# Improving Openstack Tacker's Accountability Using Blockchain Technology

Yassine Jebbar

Centre d'Excellence en Technologies de l'Information et de la Communication

July 6, 2017

## Internship's Context

- Final year internship at Cetic.
- 5 Months duration: 3 months in Morocco, 2 months in Cetic.
- **Objective:** Secure Service Function Chaining using Blockchain technology.

# Outline

## Laying The Groundwork

- Cloud Computing & Virtualization
- SDN & NFV
- Service Function Chaining
- The Blockchain Technology

## Improving Tacker's Accountability Using Blockchain

- Probelm Space Definition
- Openstack Tacker
- Logs Immutabilization
- Blockchain for Record-Keeping
- Analysis & Conception
- Implementation
- Conclusions

## Perspective & Future Work

# Cloud Computing & Virtualization

Your Connection to ICT Research

- Computing paradigm that relies heavily on outsourcing of (especially) physical resources.
- Abstraction of software from hardware
- Several advantages in combining Cloud Computing and Virtualization (Performance, Scalability, High Availability, Pay per use,...etc)

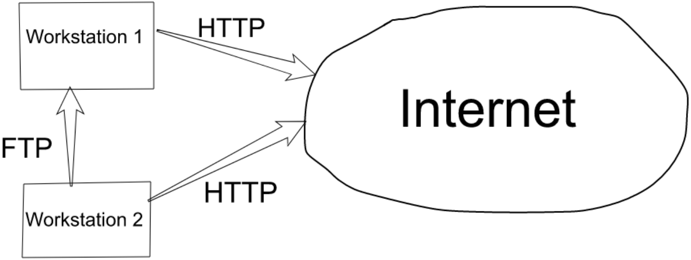
## Software Defined Networks

- Traditional networks are rigid and offer little flexibility.

- ▶ Make networks more ”programmable”.

- ▶ Abstract networking technology/protocols (control plane) from networking hardware (forwarding plane).

# Example



## Network Functions Virtualization

Your Connection to ICT Research

- Various networking hardware provided by different vendors (vendor lock-in issue).

- Specific hardware for each networking function (Firewall, NAT, DNS,...etc)

- ▶ Provide common hadrware platform for the different networking functions, and make it manageable from computing devices (servers, laptops, mobile,...etc)

## SDN & NFV Upgrades

- Central Management
- Direct Programmability
- Flexibility
- Vendors-Neutral

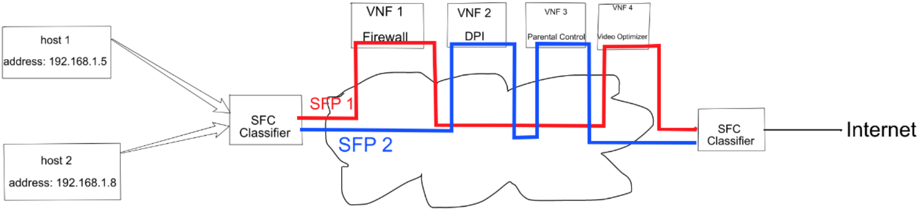
# Service Function Chaining

- Improved flexibility in networking functions.
- Instead of building a rigid networking system, instances of simple networking functions (NAT, DNS, DPI,...etc) are composed and "chained" together as required to form more complex service functions.
- SFC offers low complexity, high maintenance ability and improved adaptability.

## SFC Main Components

- **Service Function (SF)** A virtual instance of a basic networking function (Firewall, DPI, NAT, DNS,...etc)
- **Service Function Forwarder (SFF)** Logical component that forwards the incoming packets to SFs according to the information carried in the SFC encapsulation.
- **Service Function Path (SFP)** The actual path of a packet chosen by the SFF after taking into account the network's constraints (network load, availabilty of SF instances,...etc)
- **SFC Metadata** Exchanged context information among the different SFC components.

# SFC Components (Example)



- if source ip == 192.168.1.5 then forward following SFP 1
- if source ip == 192.168.1.8 then forward following SFP 2

# The Blockchain Technology

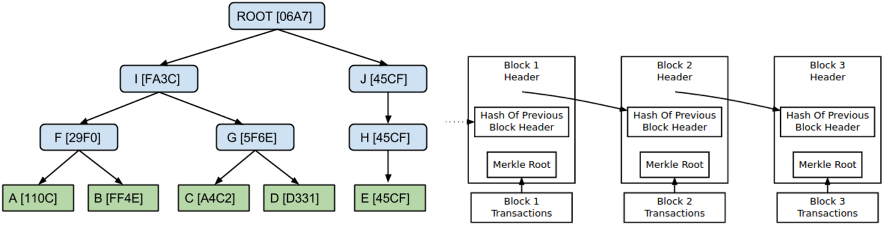
Your Connection to ICT Research

- Introduced to the IT field as the technology behind the Bitcoin cryptocurrency in 2009 by Satoshi Nakamoto.
- Blockchain is basically a massive distributed database (i.e: distributed ledger) where transactions between different peers are being recorded.
- Blockchain use cases are being expanded to other fields beside cryptocurrencies (Health care, copyrights,...etc)

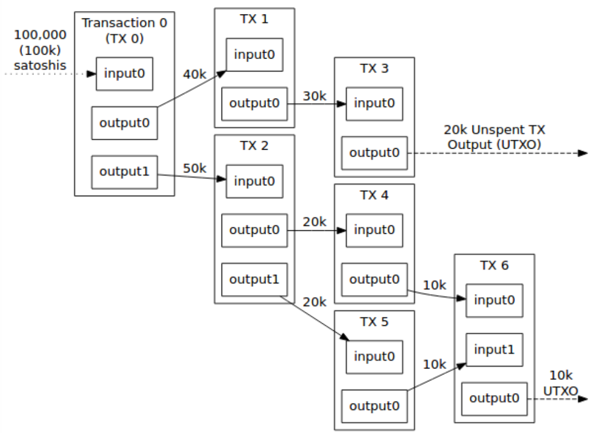
# Why The Blockchain Is Different (1)

- Blockchain solves the "double spending" issue in digital transactions.
- Spares the need for central "trust" authority in transactions.
- Complex cryptographic computations and a huge network (7692 full node on average in July 2017).

# Why The Blockchain Is Differrent (2)



# Why The Blockchain Is Differrent (3)



# Problem Space Definition

cetic
Your Connection to ICT Research

- How secure is the SFC architecture ?
- How can the Blockchain make a system more secure ?
- Is there any use case where the Blockchain can improve SFC's security ?

## Problem Space Definition: The 3 A's

- Systems' security is generally evaluated following 3 criteria:
  - Authentication
  - Authorization
  - Accountability

# Openstack Tacker

Your Connection to ICT Research

- New component (v 0.7 currently) in the IaaS provider Openstack.

- Tacker is released as the platform's main component for managing VNFs instances in a virtualized environment.

- Tacker relies on Openstack's other components (Keystone, Heat,...etc) to orchestrate and provide the required infrastructure for the VNFs (see NFV MANO framework).

## Problem Space Considerations

- Tacker still facing major issues in its setup as well as providing its use cases.
- The availabe NFV platforms still don't adhere completely to the NFV MANO framework.
    - Problem space adaptation considering the available tools and Blockchain's strength: Improving Openstack Tacker's accountability.

# Logs Immutabilization

- To make Tacker more "auditable", we ensure the log files authenticity in terms of content.

- We rely on Blockchain's strength in keeping records to store a proof about the logs' content.

- Log files must be secured before storing their proof of authenticity in the Blockchain.

# Logs Immutabilization Mechanisms (1)

- We use logs immutabilization mechanisms used in a previous research work (Distributed immutabilization of secure logs (Jordi Cucurull and Jordi Puiggal))
- The secure logs generated will have 2 types of entries: Regular entries and Checkpoint entries.
- This approach allows us to detect where the alteration of content has occured (see equations).

# Logs Immutabilization Mechanisms (2)

- Regular entries are generated as the following:

$$L_i = (LogInfo_i, h_i) \quad \text{where} \quad h_i = HMAC(K_j, (h_{i-1}|LogInfo_i)) \quad (1)$$

- Meanwhile, a Checkpoint entry is generated after an arbitrary number of lines (50 lines in our implementation), in accordance with the equation:

$$Chk_j = L_i = (LogInfo_i, K_{j-1}, E(P_{enc}, K_j), Sig_j, h_{i-1}, h_i) \quad \text{where} \qquad (2)$$

$$h_i = HMAC(K_b, (h_{i-1}|K_{j-1}|LogInfo_i))$$

$$Sig_j = S(S_{sig}, (h_{i-1}|K_{j-1}|E(P_{enc}, K_j)|h_i|LogInfo_i)$$

# Benchmarking of Blockchain-based Solutions (1)

- **Blockstack**
  - Naming system built on Bitcoin's Blockchain.
  - Allows tracking key-value pairs in form of (unique) names and their associated data.
  - Relies on Blockchain "as a communication channel for announcing state changes".
  - A dedicated data plane for storing the name-data pairs records.
- **Proof of Existence**
  - An online service for storing and timestamping the existence of a document inside the Blockchain.
  - Keeps a hash of the file in the chain, with a referrence also to the time in which the document was stored.
  - Offers the advantage of validating the documents' existence even if the online service is down (built on Bitcoin's Blockchain).
  - Uses the OP_RETURN opcode to pass information in Bitcoin's transactions.

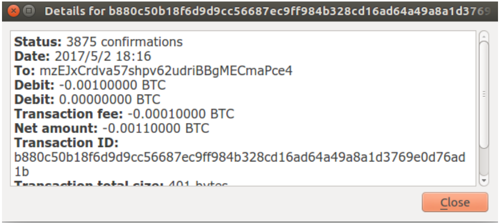
## Benchmarking of Blockchain-based Solutions (2)

- **Hyperledger**
  - Smart contract blockchain platform.
  - A set of distributed ledger solutions based on different platforms and programming languages.
  - Based on the expectation that there will be many blockchain networks, with each network ledger serving a different goal.
  - Offers various services ( Blockchain Explorer, Fabric,...etc).
- **Namecoin**
  - Another Blockchain-based solution for namespaces storage.
  - Hard fork of Bitcoin's Blockchain.
  - Allows secure storage of key value pairs (same value can exist within different namespaces though).

# Bitcoin's OP_RETURN

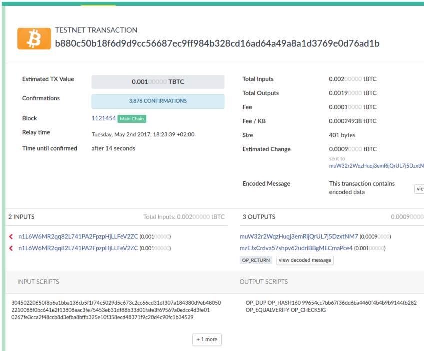
- Bitcoin's increasing popularity ==> Use cases outside finance ==> Need to send data within transactions.
- Developers tried different ways to work around this issue (adding values in outputs, manipulating Bitcoin's scripts,...etc).
- Large members of the Bitcoin's community did not like the bits of data added in transactions, especially as it affected (negatively) the mining process.
- An opcode (operation code that pushes data) called OP_RETURN was consequently standarized by Bitcoin and made available to developers in order to include up to 80 bytes of data in transactions.

# OP_RETURN Transaction (Example) (1)



Details for b880c50b18f6d9d9cc56687ec9ff984b328cd16ad64a49a8a1d376
Close

**Status:** 3875 confirmations
**Date:** 2017/5/2 18:16
**To:** mzEJxCrdva57shpv62udriBBgMECmaPce4
**Debit:** -0.00100000 BTC
**Debit:** 0.00000000 BTC
**Transaction fee:** -0.00010000 BTC
**Net amount:** -0.00110000 BTC
**Transaction ID:** b880c50b18f6d9d9cc56687ec9ff984b328cd16ad64a49a8a1d3769e0d76ad1b
**Transaction total size:** 401 bytes



TESTNET TRANSACTION
b880c50b18f6d9d9cc56687ec9ff984b328cd16ad64a49a8a1d3769e0d76ad1b

| | | | |
|---|---|---|---|
| Estimated TX Value | 0.00100000 TBTC | Total Inputs | 0.00200000 tBTC |
| Confirmations | 3,876 CONFIRMATIONS | Total Outputs | 0.00190000 tBTC |
| Block | 1121454 Main Chain | Fee | 0.00010000 tBTC |
| Relay time | Tuesday, May 2nd 2017, 18:23:39 +02:00 | Fee / KB | 0.00024938 tBTC |
| Time until confirmed | after 14 seconds | Size | 401 bytes |
| | | Estimated Change | 0.00090000 tBTC<br>sent to<br>muW32r2WqzHuqj3emRijQrUL7j5DzxtN |
| | | Encoded Message | This transaction contains encoded data |

2 INPUTS Total Inputs: 0.00200000 tBTC

n1L6W6MR2qq82L741PA2FpzpHjLLFeV2ZC (0.00100000)
n1L6W6MR2qq82L741PA2FpzpHjLLFeV2ZC (0.00100000)

3 OUTPUTS 0.00090000

muW32r2WqzHuqj3emRijQrUL7j5DzxtNM7 (0.00090000)
mzEJxCrdva57shpv62udriBBgMECmaPce4 (0.00100000)
OP_RETURN

INPUT SCRIPTS

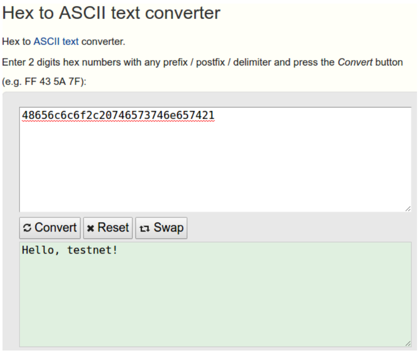
```
30450220650f8b6e1bba136cb5f1f74c5029d5c673c2cc66cd31df307a184380d9eb48050
2210088f0bc641e2f13808eac3fe75453eb31df88b33d01fafe3f69569a0edcc4d3fe01
0267fe3cca2f48ccb8d3efba8bffb325e10f358ecd48371f9c20d4c90fc1b34529
```

OUTPUT SCRIPTS

```
OP_DUP OP_HASH160 99654cc7bb67f36dd6ba4460f4b4b9b9144fb282
OP_EQUALVERIFY OP_CHECKSIG
```

# OP_RETURN Transaction (Example) (2)

6a0f48656c6c6f2c20746573746e657421
Hello, testnet!



Hex to ASCII text converter

Hex to ASCII text converter.

Enter 2 digits hex numbers with any prefix / postfix / delimiter and press the *Convert* button (e.g. FF 43 5A 7F):

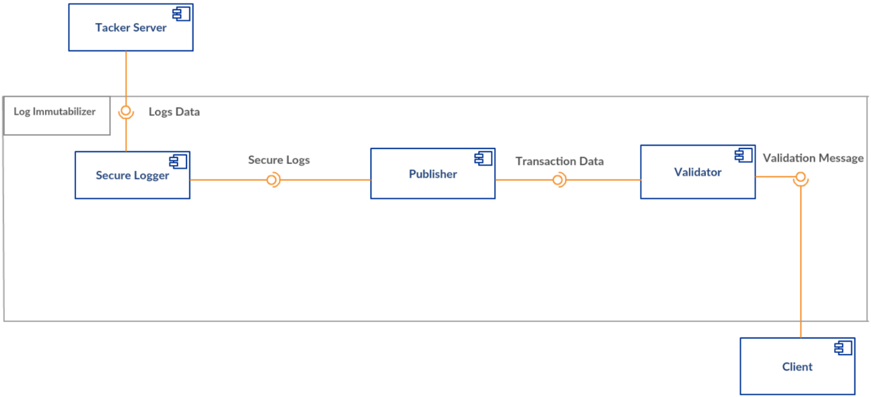
```
48656c6c6f2c20746573746e657421
```

```
Hello, testnet!
```

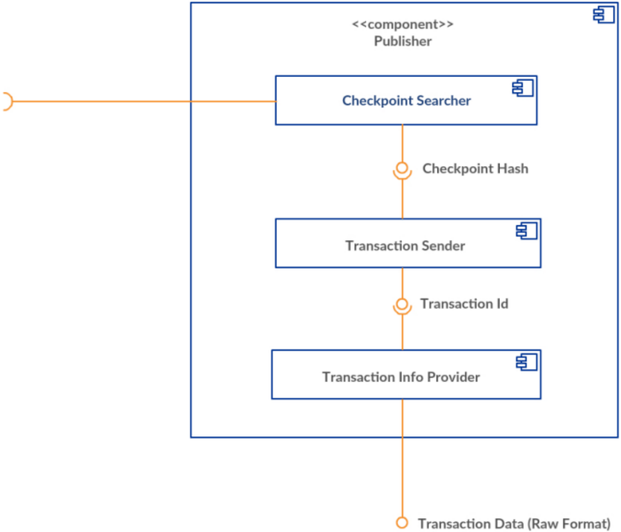
# Analysis

- In order to implement the logs immutabilizer, we rely on a component-oriented system.
- Beside the components that will secure Tacker's log files (see previous equations), additionnal components are needed to store the authenticity proofs in the Blockchain.
- A final process of Checkpoints validation needs also to be taken into account.

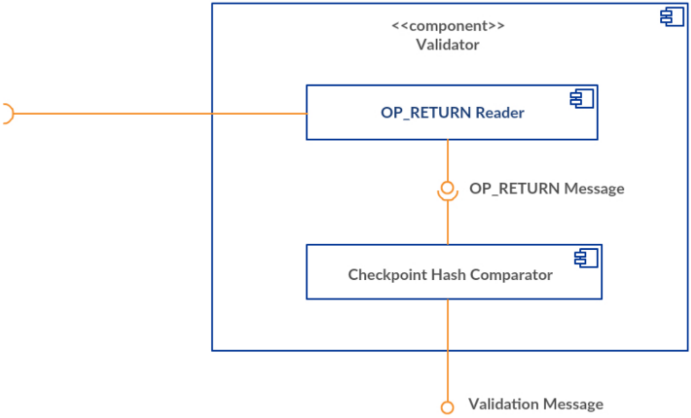
# Conception: Components Diagram



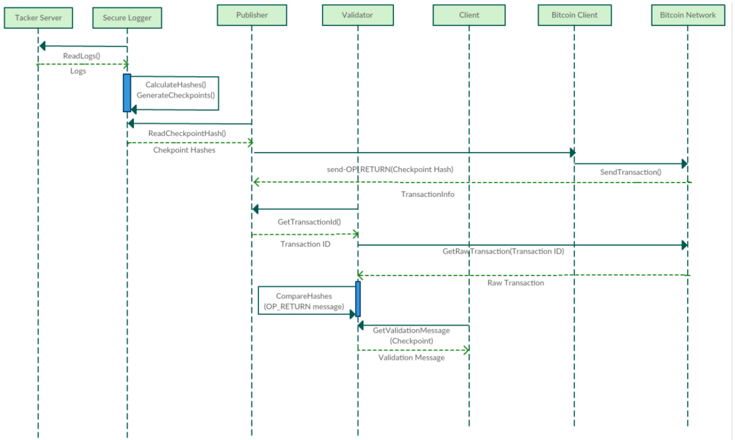
# Conception: The Publisher Component



# Conception: The Validator Component



# Conception: The Sequence Diagram



# Technologies (1)

Your Connection to ICT Research

- **OP_RETURN python library**
  - Interacts directly with Bitcoin's Blockchain.
  - Include the message taken as arguemnt through CLI command in the OP_RETURN opcode.
  - Allows 3 operations: send, store and retrieve.

- **Bitcoin Core**
  - Open source project that maintains and publishes the Bitcoin client software.
  - Besides validating transactions "locally" (160 Go need to be downloaded though !!), Bitcoin Core acts also as a Bitcoin Wallet.
  - Unlike Bitcoin's online clients, Bitcoin Core offers improved security.

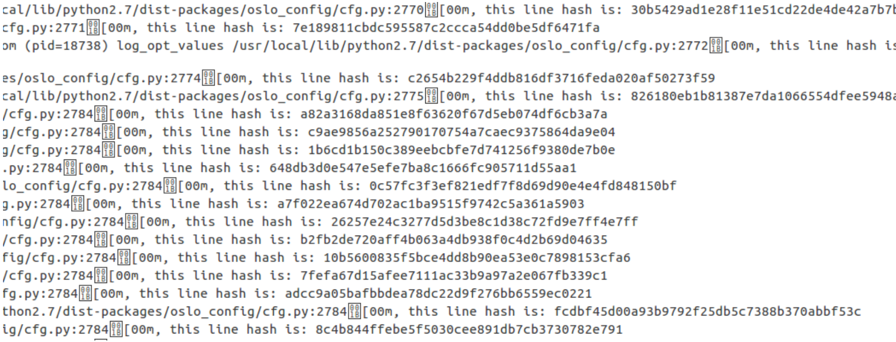
# Technologies (2)

- **RabbitMQ Messaging**
  - A message broker typically used for building integration between applications (or components) using messages.
  - Relies on the producer/consumer communication via messaging queues.
  - Supports local communication as well as remote one.

# Components' Implementation: The Secure Logger

- The component acts on text log files as the source of the incoming data.
- A sample log text file generated by Tacker server was stored, then secured as per the previous mechanisms.
- A Checkpoint entry is generated after every 50 lines.
- We used the HMAC python module to calculate the hashes.

# The Secure Log File



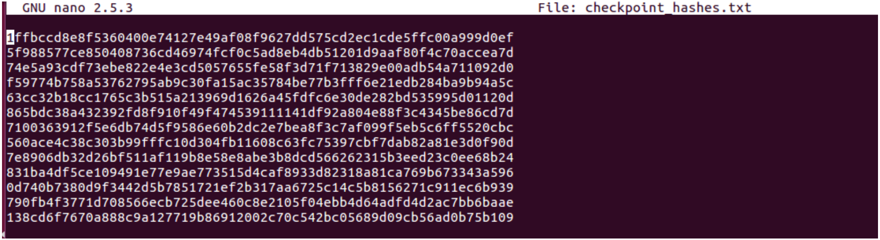
```
cal/lib/python2.7/dist-packages/oslo_config/cfg.py:2770␛[00m, this line hash is: 30b5429ad1e28f11e51cd22de4de42a7b7b
cfg.py:2771␛[00m, this line hash is: 7e189811cbdc595587c2ccca54dd0be5df6471fa
om (pid=18738) log_opt_values /usr/local/lib/python2.7/dist-packages/oslo_config/cfg.py:2772␛[00m, this line hash is

es/oslo_config/cfg.py:2774␛[00m, this line hash is: c2654b229f4ddb816df3716feda020af50273f59
cal/lib/python2.7/dist-packages/oslo_config/cfg.py:2775␛[00m, this line hash is: 826180eb1b81387e7da1066554dfee5948a
/cfg.py:2784␛[00m, this line hash is: a82a3168da851e8f63620f67d5eb074df6cb3a7a
g/cfg.py:2784␛[00m, this line hash is: c9ae9856a252790170754a7caec9375864da9e04
g/cfg.py:2784␛[00m, this line hash is: 1b6cd1b150c389eebcbfe7d741256f9380de7b0e
.py:2784␛[00m, this line hash is: 648db3d0e547e5efe7ba8c1666fc905711d55aa1
lo_config/cfg.py:2784␛[00m, this line hash is: 0c57fc3f3ef821edf7f8d69d90e4e4fd848150bf
g.py:2784␛[00m, this line hash is: a7f022ea674d702ac1ba9515f9742c5a361a5903
nfig/cfg.py:2784␛[00m, this line hash is: 26257e24c3277d5d3be8c1d38c72fd9e7ff4e7ff
/cfg.py:2784␛[00m, this line hash is: b2fb2de720aff4b063a4db938f0c4d2b69d04635
fig/cfg.py:2784␛[00m, this line hash is: 10b5600835f5bce4dd8b90ea53e0c7898153cfa6
/cfg.py:2784␛[00m, this line hash is: 7fefa67d15afee7111ac33b9a97a2e067fb339c1
fg.py:2784␛[00m, this line hash is: adcc9a05bafbbdea78dc22d9f276bb6559ec0221
thon2.7/dist-packages/oslo_config/cfg.py:2784␛[00m, this line hash is: fcdbf45d00a93b9792f25db5c7388b370abbf53c
ig/cfg.py:2784␛[00m, this line hash is: 8c4b844ffebe5f5030cee891db7cb3730782e791
```

## Components' Implementation: The Hash Reader

- Reads the Checkpoint entries from the secure log file.
- Extract the checkpoints' hashes and store them in a separate file.
- The Checkpoint hashes file is provided for the Publisher component.

# The Checkpoints' Hashes

Your Connection to ICT Research

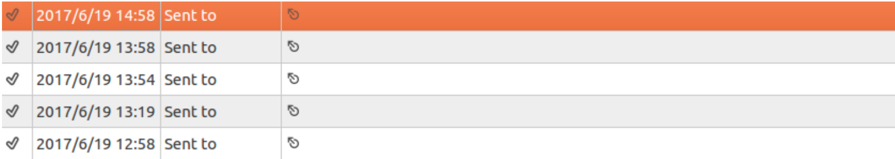


```
  GNU nano 2.5.3                                                  File: checkpoint_hashes.txt

1ffbccd8e8f5360400e74127e49af08f9627dd575cd2ec1cde5ffc00a999d0ef
5f988577ce850408736cd46974fcf0c5ad8eb4db51201d9aaf80f4c70accea7d
74e5a93cdf73ebe822e4e3cd5057655fe58f3d71f713829e00adb54a711092d0
f59774b758a53762795ab9c30fa15ac35784be77b3fff6e21edb284ba9b94a5c
63cc32b18cc1765c3b515a213969d1626a45fdfc6e30de282bd535995d01120d
865bdc38a432392fd8f910f49f474539111141df92a804e88f3c4345be86cd7d
7100363912f5e6db74d5f9586e60b2dc2e7bea8f3c7af099f5eb5c6ff5520cbc
560ace4c38c303b99fffc10d304fb11608c63fc75397cbf7dab82a81e3d0f90d
7e8906db32d26bf511af119b8e58e8abe3b8dcd566262315b3eed23c0ee68b24
831ba4df5ce109491e77e9ae773515d4caf8933d82318a81ca769b673343a596
0d740b7380d9f3442d5b7851721ef2b317aa6725c14c5b8156271c911ec6b939
790fb4f3771d708566ecb725dee460c8e2105f04ebb4d64adfd4d2ac7bb6baae
138cd6f7670a888c9a127719b86912002c70c542bc05689d09cb56ad0b75b109
```

# Components' Implementation: The Publisher

- Issues an OP_RETURN Bitcoin transaction containing the Checkpoint hash.
- Due to Bitcoin Core's scalability issues (see report), an OP_RETURN transaction is sent once per hour.
- Component implemented in Bash (rather than python) due to the OP_RETURN library's inconsistency.
- After each transaction, the transaction id is stored and sent to the next component.

# Bitcoin's Transactions



| | | |
|---|---|---|
| ✓ | 2017/6/19 14:58 | Sent to |
| ✓ | 2017/6/19 13:58 | Sent to |
| ✓ | 2017/6/19 13:54 | Sent to |
| ✓ | 2017/6/19 13:19 | Sent to |
| ✓ | 2017/6/19 12:58 | Sent to |

## Components' Implementation: The Validator

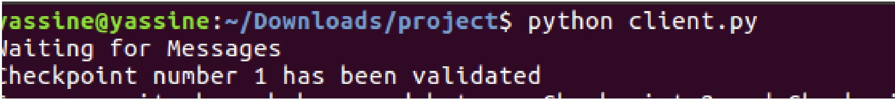
- Contains 4 subcomponents that communicate using RabbitMQ.

- JSON objects are passed through to send the recent transaction id as well as the transaction's hexadecimal format.

- The OP_RETURN message is identified using the prefix '6a20'.

- After reading the hash from the transaction, a comparison is made with the stored Checkpoint hashes, and a validation message is sent eventually to the client component.

# Checkpoint Validation Message



```
yassine@yassine:~/Downloads/project$ python client.py
Waiting for Messages
Checkpoint number 1 has been validated
```

## Design Decisions

- A Checkpoint was issued after every 50 lines (arbitrary choice), given the log file contained circa 500 lines.
- Bitcoin Core was used in the testnet mode, so that tBTCs could be obtained for free.
- Each transaction was sent with a 0.1 tBTC fee, which limited the transaction validation time in almost 50 minutes.

# Conclusion

Your Connection to ICT Research

- The Blockchain is proving to a suitable technology for protecting records from alteration.
- The network's programmability offered by SDN makes the network more flexible and adaptable to the user's requirements.
- Service function Chaining is still a long way off from being provided as a consistent use case in networking systems.

## Perspective & Future Work

Your Connection to ICT Research

- Packaging of the log immutabilzation application (Docker, Kubernetes,...etc).
- Working on the feasibility of the remaining A's (Keystone for Authentication? , Permissioned Blockchains for Authorization?).
- Making log immutabilization a real time process.
- More tests on SFC platforms (Port Chaining in Neutron?)

Thank You For Your Attention