Introduction to ICON

ICON foundation

ICON Basics

ICON foundation

Introduction to ICON

**ICON Basics**

- 1. Introduction to ICON - 2. Challenges and Solutions - 3. Architecture and Components

ICON Quickstart

- 1. Wallet and Tracker - 2. T-Bears - 3. SCORE

Development Resources

1. Introduction to ICON

ICON - new generation blockchain

1st generation : Bitcoin and the simple alt-coins (eg. litecoin)[2009]

Programmable Smart Contract

2nd generation : Ethereum. Turing complete Smart Contract [2015]

Performance Enhancement

3rd Generation : EOS, AION, Zilliqa,

ICON

[2017] ...

History

Vision

**Hyperconnect the World**

Build a community of communities based on the consensus to our governance and protocol

ICON Characteristic

- Delegated proof of contribution (DPoC)

- One confirmation

- 1000+ TPS - Multi channel

- 1000+ TPS per channel - Low / flexible transaction fee

- Interchain

- Native python code Smart Contract + JVM

ICON Characteristic

**- Delegated proof of contribution (DPoC)**

- One confirmation

- 1000+ TPS - Multi channel

- 1000+ TPS per channel - Low / flexible transaction fee

- Interchain

- Native python code Smart Contract + JVM

ICON Governance

- DPoC - IISS

ICON Governance - DPoC

- Delegated Proof of Contribution

- ICONist : each participant who uses the ICON network

- ICON Network evaluates all ICONist according to their contribution to the network (I-score).

- ICONist can delegate its I-score (proof of contribution)

- Representative is elected by the sum of delegated I-score - Representatives

-

Verify and generate blocks

- Participate in governance policy decision making by voting

ICON Governance - IISS

System for Proof of Contribution

- ICON Incentives Scoring System

- Weighted sum of level of contribution for each evaluation items.

- Example of evaluation items

- Amount of ICX

- Transaction amount

- Block generation and verification

- DApp generation and usage - Incentives

- Additionally issued / deferred ICX distribution

- Voting rights. Can be delegated to the C-Rep/ P-Rep

Contribution level

Weight

Increase rate. Higher score for fast growing activities

ICX Station - Global Accelerator Program

- Hubs in major cities around the world

- San Francisco, Seoul, Japan, Singapore, and more coming

- Mission

- Accelerate projects pushing the boundaries of real-world adoption of blockchain technology

using the ICON Protocol.

- Full stack support

- tech consultancy, product development, initial funding, community promotion, etc.

Token economics and whitepaper review

Pitch deck preparation and review

Global Demo Day

Introduction to preferred service providers with discounted rates

Concept - Terms

- LFT -

Loop Fault Tolerance Consensus Algorithm

- Low traffic

- Leader spinning

2. Challenges and Solutions

Introduction to ICON

**ICON Basics**

- 1. Introduction to ICON - 2. Challenges and Solutions - 3. Architecture and Components

ICON Quickstart

- 1. Wallet and Tracker - 2. T-Bears - 3. SCORE

Development Resources

Challenges

- Performance and scalability

- Ethereum 20 TPS v.s., PayPal 193 TPS, Visa 1667 TPS - Transaction fee

- Avg. transaction fee - 0.224 USD

- Median transaction fee - 0.085 USD - Interoperability

- Number of blockchains and distributed ledger networks has exploded

2.1. Performance and Scalability

Performance and Scalability - Common Approaches

- State Channel/ Off-chain (Raiden network in Ethereum)

- Side-chain (Plasma in Ethereum)

- Sharding - State is split into shards

Our Approaches

- DPoC (Delegated Proof of Contribution) - improved from DPoS

- One confirmation finality

- 1,000 TPS

- Sharding

- State is split into shard

- Multichannel

- Easy to scale out

- 1,000 TPS per channel

Shard + Multichannel

**SCORE**

Multi-Channel is implemented where each SCORE state is independent of each other

**ICX**

sharding by #address

**Ch dApp**

**#1 A**

**dApp B**

**dApp D**

**dApp C**

. . .

. . .

**Ch #n**

**Ch #n**

**Ch #2**

**Ch #2**

**Ch #1**

hx...01 hx...11 hx...21 hx...31 ...

hx...02 hx...12 hx...22 hx...32 ...

**1-a. Checked as ‘updating’**

**2. Update separately**

**1-b. Wait until updating ends**

2.2. Fees

Transaction Fee in other systems

- Bitcoin

-

Only Used in reward for Mining

- No more extra fees to additional function call by Bitcoin script language

- User pays 100%

- Ethereum

- Only Used in reward for Mining

- Extra fees to additional function call that uses computing resources

- User pays 100%

Transaction Fee in ICON

- Flexible Fee Structure

- User pays 100% (Traditional)

- Fee shared by operator and users

- Virtual Step - around 1% monthly interest from staking

Transaction Fee in ICON

- Step

- Unit of transaction fee (

100,000,000

Step is equivalent to 1 ICX) - Function call, DB usage, Input

param size

- Step = max ( [ ( ∑ β

i

S i

+ C ), C ] )

Step Costs programmed in Governance

"default": "0x186a0", "contractCall": "0x61a8", "contractCreate": "0x3b9aca00", "contractUpdate": "0x5f5e1000", "contractDestruct": "-0x11170", "contractSet": "0x7530", "get": "0x0", "set": "0x140", "replace": "0x50", "delete": "-0xf0", "input": "0xc8", "eventLog": "0x64", "apiCall": "0x0"

**Name Description**

S contractCall Number of times to call the smart contract function

S contractCreate Number of times to call the smart contract code generation function

S contractUpdate Number of times to call the smart contract code update function

S contractDestruct Number of times to call the smart contract code delete function

S contractSet Size of generated/updated smart contract code (Bytes)

S set Size of data newly set in the state database (Bytes)

S replace Size of data to be updated in the state database (Bytes)

S delete Size of data to be deleted in the state database (Bytes)

S input Size of input data included in transaction (Bytes)

S eventLog Size of event log as a result of transaction (Bytes)

C Default value that is charged each time transaction is executed

Step cost Operation that consumes step.

**Name Value**

β contractCall 25,000

β contractCreate 1,000,000,000

β contractUpdate 1,600,000,000

β contractDestruct -70,000

β contractSet 30,000

β set 320

β replace 80

β delete -240

β input 200

β eventLog 100

C 100,000

**Name Description**

S contractCall Number of times to call the smart contract function

S contractCreate Number of times to call the smart contract code generation function

S contractUpdate Number of times to call the smart contract code update function

S contractDestruct Number of times to call the smart contract code delete function

S contractSet Size of generated/updated smart contract code (Bytes)

S set Size of data newly set in the state database (Bytes)

S replace Size of data to be updated in the state database (Bytes)

S delete Size of data to be deleted in the state database (Bytes)

S input Size of input data included in transaction (Bytes)

S eventLog Size of event log as a result of transaction (Bytes)

C Default value that is charged each time transaction is executed

Step cost Operation that consumes step.

**Name Value**

β contractCall 25,000

β contractCreate 1,000,000,000

β contractUpdate 1,600,000,000

β contractDestruct -70,000

β contractSet 30,000

β set 320

β replace 80

β delete -240

β input 200

β eventLog 100

C 100,000

**Name Description**

S contractCall Number of times to call the smart contract function

S contractCreate Number of times to call the smart contract code generation function

S contractUpdate Number of times to call the smart contract code update function

S contractDestruct Number of times to call the smart contract code delete function

S contractSet Size of generated/updated smart contract code (Bytes)

S set Size of data newly set in the state database (Bytes)

S replace Size of data to be updated in the state database (Bytes)

S delete Size of data to be deleted in the state database (Bytes)

S input Size of input data included in transaction (Bytes)

S eventLog Size of event log as a result of transaction (Bytes)

C Default value that is charged each time transaction is executed

Step cost Operation that consumes step.

**Name Value**

β contractCall 25,000

β contractCreate 1,000,000,000

β contractUpdate 1,600,000,000

β contractDestruct -70,000

β contractSet 30,000

β set 320

β replace 80

β delete -240

β input 200

β eventLog 100

C 100,000

Step Calculation

- How step is calculated

- Step = max ( [ ( ∑ β

i

S i

+ C ), C ] )

- e.g., Deploy 512 Bytes SCORE - max((512 \* 30,000 + 1,000,000,000 + 100,000), 100,000) = 1,015,460,000 step = 10.1546 ICX

contractSet contractCreate default

- e.g., Invoke a query method - max((25,000 + 100,000), 100,000) = 125,000 step = 0.00125 ICX

contractCall default

-

Yellow paper : https://icon.foundation/resources/file/ICON\_Yellowpaper\_Transactionfee\_EN\_V1.0.pdf

Transaction Fee

- Fee = usedStep \* stepPrice

- stepPrice = ICX exchange rate

- If (usedStep > stepLimit), then tx fails and stepLimit is deducted.

- If (your balance < stepLimit \* stepPrice), then tx fails immediately.

- Governance SCORE

- Address - cx0000000000000000000000000000000000000001

- getStepCost

- getMaxStepLimit

- getStepPrice

2.3. Interoperability

What is Blockchain Interoperability

- Value transfer

- Transfer coins or tokens from one blockchain to another

- Eg. Decentralized exchange - Service invocation

- Execute the smart contract on other blockchain and share the execution result

- Eg. Public blockchain interacts with the enterprise/ government blockchain services - Scalability

- Horizontal scalability solution

- Similar to side chain approach

Existing Approach - Interoperability

- Two-way pegging

- Side Chain

- SPV(Simplified Payment Verification) required - Atomic Swap

- Exchange asset Atomically

- Should use same Hash Algorithm to exchange - Relayer

- Relay information to other Blockchain

- Example : BTC Relay

Example : BTC Relay

• Off-chain solution

• Relayer gets commission for msg delivery

• DEX Example

1. Locks ETH to smart contract

2. Relayer pushes locking evidence to

BTC script

3. BTC script sends BTC to receiver

4. Relayer pushes proof to smart contract

5. Smart contract sends ETH to receiver

BTC relay SPV client

**1 2**

**3**

**4**

**5**

ICON - BTP (Blockchain Transfer Protocol)

- Nexus coordinates interchain requests

- C-Rep in ICON is responsible for

relaying BTP messages

- All interchain requests are processed

atomically

- Influenced by cross-shard studies

- OmniLedger : a secure, scale-out

decentralized ledger via sharding

- ChainSpace : scalable sharding for smart

contracts

light client B

light client B

Light Client Channel B light

client B

light client B

Blockchain B

light client A

Blockchain A light

client A

Light Client Channel A Light

client A

light client A

Node

Blockchain A Node

ICON Nexus Node

Blockchain B

Node

BTP Process

Nexus

Chain A

Chain B

Simulating BTP Request & Register BTP Requests

BTP Request

Lock State

BTP Request

Lock State

BTP Response

BTP Response

Register Locking Evidence

BTP Commit

Commit State BTP Commit

Commit State

BTP Features

- High level transaction isolation

- Lock state in each chain - All interchain requests consume the same amount of time

- BTP request simulation in Nexus

- Concurrent verification on multiple chains - Timeout is measured in block height of Nexus

Example : Real Estate Trading (Traditional Relayer)

Realestate Blockchain

Nexus Blockchain

Government Blockchain

Kofla Blockchain

Bank Blockchain

Example : Real Estate Trading (BTP)

Realestate Blockchain

Nexus Blockchain

Government Government Blockchain Blockchain

Kofla Blockchain

Bank Blockchain

BTP Roadmap

- BTP 0.5

- Token exchange with other blockchains - BTP 1.0

- Transaction simulation by Nexus node

- Interoperate with private chains - BTP 2.0

- State to be “yanked” to state relay chain

- Temporary commits on state channel

- Process 1M BTP requests in off-chain commit

BTP 2.0

- Interchain related states are yanked to the Nexus state channel

Nexus

Register BTP Requests

BTP Request State

Simulating

Temporary Channel

BTP

Commit

BTP Yank

BTP Yank Result

Chain

Yank state to state channel

Chain

Yank state to state channel

Register Locking Evidence

BTP Response

Temporary Commit

BTP Commit

Commit State

Commit State

3. Architecture and Components

Introduction to ICON

**ICON Basics**

- 1. Introduction to ICON - 2. Challenges and Solutions - 3. Architecture and Components

ICON Quickstart

- 1. Wallet and Tracker - 2. T-Bears - 3. SCORE

Development Resources

Architecture and Components

- C-rep (Community Representative)

- P-rep (Public Representative)

- Citizen node

- T-Bears

- SDK

- Wallet

- Tracker

Architecture and Components

- C-rep

- Community Representative

- Member of governance channel - P-rep

- Public Representative

- Member of governance channel - Citizen node

- Member of ICON nodes

Architecture and Components

- T-Bears

- SDK

- Wallet

- Tracker

Wallet

Tracker

SDKs

T-Bears

**ICON network**

Community Network Topology & Ecosystem

ICON Network C-Rep Tracker

P-Rep

P-Rep

SDKs

T-Bears

P-Rep

Citizen

Citizen

Citizen

Nexus

Wallet

LB

Citizen

Node View

ICON Node

- IconService

- SCORE execution environment

- ICX base coin management

- Transaction fee calculation

- Loopchain

- Peer management

- Block management

- LFT consensus engine

Node View

ICON Node

- JSON RPC

- Request

1. jsonrpc

2. method

3. parmas

4. id - Response

1. jsonrpc

2. result

3. error

4. id

Emulated node environment. Mimic block generation without consensus.

T-Bears

ICON Quickstart

ICON foundation

1. Wallet and Tracker

Introduction to ICON

ICON Basics

- 1. Introduction to ICON - 2. Challenges and Solutions - 3. Architecture and Components

**ICON Quickstart**

- 1. Wallet and Tracker - 2. T-Bears - 3. SCORE

Development Resources

Wallet

- ICON chrome wallet extension ICONex

- ICON Android Wallet app

- ICON iOS Wallet app

Wallet

Explore basic functions of Chrome Wallet and learn how to connect to different networks.

- Install Chrome Wallet

- Make your own wallet on ICONex

- Import the keyfile

- Connect to testnet

- Require testnet ICX for test

- Check your balance

- Send transfer Tx

Tracker

Through tracker, explore blocks, transactions, and the contracts deployed on the network. View and download contract’s source code.

- Mainnet Tracker : https://tracker.icon.foundation

- Testnet Tracker : https://bicon.tracker.solidwallet.io

- Block explorer

- TX explorer

- Contract explorer

Wallet - 1/6

Install Chrome Wallet

Cherome Extension ICON Wallet

[

https://chrome.google.com/webstore/detail/iconex/flpiciilemghbmfalicajoolhkkenfel/related?hl=kr]

Wallet - 2/6

Make your first wallet on ICONex

1. Input Wallet Name, Password

a. At least, 8 characters including letters, Numbers and special characters 2. Download Keystore file

3. Copy Private Key & Save the Key Safely

Wallet - 3/6

Import the keyfile

1. Add Wallets 2. Check Load Wallet 3. Select Wallet file

a. Can load wallet by private key too 4. Enter Password

Wallet - 4/6

Connect to testnet

1. Choose Chrome Developer menu

2. Click Application Tab

3. Add new Key “isDev”

4. Set isDev value to “true”

5. Select network Mainnet to

testnet

Wallet - 5/6

Require testnet ICX for test

- Mail to testicx@icon.foundation

Check your balance

Wallet - 6/6

Send transfer Tx

1. Enter Password

2. Enter transfer amount of ICX, Data(Optional), Receiving Address

Tracker - 1/4

- Addresses explorer

Tracker - 1/4

- Address

Tracker - 2/4

- Contracts explorer

Tracker - 2/4

- Contract

Tracker - 3/4

- Blocks explorer

Tracker - 3/4

- Block

Tracker - 4/4

- Transactions explorer

Tracker - 4/4

- Transaction

2. T-Bears

Introduction to ICON

ICON Basics

- 1. Introduction to ICON - 2. Challenges and Solutions - 3. Architecture and Components

**ICON Quickstart**

- 1. Wallet and Tracker - 2. T-Bears - 3. SCORE

Development Resources

Installing T-Bears - Docker

**Why Docker?**

Installing T-Bears - Docker

- $ docker pull iconloop/tbears

- $ docker create --name local-tbears -p 9000:9000 -it

iconloop/tbears

- $ docker container start local-tbears

- $ docker container ls -a

- $ docker container attach local-tbears

- root@07dfee84208e:/tbears# exit

Installing T-Bears - Docker

- $ docker run -it --name local-tbears -p 9000:9000

iconloop/tbears

- root@07dfee84208e:/tbears# exit

- $ docker container ls -a

T-Bears : Test Account

- Keystore file : keystore\_test1 - Password : test1\_Account

{

"address": "hxe7af5fcfd8dfc67530a01a0e403882687528dfcb", "crypto": { .... }, "id": "e2ca66c6-b8de-4413-82cb-52c2a2200b8d", "version": 3, "coinType": "icx" }

T-Bears : Check your Account Balance

- $ tbears balance [address]

# tbears balance hxe7af5fcfd8dfc67530a01a0e403882687528dfcb balance in hex: 0x2961fff8ca4a623277ff566 balance in decimal: 800459999999999999999997286

T-Bears : Send Queries

- $ tbears totalsupply

- $ tbears lastblock

- $ tbears blockbyheight 0x1

Testnet : Create an Account

- $ tbears keystore [file\_path]

- Examine the keystore file

{

"address": "hxe7af5fcfd8dfc67530a01a0e403882687528dfcb", "crypto": {

.... }, "id": "e2ca66c6-b8de-4413-82cb-52c2a2200b8d", "version": 3, "coinType": "icx" }

Testnet : Account & Balance

- We will use the same keystore file

created before

- Same address, but different accounts

- To receive test ICX, send email to

testicx@icon.foundation with following information

- Testnet node url

- Address to receive the testnet ICX

- Faucet : http://52.88.70.222/

Name Yeouido (여의도)

**Node https://bicon.net.solidwallet.io**

API endpoint https://bicon.net.solidwallet.io/api/v3

Network ID (nid) 3

Tracker https://bicon.tracker.solidwallet.io

Transaction fee on

SCORE audit off

Testnet : Send Queries

- $ tbears balance [account] -u https://bicon.net.solidwallet.io/api/v3

- $ tbears totalsupply -u https://bicon.net.solidwallet.io/api/v3

- $ tbears lastblock -u https://bicon.net.solidwallet.io/api/v3

- $ tbears blockbyheight 0x0 -u https://bicon.net.solidwallet.io/api/v3

3. SCORE

Introduction to ICON

ICON Basics

- 1. Introduction to ICON - 2. Challenges and Solutions - 3. Architecture and Components

**ICON Quickstart**

- 1. Wallet and Tracker - 2. T-Bears - 3. SCORE

Development Resources

3.1. Smart Contract Basics

Smart contract Basics

- What is smart contract?

- SCORE(Smart Contract On Reliable Environment)

- What is SCORE?

- Use Python (Easy to learn)

3.2. T-Bears Commands for SCORE Development

Basic commands for SCORE development

- init

$ tbears init -h

$ tbears init project test\_SCORE

usage: tbears init [-h] project scoreClass

test\_tbears\_init

Initialize SCORE development environment. Generate <project>.py and

├── keystore\_test1 package.json in <project> directory. The name of the score class is <scoreClass>.

├── project │ ├── \_\_init\_\_.py positional arguments:

project Project name scoreClass SCORE class name

│ ├── package.json │ └── project.py

optional arguments:

-h, --help show this help message and exit

├── tbears\_cli\_config.json └── tbears\_server\_config.json

Basic commands for SCORE development

- scoreapi

$ tbears scoreapi -h

usage: tbears scoreapi [-h] [-u URI] [-c CONFIG] address

Get score's api using given score address

positional arguments:

address Score address to query score api

optional arguments:

-h, --help show this help message and exit -u URI, --node-uri URI

URI of node (default: http://127.0.0.1:9000/api/v3) -c CONFIG, --config CONFIG

Configuration file path. This file defines the default value for the "uri" (default: ./tbears\_cli\_config.json)

Basic commands for SCORE development

- call

$ tbears call -h

usage: tbears call [-h] [-u URI] [-c CONFIG] json\_file

Request icx\_call with user input json file.

positional arguments:

json\_file File path containing icx\_call content

optional arguments:

-h, --help show this help message and exit -u URI, --node-uri URI

URI of node (default: http://127.0.0.1:9000/api/v3) -c CONFIG, --config CONFIG

Configuration file path. This file defines the default value for the "uri" (default: ./tbears\_cli\_config.json)

Basic commands for SCORE development

- sendtx

$ tbears sendtx -h

usage: tbears sendtx

optional arguments: [-h] [-u URI] [-k KEYSTORE] [-c CONFIG] [-p PASSWORD]

-h, --help show this help message and exit json\_file

-u URI, --node-uri URI

URI of node (default: http://127.0.0.1:9000/api/v3) -k KEYSTORE, --key-store KEYSTORE

Keystore file path. Used to generate "from"address and Request icx\_sendTransaction with user input json file and keystore file. If

transaction signature keystore file is not given, tbears sends request as it is in the json file.

-c CONFIG, --config CONFIG

Configuration file path. This file defines the default positional arguments:

value for the "uri" (default: json\_file File path containing icx\_sendTransaction content

./tbears\_cli\_config.json) -p PASSWORD, --password PASSWORD

Keystore file's password

Basic commands for SCORE development

- txresult

$ tbears txresult -h

usage: tbears txresult [-h] [-u URI] [-c CONFIG] hash

Get transaction result by transaction hash

positional arguments:

hash Hash of the transaction to be queried.

optional arguments:

-h, --help show this help message and exit -u URI, --node-uri URI

URI of node (default: http://127.0.0.1:9000/api/v3) -c CONFIG, --config CONFIG

Configuration file path. This file defines the default value for the "uri" (default: ./tbears\_cli\_config.json)

Basic commands for SCORE development

- deploy

$ tbears deploy -h

usage: tbears deploy [-h] [-u URI] [-t {tbears,zip}] [-m {install,update}] [-f FROM] [-o TO] [-k KEYSTORE] [-n NID] [-p PASSWORD] [-s STEPLIMIT] [-c CONFIG] project

Deploy the SCORE

positional arguments:

project Project directory path or zip file path

optional arguments:

-h, --help show this help message and exit -u URI, --node-uri URI

URI of node (default: http://127.0.0.1:9000/api/v3) -t {tbears,zip}, --type {tbears,zip}

This option is deprecated since version 1.0.5. Deploy command supports zip type only

-m {install,update}, --mode {install,update}

Deploy mode (default: install) -f FROM, --from FROM From address. i.e. SCORE owner address -o TO, --to TO To address. i.e. SCORE address -k KEYSTORE, --key-store KEYSTORE

Keystore file path. Used to generate "from" address and transaction signature -n NID, --nid NID Network ID -p PASSWORD, --password PASSWORD

keystore file's password -s STEPLIMIT, --step-limit STEPLIMIT

Step limit -c CONFIG, --config CONFIG

deploy config path (default: ./tbears\_cli\_config.json)

Hands-on Training

- Create a SCORE

- Modify the SCORE

- Deploy the SCORE on T-Bears

- Execute the SCORE functions

- Expected output

“hello, [your name]. It’s [score name]”

Summary

- Basic commands for SCORE development

- init : Initialize T-Bears project

- scoreapi : Get score’s api using given score address

- call : Request icx\_call with user input json file

- sendtx : Request icx\_sendTransaction with user input json file and keystore file.

- txresult : Get transaction result by transaction hash

- deploy : Deploy the SCORE - https://github.com/icon-project/t-bears/blob/master/README.md

-

3.3. One Step Further

Assignment - Your First SCORE

Create a new project template. Implement a feature.

- $ tbears init [projectName][scoreName]

- Make it accept ICX

- Greedy Hello World

Assignment - Greedy Hello World

- Releasing Sample Code on the Last Day

- Can get Bonus via implementing SCORE to accept ICX

- Deploy the implemented SCORE to Testnet.

- Send us an email including your team name & SCORE Address

- Check the result by transferring ICX to deployed SCORE

Development Resources

Introduction to ICON

ICON Basics

- 1. Introduction to ICON - 2. Challenges and Solutions - 3. Architecture and Components

ICON Quickstart

- 1. Wallet and Tracker - 2. T-Bears - 3. SCORE

**Development Resources**

Development Resources

- GitHub

- Developer Portal

- ICON official Documentation Project

- ICON Improvement Proposal

GitHub

https://github.com/icon-project

- loopchain

- icon-service

Node

- icon-rpc-server

- t-bears

- icon-sdk-python

Dev tools - icon-sdk-java

- iconex\_android

- iconex\_ios

- iconex\_chrome\_extension

Developer Portal

https://www.icondev.io

- Community portal for ICON DApp ecosystem

Getting Started Tutorials for developers to get started

SCORE Details on ICON’s Smart Contract, SCORE

Community Forum for Korean/English developers to discuss and communicate

DApp Overview of ICON DApp Partners

Documentation Project

https://icon-project.github.io

- GitHub Pages

- ICON overview - Network info - Account management - Client SDKs - SCORE development - Mainnet SCORE audit guideline - Tutorials with sample codes

- We welcome contributors !

- Read CONTRIBUTING.md - French / Chinese translation provided

by volunteer contributors.

ICON Improvement Proposal

https://github.com/icon-project/IIPs

- IIP describes a standard for

ICON platform.

- Anyone can prompt suggestions

and discussions on new functions or improvement.

- Selected items will be

implemented on ICON network.