International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-11 Issue-2, December 2021

Proposed Methods to Rollback A Failed Update of IoT Devices

Hoai-Nam Nguyen, Truong-Thang Nguyen, Thu-Nga Nguyen Thi, Manh-Dong Tran, Ba-Hung Tran

Abstract*: The increasing number of IoT devices leads to the demand for updating them securely, and stably. Without updating there is a high risk of being attack by hackers or malfunction due to outdated packages. To the date of this article, a lot of IoT updating methods were introduced. Among them are updating using VM containers, using packages, or using blockchain. However, some methods take time to recover a failed update while some cases require the minimum downtime of the IoT devices, or even some devices cannot communicate to the control server or cannot perform the rollback themselves after the update failure. In this article, our contribution is implementing different methods to reduce the rollback down time issue including using primary - secondary selection, using git revert, and using blockchain-based revert.

***Keywords*: Blockchain, Internet of Things, Networks.**

I. INTRODUCTION

Until 2020, around 200 billion IoT devices connect to the Internet. The rise of covid-19 leads to a higher demand for using IoT for working and living [1]. The main reason is remote working requires more devices to keep the daily tasks easier such as a smart camera for video call, smartwatch for health monitor, smart speakers for home convenience. With the average estimation of 26 smart devices for every person, we also see a high risk of being attacked via IoT devices. The more reliant on devices, the more vulnerable people are with the diversity of cyberattacks. Take an example, the house is not safe anymore because the hacker can disable your alarm by attacking the IoT system before a physical attack happens.

That kind of danger leads to a demand for updating the IoT device frequently to avoid being attacked. However, most of the updates are over-the-air, which means, on one hand, the owner or the organization doesn’t have to bring the devices to a central location to do the update, but on the other hand, the engineer cannot validate the update after the device updated. That means if the update fails, it takes time to recover the device to the server the operation, which might bring a lot of damages to the owner or the organization.

For those reasons, updating IoT devices is a -must- however it must satisfy the following factors:

- Integrity: Ensure the update files are transferred without losing or being damaged.
- Stability: Ensure the IoT devices work as before they were updated.
- Roll-back: Ensure the update can be rolled back when failure happens.

Concentrating on how the system can keep continuing to operate after the update and how the device can be recovered/rolled back, in this paper, we focus on the following issues:

- Researching different approaches to handle the failed update cases, including (i) Using Primary - Secondary device selection; (ii) Using git to roll back the update; (iii) Using blockchain to roll back the update.
- Comparing the above approaches in terms of (i) the Possibility to handle failures to keep the system running; (ii) the Time to roll back to the previous version.

By defining the general scope of updating IoT devices as updating the OS or packages or configuration, we only filter the existed solutions of the same scope. With that scope, the update falls into application update, which has several solutions including virtual ma-chine containers, using packages, or using blockchain to store the update information. The solution using virtual machines is not dug deeper in this article, because it only serves powerful IoT devices that cannot be a wide applicable solution. Moreover, one of the below solutions - using the snap package - already contains the idea of using a virtual machine.

A. Using package to handle update and rollback [2]

One solution is using a snap package by Ubuntu Core. Ubuntu Core is a secure, lightweight, robust, application-centric operating system created for the Internet of Things. Its read-only root filesystem is built from the same packages used to build the wider set of Ubuntu distributions, but differs in the way packages are delivered, and crucially, updated. This is all handled by snap, a secure, confined, dependency-free, cross-platform Linux packaging system.

The main approach of Ubuntu Core is updating via snap packages. Those packages have a clean separation between the system and the application. For that reason, unlike traditional updates, a failed update does not affect the whole system.

Manuscript received on December 05, 2021.
Revised Manuscript received on December 10, 2021.
Manuscript published on December 30, 2021.

* Correspondence Author
Hoai-Nam Nguyen*, Institute of Information Technology, Vietnam Academy of Science and Technology, Hanoi, Vietnam. Email: nguyenhoainam@ioit.ac.vn
Truong-Thang Nguyen, Institute of Information Technology, Vietnam Academy of Science and Technology, Hanoi, Vietnam. Email: ntthang@ioit.ac.vn
Thu-Nga Nguyen Thi, Institute of Information Technology, Vietnam Academy of Science and Technology, Hanoi, Vietnam. Email: nttnga@ioit.ac.vn
Manh-Dong Tran, Institute of Information Technology, Vietnam Academy of Science and Technology, Hanoi, Vietnam. Email: dongtm@ioit.ac.vn
Ba-Hung Tran, Institute of Information Technology, Vietnam Academy of Science and Technology, Hanoi, Vietnam. Email: tbhung@ioit.ac.vn

The system can run in any state of update: either success or failure. In case of failure, the system can revert to a previous version because the system can store multiple versions of the same snap. This mechanism, according to the Ubuntu Core document, is known as parallel installation, which means independent instances of the same snap can be installed and run within their own namespace. As we experienced the snap revert, the time for rolling back is 0.2 second to 0.3 second.

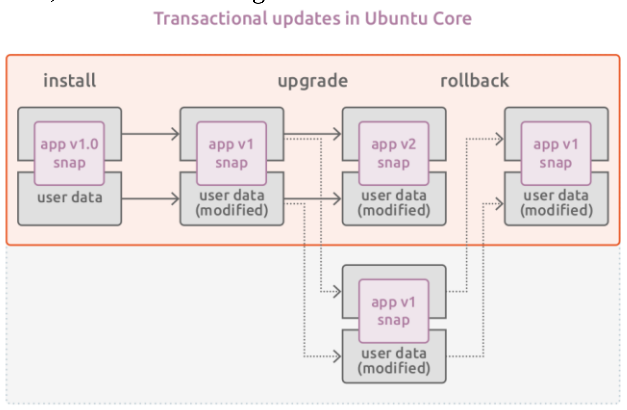


Fig. 1.Transactional updates using snap by Ubuntu Core (Source: https://pages.ubuntu.com/rs/066-EOV-335/images/Over-the-air%20software_12.05.20.pdf)

B. Using docker to perform the rollback

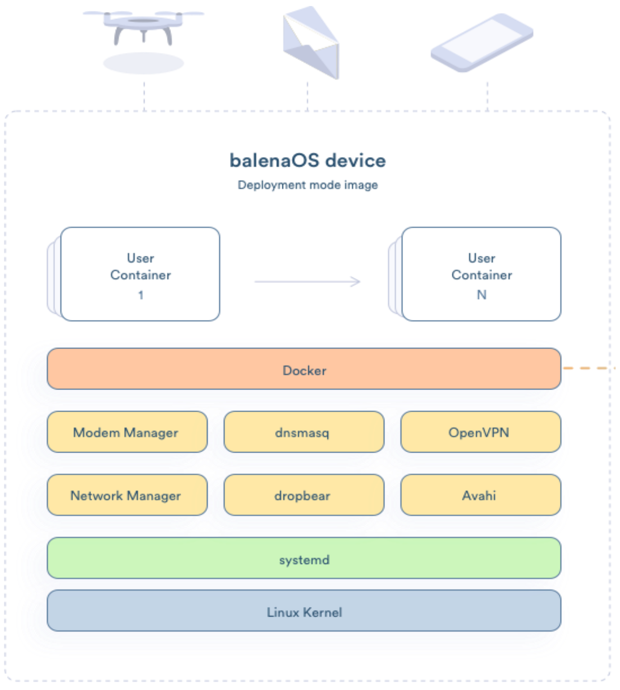


Fig. 2.Docker roll back mechanism (Source: https://www.balena.io)

One of the solutions that use docker is Balena. It is a platform that allows docker containers to run with reliability.

The container update uses 4 strategies [3-4]:

- Download then kill: Download the new container first then kill the old one
- Kill then download: Kill the current container then download the new one.
- Delete then download: Kill then download the current container, then download the new one.
- Hand-over: In case the required downtime is zero, the old container and new container run together, then the Supervisor of Balena kill the old container and delete it.

Among those strategies, the Hand-over strategy is the most suitable one to roll back a failed update. The Supervisor of Balena simply switch back to the old container. However, there will be a step of taking back all the resources grant to the new container before switch back to the old one. The default time-out for this approach is 1 minute.

C. Using partitions to handle update [5]

One of the commercial solutions use this method is Mender [6]. The requirement of this method is the IoT device must have at least 2 partitions so that the update can be installed to either of those partitions.

The mechanism of this update method is the passive partition will receive the update. After that, the device will reboot to the passive partition, then perform the update progress. In this case, the passive partition will become the active partition. If the update fails, the device can roll back to the original active partition.

This method can assure that the device will not become a brick after being updated. However, it contains many drawbacks including:

- The hardware cost: because the device needs larger storage for multiple partitions.
- Higher downtime: the device must be rebooted to perform the update, even the smallest update. In this case the downtime cannot be counted as seconds.
- Larger data transferred and longer update time: because the whole partition is updated.

For the above reasons, this method is not considered as a method to investigate in this paper, because the main target of this article is to minimize the downtime for rollback process.

II. PROPOSED SOLUTIONS

The three above solutions can handle the updating IoT device process, which ensures the integrity and availability of the update, however focusing on the rollback part of updating the IoT device process, those solutions can mostly handle the following cases:

- The devices may malfunction however they can still connect to the gateway and can be controlled by the gateway.
- The devices can handle the rollback themselves after a failed update.
- The update part is the same with every updated device, so that the roll-back process requires only one update binary file, however, in the case of updating the config-uration, which is different for each device, we need another mechanism.
- Downtime for updating is not a factor to be considered.

To expand the capability of roll back a failed update, we would like to propose different methods as the following.

A. Perform the rollback using Git system

To rollback a device application to the previous version, using Git will bring the following advantages:

- The version can be customized for each device. Using a package and do the rollback is possible, however, if changing or preserving the device parameters is required, that solution is not suitable. Moreover, sometimes the configuration is changed after an update that rolling back cannot help restoring the previous configuration. Using Git can help saving parameters, configuration, and the application version.
- Using Git system allows the replacement of a "brick" device by a new device, with the ability to restore the full state of the previous device.

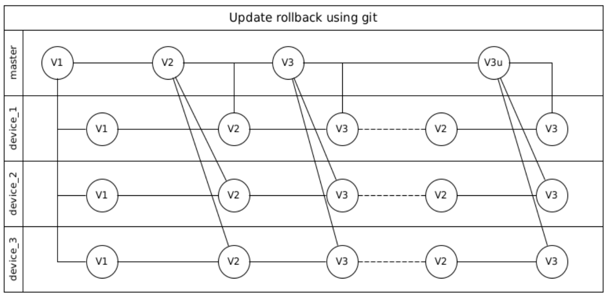


Fig. 3.Device update information as a snapshot for each git version

In the figure, if the updated version #3 fails, all devices can be restored to the pre-vious version #2, while waiting for the update of version #3 (named #3u). The mechanism introduced here is instead of pulling the update from the master branch, each device has its own branch, and each branch will merge the update from the master branch first, then the update progress is made from each device's branch. This helps preserve the configuration of each device and save the storage size of the git update folder that each device only stores its own git branch and the master branch in the local storage. The Figure below describes the progress of updating a device using Git system. The Control Server pushes the update to the Git Server, then requests the device to perform the update. After that, the device will pull the update from the Git Server. The Control Server will test the device after the update. If the test is not passed, the Control Server will request the device to perform the rollback progress.

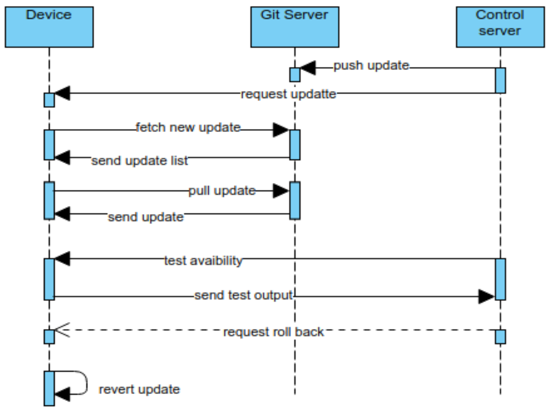


Fig. 4.Update process using git mechanism

B. Perform the rollback using a blockchain-based system

There are already plenty of solutions that serve the update process of IoT devices using a blockchain-based system. The blockchain-based system helps the devices pull the latest update without the dependency on a single server. Based on those, the main idea of using a blockchain-based system is in case of cannot connect to the central server, there will be many alternative version control sources that help the device revert to the previous version or download a new fix. The alternative version control sources can be another control server or the gateway servers.

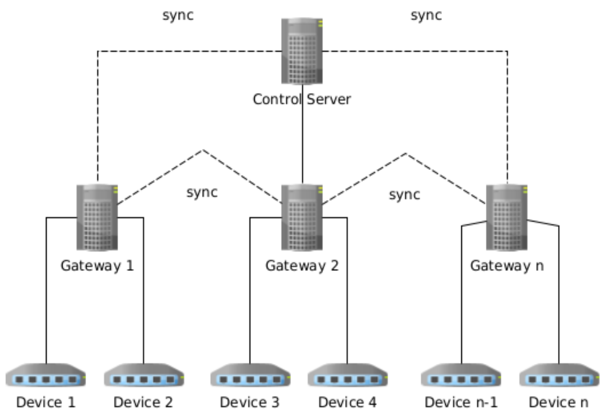
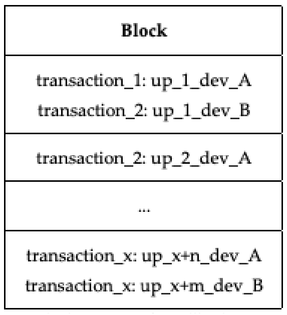


Fig. 5.Rollback using blockchain-based system model

In the above Figure, not only the control server, but each gateway is also a node that stores the version of updates, including the config file. With this model, in case of update failure, a device can connect to the nearest gateway to perform the rollback. However, if the closet gateway does not work, a device can connect to the second nearest gateway to do that.



Block
transaction_1: up_1_dev_A transaction_2: up_1_dev_B
transaction_2: up_2_dev_A
...
transaction_x: up_x+n_dev_A transaction_x: up_x+m_dev_B

Fig. 6.Block structure for rollback process

The above Figure describes how the updated versions of each device are store. Each version (including the configuration) information is represented as a transaction in a block. To roll back, the device will ask for the previous transaction of itself, then download the specific version to install. To avoid using the predefined transaction fields, the blockchain used for this solution should be Private or Consortium Blockchain. It not only helps to store the necessary information but also helps reduce the time for inserting a block.

The information of a transaction can be (1) Transaction hash; (2) Device ID; (3) Version; (4) file hash/signature. Once getting the information for the transaction, the device can download the specific version to roll back. The same mechanism is made for the case of a failed update; however, the devices need to download a new version to fix the update.

To reduce downtime, the proof method will not be the proof of work or the proof of stake, but rather the proof of authority, which means the proof method will check whether the node is in the trusted list to have the permission to insert a block. The roll back flow is described as the following:

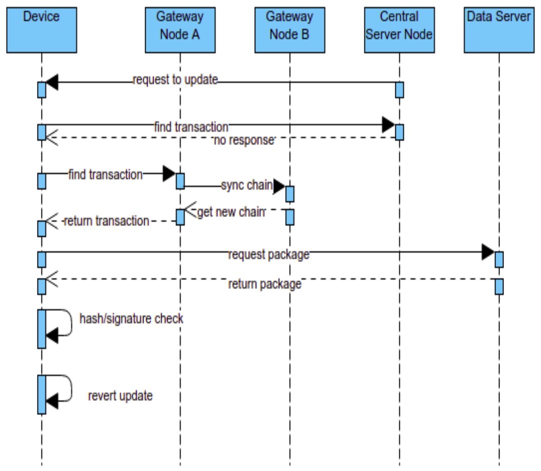


Fig. 7.Blockchain roll back flow

C. Perform the rollback using primary - secondary selection

The main problem of the first 2 solutions or some other existed solution is the downtime is significant. For critical system, if the devices need a lot of time to recover to the previous state, there might lead to damage or dangerous situation, for example, the camera system or the health care system.

Another problem of the other solutions is the rollback progress can only perform when the devices can still operate the function themselves. In the case of "brick" devices or the devices freeze or in a long process, it requires someone to come and handle the failure manually, which takes some time for traveling from one location to the next location.

For that reason, to accelerate the process to reduce the downtime and to keep the system running before the engineer approaches the device, another solution is to switch from the failed update device (the primary device) to the working device with the previous software version (the secondary device).

To apply this solution, for each device we must prepare another secondary device, with the same update version as the primary one. The control server or the IoT gateway will have the responsibility of deciding which device will become the primary device and which device will become the secondary device.

The general architecture of the update system will be:

1) For the case of using one central control server as the selection server:

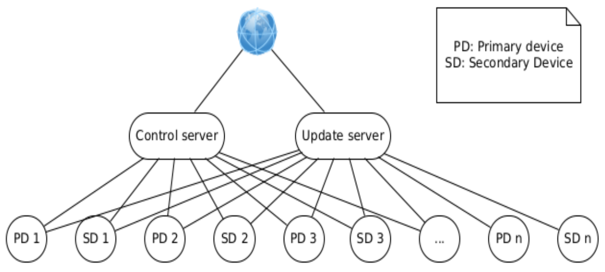


Fig. 8.Primary Secondary Selection solution with one central control server

2) For the case of using the IoT Gateway as the selection server:

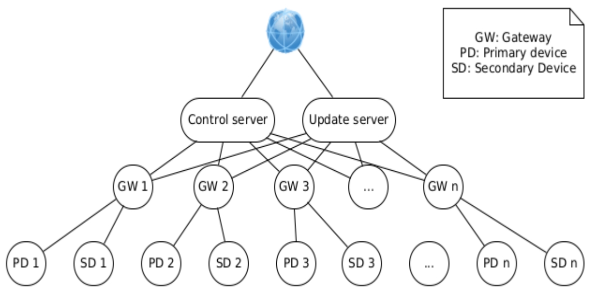


Fig. 9.Primary Secondary Selection solution using a gateway to select the device

The update process is described as below:

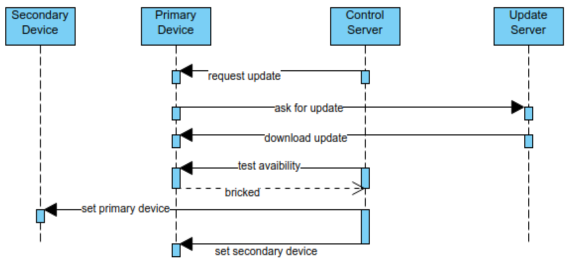


Fig. 10. Primary - Secondary selection rollback process

This solution solves the problem of downtime; however, it increases the cost due to the implementation of an extra device as the secondary device.

III. IMPLEMENTATION

A. Environment

To demonstrate the solutions, we proposed the following environment:

Table- I: Central server

Hardware	Specification
CPU	2GHz single core
Memory	2 GB
Location	Singapore
OS	Ubuntu 20.04 LTS

Table- II: Client device

Hardware	Specification
CPU	2GHz 1 core
Memory	2 GB
Location	Hanoi, Vietnam
OS	Raspberry Pi OS

Table- III: Technology Stack

Purpose	Technology
Update package	PIP package
API	Django Rest Framework
Private Blockchain	Python
Git version control	Private Git
Emulator	Sense HAT Emulator

The package: In this case, we created a package that obtains the temperature from the sensor and sends it to the central server for storing via /api/device-temp API. The information sent including {‘device_ip’: ‘The IP of the device; ‘temp’: ‘The device tem-perature’, ‘timestamp’: ‘Timestamp taken’}. The size of the old package is 1.725 bytes, and the new package is 1.953 bytes.

The command: The central server will take care of deciding when the rollback happens. It can be the loss of the request from the devices, or the update found wrong. The device will request to an API /api/central-server-command to perform the suitable action.

B. Git rollback system

To demonstrate how a failed update can be recovered, we introduce 2 scenarios:

- The device will roll back to the previous version by using git.
- The device downloads a new fix from the central server and performs the update.

The central Git server will store each device’s package versions in a branch. This helps the device store include its own configuration without being overwritten by others.

The device will either pull the update from the Git server or push the current version to the central Git server to backup.

To perform the private Git server, each device must add the origin repo url. In our demonstration the URL is:

git remote set-url origin <usernam>@<git-url>:<path_to_git_folder>

In the first case, the device will roll back using reset to switch back to the previous version stored in its storage. The Python subprocess will perform it as:

```
subprocess.run(
    ['git', 'reset', '--hard', 'HEAD^1'],
stdout=subprocess.PIPE,\
    cwd=pi_git_path)
```

The second case, the device will pull the update fix from the central server:

```
    subprocess.run(['git', 'pull', 'origin', 'device1'],
                              stdout=subprocess.PIPE,
cwd=pi_git_path)
```

After pulling, the device will install the package using Python pip package install.

C. Blockchain-based rollback system

We have built a private blockchain [7] using Python with the following attributes:

- current_transactions: A list of pending transactions that are not stored in a block.
- chain: A chain of blocks. Each block has the following attributes: index, timestamp, transactions, proof, previous_hash.
- nodes: A list of neighbor nodes in the network.
- last_block: The last stored block in the chain.

Below is the blockchain’s methods:

- __init__(): The blockchain constructor, that sets the default value of current_transactions, chain, nodes.
- new_block(self, proof, previous_hash): Create a new block that stores all current transactions, proof, and previous hash.
- proof_of_authority(self, node): The proof value of each block is calculated using the proof_of_authority, which takes a node’s information as the input and validates if it is a trusted node or not.
- hash(block): To generate the hash of the block.
- new_transaction(self, tx_hash, device, version, file_hash): To append a transaction to the current transaction list.
- register_node(self, address): To register a new node into the network.
- valid_chain(self, chain): To validate if a chain is correct based on the hash comparison and the proof of authority.
- resolve_conflict(self): To update the chain if there is any node that contains a newer chain.

To demonstrate the rollback using the blockchain-based system, we also have 2 scenarios:

- The device will get the previous package by searching its previous transaction via the central node.
- The device will get the previous package by searching its previous transaction via a gateway node; however, the gateway node must sync the chain from a nearby gateway node that has the full chain with all transactions.

The first scenario, we performed the steps to obtains the previous update versions:

- Find all transactions that belong to the current device via the central node.
- Get the transaction details of the N-1 transaction.
- Download the specific package version from the server.
- Compare the hash (or check the signature if the package has the signature). If the package is validated, the device will install the package including the configuration.

The second scenario, besides the steps of the first scenario, there will an extra step:

- The requested node will sync the chain with other nodes.

The above step we have built with the case of 100 blocks difference between the target node and the central node.

D. Primary - Secondary section system

The primary - Secondary selection system is the simpliest implementation. The central server or the gateway waits for the requests from the device periodically. After a predefined timeout, the central server or the gateway will update in the database that: (1) the primary device will be in "secondary" mode; (2) the sencondary device will be in "primary" mode.

IV. RESULTS AND DISCUSSION

Having the implementation as above with pre-defined scenarios, we have the fol-lowing comparison results among 3 methods:

A. Comparison of update failure cases

Table- IV: Update failure cases

Methods	Cases			
	Update causes "brick" device	**Some devices lost network connection**	**Devices roll back using local package version**	**Cannot connect to central server**
Rollback using Git system	Not solved	Solved	Solved	Partly solved
Rollback using Blockchain-based system	Not solved	Not solved	Not solved	Solved
Rollback using primary – secondary selection	Solved	Solved	Not solved	Not solved
Rollback using snap	Not solved	Solved	Solved	Solved
Rollback using partitions	Solved	Not Solved	Not Solved	Not Solved

B. Comparison of down time

To evaluate the time to recover a stable state, we have made 10,000 requests for each of the following cases.

1) Case 1: Using Git system

The chart below describes the response time using git revert when a failure happens.

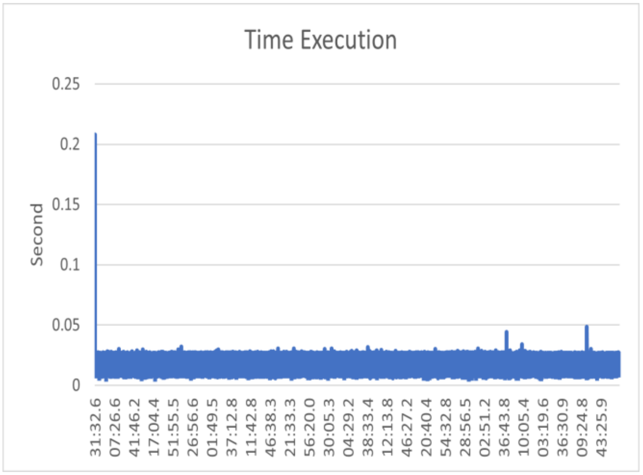


Fig. 11. Execution time using git revert to previous version.

The chart below describes the response time using git pull when a failure happens.

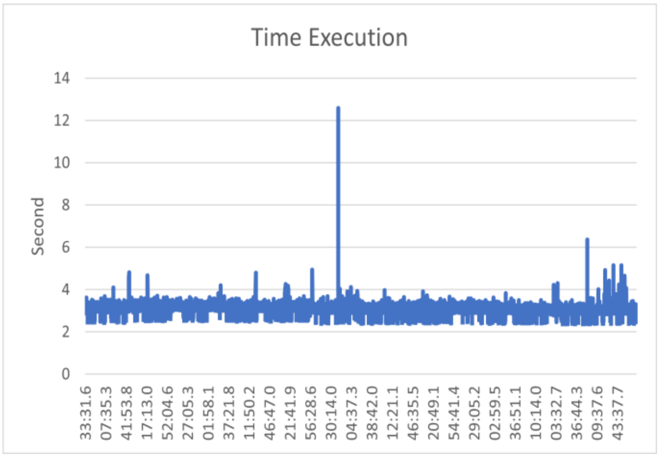


Fig. 12. Exection time using git pull to install the previous version.

In comparison, the execution time of 'git revert' solution is significantly less than the one of 'git pull'.

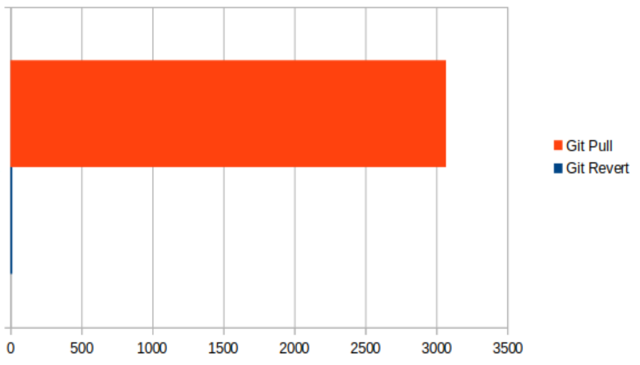


Fig. 13. Comparison of exection time between git revert method and git pull method

2) Case 2: Using a blockchain-based system

The chart below describes the execution time if the devices connect directly to the central server as a node.

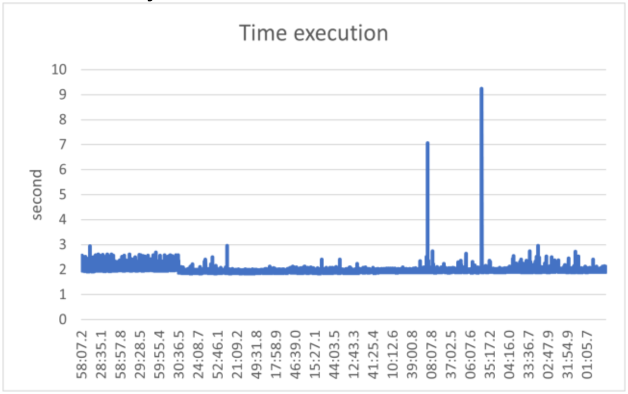


Fig. 14. Execution time by directly connecting to central node

The chart below describes the execution time if the devices connect to a nearby gateway as a blockchain node, while the gateway node will sync the chain with other nodes.

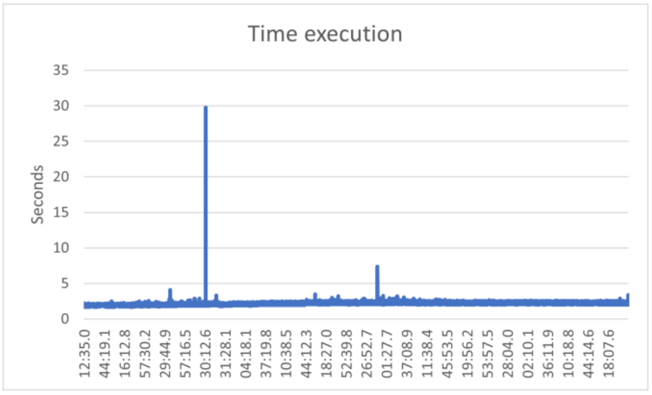


Fig. 15. Execution time by connecting to a nearby node and sync with other nodes

In comparison, the method of synchronizing among the nodes took more time for rolling back than the method of connecting to the central server as a node. The time dif-ference will be higher if the number of nodes increases. However, the execution time can be shorter if the nodes sync with each other before the rollback action is performed.

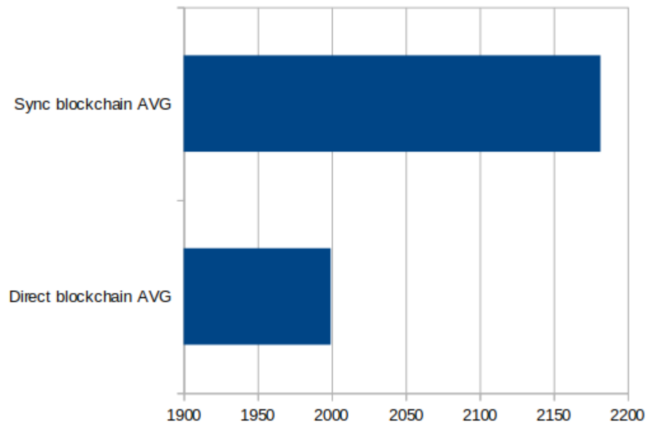


Fig. 16. Comparison of execution time between direct connecting to central node and connecting to nearby node

Without calculating the primary - secondary selection solution because the downtime depends on the predefined timeout, we can observe the execution time of the 2 other solu-tions:

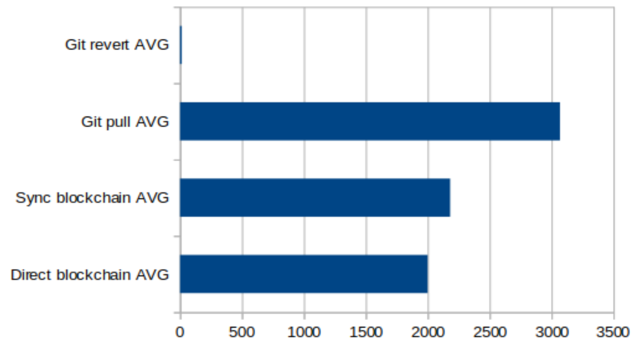


Fig. 17. Comparison of execution time among 4 cases: git revert, git pull, direct connecton, node sync connection

Therefore, the self revert to the previous update version is the fastest way for a device to roll back. While using the same method, the git pull method tends to be the slowest way to perform the rollback.

V. CONCLUSIONS

In this paper, we proposed 3 solutions to roll back a failed update of IoT devices with the target of reducing the downtime for the devices: (1) using Git system; (2) using the blockchain-based system; (3) using primary - secondary selection. Depends on the condi-tion of the devices, including the self-operated capability, network connections, the con-nection among the central server and gateways, each solution has its own advantages and limitation.

The implementation in this paper is only for the proof of concept which serves the 2 factors: availability and integrity of the rollback. Our future work will include the confidentiality factor that ensures the roll-back package will not be modified or misused by an unauthorized access.

ACKNOWLEDGMENT

We would like to thank CS'20.10 project of the Institute of Information Technology, Vietnam Academy of Science and Technology that has provided funding for the study.

REFERENCES

1. The IoT landscape in 2020 and the need to keep our connected devices secure. Available online: https://licelus.com/insights/the-iot-landscape-in-2020-and-the-need-to-keep-our-connected-devices-secure
2. How Ubuntu Core transforms over-the-air software updates. Available online: https://pages.ubuntu.com/rs/066-EOV-335/images/Over-the-air%20software_12.05.20.pdf
3. Balena Mechanism. Available online: https://www.balena.io/static/flow-desktop@1x-291bec507a7c6ed12a8d2d7e282c4a0b.png
4. Balena update strategies. Available online: https://www.balena.io/docs/learn/deploy/release-strategy/update-strategies/
5. Two Ways to Update Embedded Devices Over-The-Air. Available online: https://mender.io/blog/the-two-main-ways-to-update-embedded-devices-over-the-air
6. Konstantinos Arakadakis, Pavlos Charalampidis, Antonis Makrogiannakis, Alexandros Fragkiadakis, Firmware over-the-air programming techniques for IoT networks - A survey, arXiv:2009.02260.
7. Learn Blockchains by Building One. Available online: https://medium.com/@vanflymen/learn-blockchains-by-building-one-117428612f46
8. Elizabeth Nathania Witanto, Yustus Eko Oktian, Sang-Gon Lee and Jin-Heung Lee, A Blockchain-Based OCF Firmware Update for IoT Devices, Journal of Applied Science, 2020, 10, 6744; doi:10.3390/app10196744.
9. Saraju P. Mohanty, Proof of Authentication: IoT-Friendly Blockchains, Article in IEEE Potentials · January 2019, DOI: 10.1109/MPOT.2018.2850541.
10. Aymen Boudguiga, Nabil Bouzerna, Louis Granboulan, Alexis Olivereau, Flavien Quesnel, Anthony Roger, Renaud Sirdey, Towards Better Availability and Accountability for IoT Updates by means of a Blockchain, IEEE Security & Privacy on the Blockchain (IEEE S&B 2017) an IEEE EuroS&P 2017 and Eurocrypt 2017 affiliated workshop, IEEE, Apr 2017, Paris, France. hal-01516350
11. Jonathan Bell, Thomas D. LaToza, Foteini Baldmitsi and Angelos Stavrou, 2017 IEEE/ACM 12th International Workshop on Software Engineering for Science (SE4Science).
12. Tanweer Alam, Blockchain and its Role in the Internet of Things (IoT), International Journal of Scientific Research in Computer Science, Engineering and Information Technology, pp. 151-157, 2019.

13. Liangqin Gong, Daniyal M. Alghazzawi and Li Cheng, BCoT Sentry: A Blockchain-Based Identity Authentication Framework for IoT Devices, Information 2021, 12, 203.
14. Alfonso Panarello * ID, Nachiket Tapas ID, Giovanni Merlino ID, Francesco Longo ID and Antonio Puliafito, Blockchain and IoT Integration: A Systematic Survey, Sensors 2018, 18, 2575; doi:10.3390/s18082575.
15. Nidhi Agarwal, Framework for Integration of Blockchain with IoT Devices, Mphasis.
16. K. Christidis, M. Devetsiokiotis, Blockchains and Smart Contracts for the IoT, Digital Object Identifier 10.1109/ACCESS.2016.2566339.
17. Nafiz Al Asad; Md. Tausif Elahi; Abdullah Al Hasan; Mohammad Abu Yousuf, Permission-Based Blockchain with Proof of Authority for Secured Healthcare Data Sharing, 2020 2nd International Conference on Advanced Information and Communication Technology (ICAICT)
18. Wenbo Wang, Dinh Thai Hoang, Peizhao Hu, Zehui, Xiong, Student Dusit Niyato, Ping Wang, Yonggang, Wen, and Dong In Kim, A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks, IEEE Access (Volume: 7)
19. John D. Blischak, Emily R. Davenport, Greg Wilson, A Quick Introduction to Version Control with Git and GitHub, PLOS Computational Biology
20. Ismail Butun, Patrik ¨Osterberg, and Houbing Song, Security of the Internet of Things: Vulnerabilities, Attacks and Countermeasures, IEEE Communications Surveys & Tutorials 22(1):616-644, 2019
21. Miao Yu, Jianwei Zhuge, Ming Cao, Zhiwei Shi and Lin Jiang, A Survey of Security Vulnerability Analysis, Discovery, Detection, and Mitigation on IoT Devices, Future Internet 2019, 12, 27; doi:10.3390/fi12020027.

AUTHORS PROFILE

Hoai-Nam Nguyen, received the BSc. in University of Engineering and Technology of Vietnam National University in 2009, MSc. in Hof University of Applied Science, Germany in 2015. Currently working at the Institute of Information Technology of Vietnam Academy of Science and Technology. Research fields: Cybersecurity, software engineering.

Truong - Thang Nguyen, received a Ph.D. in 2005 at the Japan Advanced Institute of Science and Technology (JAIST), Japan. Currently working at the Institute of Information Technology, Vietnam Academy of Science and Technology. Research fields: software quality assurance, software verification, program analysis.

Thu-Nga Nguyen Thi, received MSc. in University of Information and Communication of Thai Nguyen. Currently working at the Institute of Information Technology of Vietnam Academy of Science and Technology. Research fields: Cyber security, crypto.

Manh-Dong Tran, received a M.S. degree in 2013 at the University of Engineering and Technology, Vietnam National University, Hanoi. Currently working at the Institute of Information Technology, Vietnam Academy of Science and Technology. Research fields: software quality assurance, software verification, program analysis.

Ba-Hung Tran, receivved a Master degree in 2014 at Military Technical Academy working at the Institute of Information Technology. Research fields: Network Infrastructure, software verification, program analysis.