is a huge risk for some companies since they cannot afford their services to go down for even a very short period. Especially for financial industry which needs to provide the clients updated information in real time, even a few seconds of unavailable services could incur huge losses.

3.2 Censorship Issue

The censorship problem of the Internet can be described as government or authorities filtering and blocking content from the users. Common strategies are IP blocking which means blacklisting certain undesired websites, and key word filtering which means blocking the request of URL that contains certain targeted terms. One of the disadvantages of Internet censorship is that it is blocking too much information, not only the harmful information, but also useful information. For example, it is a great disadvantage for scholars or researchers to conduct more comprehensive research since the information they could get is mostly restricted.

3.3 Privacy Issue

When the authorities are given the permission to access the DNS, the information of the users is revealed. Authorities can track the users based on the requests on the DNS. Furthermore, anyone who has access to the DNS user information could sell it for profit. In this case, the users do not have any control of their information on the DNS. A lot of users are more aware of their own privacy, which means they care if they have the ownerships of their own information on the DNS.

4. TECHNICAL DETAILS OF DDNS

4.1 Overview

The technical design for DDNS is to build a distributed system based on the Ethereum network compatible with the current DNS protocol. The design is illustrated by the figure 4.

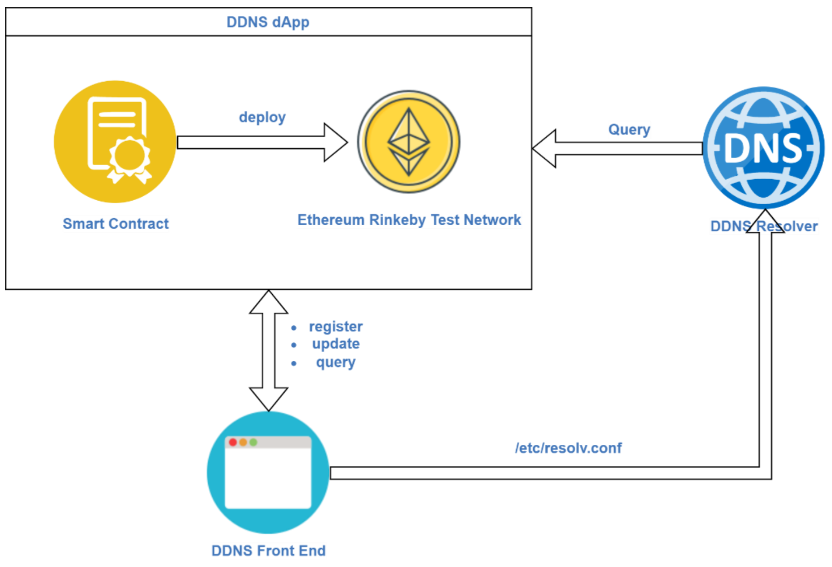

Figure 4 DDNS design

Three components make up a fully compatible DNS service.

4.1.1 DDNS dApp

The first component of DDNS is a dApp deployed to Ethereum network. For testing purposes, we choose Rinkeby Test Network for deployment and solidity version 0.5.0 for compilation. Two smart contracts are built for both Top Level Domain (TLD) and secondary level domains. The dApp provides interface for end users to register a new subdomain, update DNS records and query DNS records.

4.1.2 DDNS Resolver

The resolver is designed as an authoritative DNS server for distributed domains deployed to Ethereum. The resolver is not designed to be distributed. Instead, it leverages traditional server-client model just as the current DNS design. It opens the UDP port 53 and is fully compatible with DNS protocol. This is a trade-off between the compatibility and de-centralized design. The DNS resolver should not be necessary if the resolving mechanism is integrated into local DNS library. But the compatibility will be a challenge. Here, we choose compatibility over de-centralization. The resolver is ready to be deployed as a Docker container.

4.1.3 DDNS Frontend

We also provide a front-end UI interface for the end-users to manage their own domain. The front-end application is implemented with the JavaScript React framework. We try to mimic the functionalities of current DNS management console like Route53 of AWS and GoDaddy websites.

The front-end app is also available as Docker image.

4.2 Smart Contracts

We create three smart contracts: Ownable, Domain and DomainFactory. Ownable is a library contract that manages ownership for a smart contract. Both Domain and DomainFactory contracts inherit Ownable contract. DomainFactory contract deploys a TLD, and Domain contract deploys a secondary level domain.

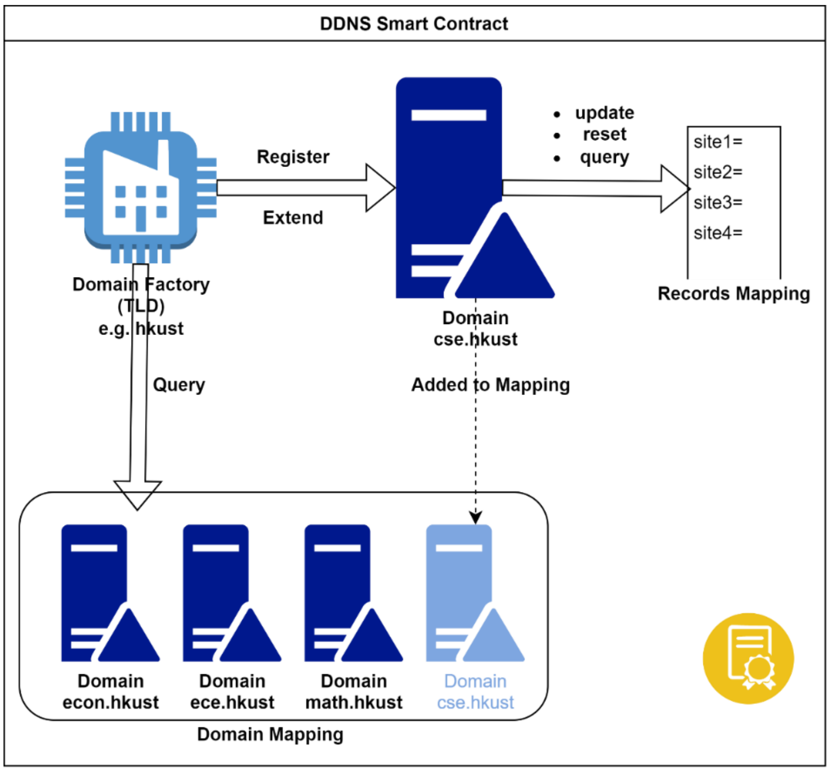

Figure 5 DDNS smart contract

<u>4.2.1 Ownable</u>

The design of the Ownable contract originates from the openzeppelin library. Some modifications are made to remove the unnecessary functions. The contract manages everything related to ownership.

Attributes			
Name	*Type*	*Visibility*	*Description*
_owner	address	private	Stores the owner's address
Constructor			
Inputs	*Modifiers*	*Description*	
Null	Null	Sets the private attribute _owner to tx.origin	
Modifiers			
Name	*Input*	*Description*	
onlyOwner	Null	Requires tx.origin is _owner	
Functions			
Name	*Inputs*	*Outputs*	*Visibility*
Modifiers	*Description*		

isOwner	Null	bool	public
Null	Checks if tx.origin is _owner		
owner	Null	address	public
Null	Gets owner's address		
transferOwnership	address	Null	public
onlyOwner	Transfers ownership to another address		
_transferOwnership	address	Null	private
Null	Transfers ownership to another address		

Table 1

A private attribute _owner is defined to store the owner's address. The constructor sets the owner as the tx.origin. And a modifier onlyOwner restricts only the owner can execute particular functions. Finally, the transferOwnership function transfers the ownership of the smart contract to another address. The Ownable contract is simple by design but useful since both Domain and DomainFactory contracts inherit the Ownable properties.

4.2.2 Domain

The Domain contract mainly supports a database for each secondary level domain. By registering under a TLD, a Domain contract is deployed to Ethereum network. It inherits the Ownable contract and provide interface for updating, resetting and querying DNS records. For simplicity, the DNS record types are not differentiated here. You can simply treat the DNS records as a HashMap from string to another string.

Attributes			
Name	*Type*	*Visibility*	*Description*
tld	string	public	Store the tld
domain	string	public	Store the domain name
domainRecords	bytes32 => string	private	DNS records HashMap
Constructor			
Inputs	*Modifiers*	*Description*	
1. domainName 2. domainTLD	Null	Sets the private attribute domain and tld	
Modifiers			
Name	*Input*	*Description*	
recordsExists	string	Requires a DNS record exists	
Functions			
Name	*Inputs*	*Outputs*	*Visibility*
Modifiers	*Description*		
getRecordHash	string	bytes32	private
Null	Gets record hash by record name		
query	string	string	public
recordExists	Gets record response by name		

reset	string	Null	public
onlyOwner	Resets record by name		
set	string, string	Null	Public
onlyOwner	Sets record from name to value		

Table 2

The main design in the Domain contract is the DNS record HashMap, which acts like a database for DNS records. For avoiding running loop in solidity, we use mapping from the record name to record value for DNS record storage. The advantage of this design is the speed and simplicity, also saving on gas cost when updating the records. The drawback is that we cannot provide a list of records for reference for the user just like the traditional DNS management service does. Also, to prevent the headache of string of variable length, we use the hash value of record name as the key to the HashMap. In this way, the mapping is always between a 32-byte string to a string of variable length. So we declare our domainRecords HashMap as mapping (bytes32 => string). The Domain contract also provides interface to query DNS record. As we may notice, the functions set and reset both requires the transaction is from the smart contract owner. But the query function does not have such restriction, which means that all the DNS queries are public, and we do not support DNS records that are only visible to the owner. This could become a potential improvement on the current design to support both private and public DNS queries.

4.2.3 DomainFactory

The DomainFactory contract acts as a TLD. It provides an interface to register sub-domains under it. Each registration will cost 1 ether, and the expiration of the domain is 365 days. You can choose to extend the expiration date anytime by another 365 days, and it also costs you 1 ether. The contract restricts that both the TLD and the sub-domains should have at least 2 characters.

Structures			
Name	*Components*		
SubDomain	*1.* address domainAddress *2.* uint domainExpiration		
Attributes			
Name	*Type*	*Visibility*	*Description*
COST	uint constant	private	Register cost: 1 ether
EXPIRATION	uint constant	private	Expiration: 365 days
DOMAIN_MIN_LENGTH	uint8 constant	private	Domain >= 2 chars
TLD_MIN_LENGTH	uint8 constant	private	TLD at least 2 chars
topLevel	string	public	TLD name
domainRecords	bytes32 =>SubDomain	private	Records HashMap
Constructor			
Inputs	*Modifiers*	*Description*	
string	isTopLevelLengthAllowed	Initiates a TLD	
Modifiers			

Name	*Input*	*Description*	
isTopLevelLengthAllowed	string	Requires TLD name at least 2 chars	
isDomainNameLengthAllowed	string	Requires domain name at least 2 chars	
isRegistered	string	Requires domain name is registered	
notRegistered	string	Requires domain name is not registered	
collectPayment	Null	Requires payment amount is enough	
Functions			
Name	*Inputs*	*Outputs*	*Visibility*
Modifiers	*Description*		
getDomainHash	string	bytes32	private
Null	Gets domain hash by domain name		
getDomainAddress	string	address	public
isRegistered	Gets domain contract address by domain name		
getDomainExpiration	string	uint	public
isRegistered	Gets domain expiration by domain name		
register	string	Null	public
1. isDomainNameLengthAllowed 2. notRegistered 3. collectPayment	Registers a domain and collect payment		
extend	string	Null	Public
1. isRegistered 2. collectPayment	Extends a domain expiration and collect payment		
query	string, string	string	public
isRegistered	Queries a DNS record		
withdraw	Null	Null	public
onlyOwner	Withdraws the contract balance		

Table 3

Like the Domain contract, the DomainFactory contract also uses a HashMap to store the sub-domain details under the TLD. Each sub-domain records an address pointing to the deployed Domain contract and an expiration date. With the same reason as the Domain contract, the HashMap here also maps the hash value instead of the string of variable length of the domain name. Multiple modifiers are defined here to restrict the behavior of validating the domain name, registering domain, collecting payment, etc. The expiration design is based on the blog by Milen Radkov (https://medium.com/hack/build-a-decentralized-domain-name-system-ddns-dapp-on-top-of-ethereum-86391681c68a).

However, since the design of DDNS in this blog is not quite compatible with the current DNS service provided by AWS Route53 or GoDaddy, only the expiration design is kept. Finally, we provide a withdraw function to allow the TLD owner to withdraw the balance in the TLD smart contract to his/her own wallet.

By combining the functionalities of the three smart contracts above, we can obtain a dApp quite compatible with the current DNS service with distributed design on Ethereum network. The dApp provides interface to allow end users to register, extend a sub-domain under certain TLD and manage the DNS records if they own the sub-domain. The next step is to build and deploy the smart contracts to Ethereum network.

4.2.4 Build and Deploy

We choose the solidity compiler (solc) version 0.5.0. A JS script *({proj_home}/ethereum/compile.js)* is provided to auto-build the smart contracts. The build artifacts are written to the build directory *({proj_home}/ethereum/build/*.json)*. The JSON files include the ABI, Bytecode for each smart contract. After the compilation, the script deploy.js *({proj_home}/ethereum/deploy.js)* can deploy the contract DomainFactory to the Ethereum Rinkeby test network. The provider is using Infura as the gateway for deployment. Since the contract Domain can be created by calling the register method of DomainFactory, only the smart contract DomainFactory is deployed. During deployment, an argument can be passed to define the TLD name. In the deploy script, the TLD "hkust" is assumed. After the deployment, the smart contract address is included in the output in the stdout. Under the project repository, a file with name "ADDRESS" which records the same address retrieved from the deployment script.

4.2.5 Unit Test

The unit test is implemented with the Mocha JavaScript test framework. The tests are empowered by Ganache CLI to run the blockchain network locally before deploying the smart contract to the test network. The test cases span all the functionalities of all three smart contracts we created. The test cases include

- deploys a factory and a domain
- checks ownership of deployed factory and domain
- checks domain name and tld name
- allows set/reset/query domain records
- checks factory tld name
- checks factory domain address getter
- checks factory query
- checks expiration and extension

The test cases are defined in *{proj_home}/test/Domain.test.js.* To run the unit tests, simply *run npm run test* under the project repository home.

4.3 DDNS Resolver

The DDNS Resolver is where we compromise the de-centralized design in trade of better compatibility with the current DNS protocol. We build the DDNS resolver as a standalone Docker container, which opens UDP Port 53 (the same as typical DNS servers) for DNS queries.

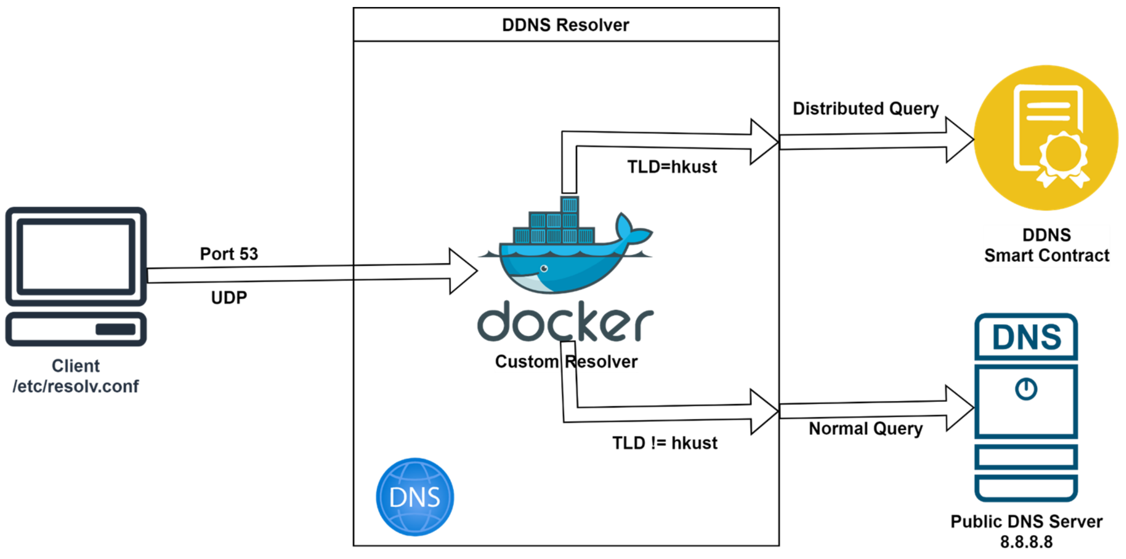

Figure 6 DDNS Resolver

The implementation is purely on JavaScript, and it is dependent on a nodejs package dns2 (https://github.com/song940/node-dns). The resolver checks if the TLD is a traditional DNS domain or a distributed DNS domain. A distributed DNS domain leads the resolver to make queries to the DomainFactory smart contracts, while the traditional DNS domain recursively make queries to the public DNS resolver like 8.8.8.8. The resolver is deployed as a Docker image to a public AWS ECR repository: public.ecr.aws/n0x2w4q4/ddns-resolver.

4.4 DDNS Frontend

The frontend of DDNS mimics the design of DNS management console like AWS Route53 or GoDaddy. It provides the web interface for domain owners to manage their own domains, register or extend new domains, and make queries. The backend of web app is provided by the ABI of the smart contracts. And the web app itself is implemented with the NextJS framework and also deployed as a Docker image to a public AWS ECR repository: public.ecr.aws/n0x2w4q4/ddns-frontend.

5. ADVANTAGES OF DECENTRALIZED DNS

The DDNS can help to address the problems of the centralized DNS mentioned above, which are concern of single point of failure, censorship issue and privacy issue.

5.1 No Single Point of Failure

The Decentralized DNS remove the possibilities of single point of failure because it contains a lot of redundant parts and thus there are multiple points to access the DNS records when using DDNS. The attackers must attack multiple points at the same time to bring down the service, which is expensive, and therefore, it is less likely that they would attack the system. Having a stable and reliable system is essential for companies in the financial industry, so the reliability of the DDNS could be an attractive factor to these companies.

5.2 No Censorship Issue

It is difficult to restrict users from reaching certain contents on the Internet with the DDNS built on blockchain technology. The government or authorities cannot directly intervene the system to block certain contents since there could be a large number of points in the DDNS and the service is based on smart contract which is hard to alter once it is deployed. Users who need to access and collect a wide range of data and information for their jobs might be interested as they can obtain most of the information freely without restrictions.

5.3 No Privacy Issue

Since the data such as the domain names and the request records could be encrypted on the DDNS, it cannot be accessed without any permission from the users. Therefore, it is possible for the users to have own control of data, whether they want to keep it or sell it. The DNS providers and authorities can no longer reach the data and track the users without permission. It could be useful for the companies since some may not want any of their business information being revealed at any means. Individual users may also be benefit if they do not want their records to be read freely.

6. FUTURE DEVELOPMENT PLANS

Although the DDNS could solved some major problems of the centralized DNS and seems attractive, the development of the system is still on early stages whether it is on business or technical perspective, and the cost is one of the major concerns. Plans for the future development of the DDNS are listed below.

6.1 DDNS as the Supplementary System

The traditional DNS is dominating currently as almost all DNS in the market are centralized, and therefore, it is infeasible to replace the DNS in a short period using DDNS. However, it is possible to use the DDNS as a backup system for the centralized DNS due to the stability. When the DNS is being attacked, the DDNS would serve as a secondary pathway where the services could be maintained even if the DNS goes down.

6.2 Browser Support for DDNS

Currently, since DDNS on blockchain is not common, browsers, emails, and other Internet based application do not support blockchain DDNS. In order to use the blockchain DDNS, users may add custom DNS servers or resolvers in local. Therefore, there is still a way to go for the browser compatibility issues with the blockchain DDNS.

6.3 Support for more DNS Record Types

For our current development of the DDNS, there are only a few types of DNS records are supported in the system. However, there are many record types for the DNS such as A Record, AAAA Record, CNAME Record, MX Record, NS Record, PTR Record, CERT Record, SRV Record, TXT Record, SOA Record, etc. For the completeness of the blockchain DDNS, it must at least support all of the above DNS record types which are commonly supported by centralized DNS.

7. SOURCE CODE

Source code is available on GitHub: https://github.com/yutong-niu/msbd5017-proj