Towards Application Portability on Blockchains

Kazuyuki Shudo
Tokyo Institute of Technology
Tokyo, Japan
Email: shudo@c.titech.ac.jp

Reiki Kanda
Tokyo Institute of Technology
Tokyo, Japan
Email: kanda.r.aa@m.titech.ac.jp

Kenji Saito
Keio University
Kanagawa, Japan
Email: ks91@sfc.wide.ad.jp

***Abstract*—We discuss the issue of what we call *incentive mismatch*, a fundamental problem with public blockchains supported by economic incentives. This is an open problem, but one potential solution is to make application portable. Portability is desirable for applications on private blockchains. Then, we present examples of middleware designs that enable application portability and, in particular, support migration between blockchains.**

***Index Terms*—blockchain, incentive mismatch, application portability, migration between blockchains**

I. Introduction

Blockchain is extending its field of applications, not limited to cryptocurrency, due to its ability to provide us a kind of trust even with no centralized entity such as a government as shown in Figure 1.

Incentive mismatch is a fundamental problem with public blockchains. The incentive for nodes to support a blockchain is economic, i.e., gaining coins, and is different from the incentives for blockchain applications. An application cannot continue working if its underlying blockchain collapses due to the economic motivation disappearing. It is non-trivial to align the blockchain node and application incentives, and this is still an open problem.

Making applications portable is one potential solution to protecting them against collapsing along with their underlying blockchains. Portability is also desirable for applications on private blockchains. In fact, minimizing the dependence of applications on their underlying middleware is a well-known best practice.

However, current blockchain middlewares provide their own application programming interfaces (APIs), making an application written for one middleware difficult to port to another. It is not even clear that it would always be possible to design common API functions, because each middleware uses its own abstractions, such as Ethereum's Solidity language.

In this paper, we introduce the incentive mismatch problem and discuss application portability as a potential solution. Then, we present software architectures and techniques for enabling application migration between blockchains.

II. Benefits of application portability

Portability is a desirable property for applications on both public and private blockchains, although the reasons are somewhat different.

978-1-5386-4870-4/18/$31.00 ©2018 IEEE

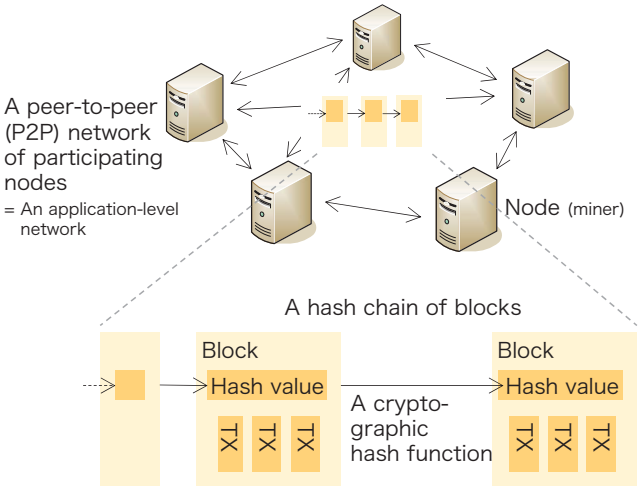

Fig. 1. A blockchain depicted as a distributed system.

A. Applications on public blockchains

Public blockchains have a fundamental problem we call *incentive mismatch*: blockchain nodes and applications do not share a common incentive. Public blockchain transactions are confirmed by Proof of Work [1], and derivatives such as Proof of Stake [2] and Proof of Elapsed Time [3], that are based on the nodes' economic incentives. The nodes simply try to gain coins and have no direct incentive to support particular applications.

If a public blockchain cannot provide sufficient economic incentives to its supporting nodes, it loses the ability to confirm transactions securely. For example, a fall in coin prices might lead to some of its nodes defecting, and reducing its ability to confirm transactions. Blockchains with fewer supporting nodes are also more vulnerable to attacks such as majority (51%) attacks and eclipse attacks [4], [5]. For example, in May 2018, attacks against public blockchains supporting cryptocurrencies succeeded in a row. In case of Monacoin, attackers voided over 20 blocks that had previously been confirmed by a block withholding attack. In case of Bitcoin Gold, attacker voided 22 or more blocks and succeeded double-spending of coin. This means that blockchain applications have no control over the confirmation ability they are based on and, hence, they can collapse due to economic circumstances.

Is it possible to align the incentives of the nodes supporting

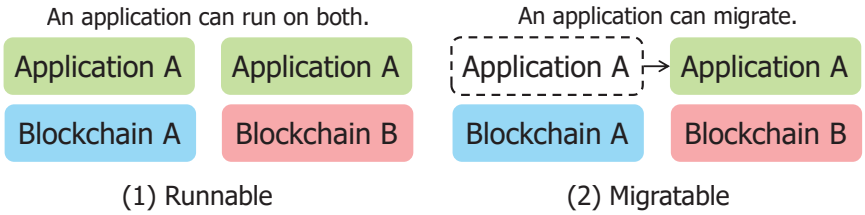

Fig. 2. Two application portability levels.

a blockchain and the applications on it? The currency system that supports a public blockchain is itself a blockchain application, because it utilizes the blockchain's confirmation ability. Their incentives match by design; however, this is a special case. The incentives also match if the same entity both deploys and operates the blockchain nodes and runs the application; however, in that case it essentially becomes a private blockchain.

Otherwise, public blockchains and their applications generally have no common incentive. Whether or not it is possible to align their incentives, i.e., design a mechanism that enables applications to ensure they retain this confirmation ability, is an open problem. However, making application portable is one potential solution to this problem: if an application is portable (i.e., migratable), we can simply migrate it to another blockchain if the current one collapses (Section III).

B. Applications on private blockchains

Portability is also a desirable property for applications on private blockchains. If it is difficult to port a given application, it may suffer from so-called vendor lock-in, preventing, for example, the application and its users benefiting from a different and better middleware platform. For example, an application and its users cannot benefit from a better another middleware. At worst, this means the application will die along with its underlying middleware if that middleware becomes too old and stale to use.

III. Application Portability

There are several different levels of application portability. As Figure 2 shows, here, we focus on the following two levels.

1) *Runnable* : The application can run on different blockchains, but cannot migrate from one to another.
2) *Migratable* : Both the application and its data can migrate from one blockchain to another.

Using a common blockchain API enables *runnable* portability. Applications are also clearly portable between different blockchains running the same middleware. However, it is less clear whether we can design common API functions for use by different middlewares, because they often adopt different abstractions, for example adopting a directed acyclic graph (DAG) instead of a hash chain of blocks. In addition, each middleware has its own smart contract mechanism, such as Ethereum's Solidity language. However, applications can still be portable if we limit the blockchain functions they use and provide a common API for this (more limited) set of functions. For example, all blockchains should support storing a hashed value together with its time stamp, and we can provide a common API for that function.

If an application is portable at the *runnable* level, it can only be migrated to another blockchain by restarting it in its initial state and accepting the loss of all the accumulated data. Because proof of data existence of and verification of state changes are fundamental blockchain features, migration without the logs needed to enable those features (Section IV-A) is unlikely to be useful. In contrast, the *migratable* portability level enables applications to survive both the loss of private blockchain middleware (Section II-B) and the collapse of an underlying public blockchain (Section II-A).

IV. Migrating Applications Between Blockchains

In this section, we discuss a method of enabling application migration between blockchains, and present preliminary middleware designs based on it.

A. Data to be migrated

What data should be migrated between blockchains? When an application that does not depend on a blockchain is migrated between middlewares, it is generally sufficient to migrate its current state, for example, records in relational databases. However, one of the strongest reasons for using blockchains is that they can prove data existed at a given time in the past and verify state (data) changes. In this case, migrating just the application's current state is not enough; the logs enabling these proof and verification processes must also be migrated. Thus, we need to migrate the following two types of data.

1) *Current state* of the application
2) *Logs* – metadata of the states that describe state changes and their time stamps.

In currency applications such as Bitcoin, for example, the former data are account balances, although, in Bitcoin, these are not explicitly recorded and can instead be calculated by adding up related transactions. The latter data are transactions, that describe balance changes and their time stamps.

B. Middleware design that enables migration

We adopt the following two principles to design migration-friendly middleware.

- Minimize dependence on specific blockchain middleware. The only requirement for our middleware design is to

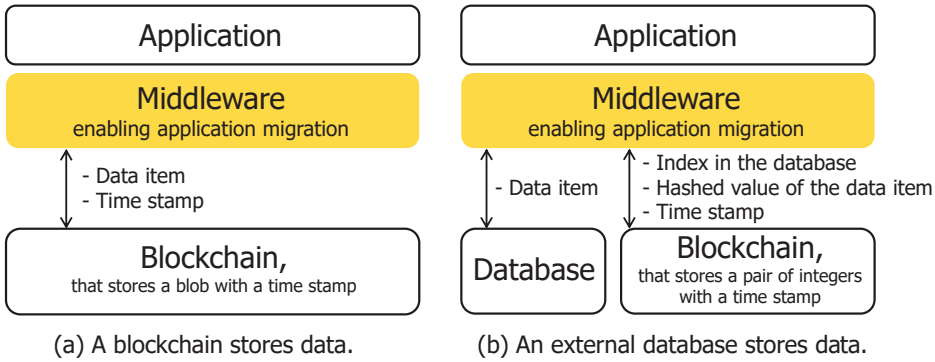

Fig. 3. A software architecture that improves application portability by limiting dependency on a blockchain.

support the storage of simple data items, such as numbers or byte sequences, together with associated time stamps.

- Do not expect to be able to retain trust in the original (source) blockchain.
  One of the main reasons for an application to leave its current blockchain is imminent collapse of the blockchain, for example, due to the loss of too many nodes (Section II-A). In such a situation, we cannot continue to rely on the previous blockchain.

Figure 3 shows two candidates for middleware designs that enable migration between blockchains. To minimize the dependences discussed above, these only expect the underlying blockchain to support recording a set of numbers and a time stamp with its indexing key (e.g. account ID such as Bitcoin address). At most, we expect a time-stamped byte sequence. We will investigate enabling migration while supporting more features than this in future work.

In the design shown in Figure 3 (b), we have chosen to store the data in an external database, not in the blockchain, although the blockchain still contains information that must be migrated. The Beyond Blockchain One (BBc-1) blockchain middleware [6] adopts such a design. This approach reduces the blockchain size, which is advantageous because the blockchain is copied to all nodes and thus occupies storage space on all of them. In addition, if an application does not require certain state (data) changes to be verified, we can update the data simply by overwriting it in the database, further reducing the storage requirements.

With the design shown in Figure 3 (b), we must also ensure the database is sufficiently fault-tolerant to achieve enough data availability. We can achieve this by utilizing replication or erasure coding [7], which are supported by most distributed databases, and adjusting the fault tolerance level by configuring the number of replicas or the erasure coding parameters. This flexibility is an advantage over the design shown in Figure 3 (a) that simply copying everything to all the blockchain nodes. Currently, such highly-available databases are provided as public cloud services and we can even store data in multiple databases instances across different cloud providers.

(1) Runnable
A common API and language enable it.
(2) Migratable
Verification of state (data) changes is
(2-1) required.
In Fig. 3(a), the source blockchains have to be kept.
In Fig. 3(b), update history, i.e. all the versions of data have to be kept.
(2-2) not required.
In Fig. 3(a), the source blockchains can be abandoned after copying metadata required to prove the data exist.
In Fig. 3(b), data can be overridden.

Fig. 4. Design choices for blockchains supporting application migration.

In the design shown in Figure 3 (b), if an application does not require verification of state (data) changes, we can overwrite data in the database when they are updated. It reduces the amount of storage occupation further.

Figure 4 summarizes the design choices for blockchains supporting application migration.

C. Migration process

If an application requires access to all the logs (Section IV-A) in the original (source) blockchain, the middleware must provide such access. However, we do not expect to be able to continue trusting the source blockchain (Section IV-B), for security reasons. Furthermore, it is safer not to rely on the source blockchain's middlewares to still be running and accessible online. In any case, we cannot generally control public blockchains, and maintaining private blockchain middleware requires human effort and CPU resources. Thus, we need a way of maintaining access to previous blockchain logs without requiring the original blockchain to continue to be trustworthy, or even exist.

Figure 5 shows a migration process that meets these requirements. The middleware stores a static copy of the source blockchain, truncating it at the expiration time, and accesses the stored blockchain. Here, the middleware sets the source blockchain's expiration time to just before the migration. The

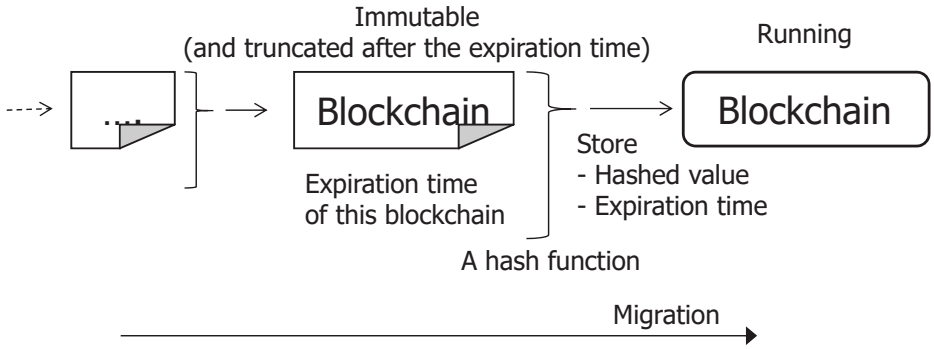

Fig. 5. Technique for migrating an application between blockchains.

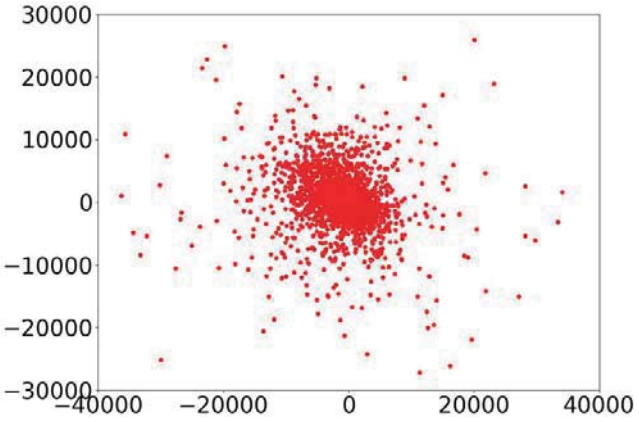

Fig. 6. A network coordinate representing about 10,000 Bitcoin nodes (in millisecond).

expiration time would be specified in block height because real-time time stamps can be manipulated to some degree. Saving a static copy obviates the need for the blockchain middleware to continue running, and truncating it allows us to stop trusting the blockchain after the expiration time.

Then, the middleware creates a hash chain of the saved blockchains as shown in Figure 5. This hash chain enables it to detect any alterations of the saved blockchains, thus, proving it has not been modified, and confirm the stored expiration time is correct.

Here we have to remember that it is possible for a once-confirmed block to be voided in a public blockchain. It also happens naturally as well as by attacks mentioned in Section II-A. It seems that an application on the blockchain can just ignore such invalidation, because the application relies only on the saved blockchains, not still-running old blockchains. Though there may be cases in which the application has to take account of the block invalidation. In those cases the statically saved blockchains are also to be updated. At least, expiration time of the saved blockchain has to be rewinded to a still-valid block. Anyway, we want to avoid such updates, that are time-consuming and halt the application for the updates. We are trying to reduce block propagation latency among nodes, that is the reason of naturally-happening fork and resulting block invalidation. The techniques for reducing would be neighbor selection or route selection [8]. Such neighbor and route selections for performance requires network proximity information between nodes. Therefore, first, we are trying to establish a technique to estimate communication latencies between nodes using network coordinates such as Vivaldi [9]. Figure 6 shows communication latencies among Bitcoin nodes estimated using Vivaldi on a two-dimensional Euclidean plane. The next step in this activity is accuracy improvement.

The middleware also has to be able to interpret all possible source blockchains, which may be in a variety of formats. However, developing such interpreting functions takes significant effort, and the resulting software is likely to be complicated and bug-prone. Therefore, we would prefer to simply abandon the source blockchains if the application allows it. For applications that need to prove data exists but not verify state changes, we can abandon the source blockchain after copying the metadata needed to prove the data exist (and the data themselves in the case of Figure 3 (a)) to the new blockchain. If we take this approach, though, how can we prove the copied metadata are correct without the original metadata? The trust provided by blockchains is based on verifiable data; however, such a copying approach means the copied metadata are not verifiable because they are not explicitly connected to the original metadata. Although the application trusts the middleware that performed the copy, trust based on software correctness is weaker than trust gained through verifiable data because, for example, the middleware is subject to human mistakes. The next-best solution to this problem would be to make the copying program verifiable, possibly by saving both it and its hashed value somewhere.

V. Conclusion

Applications for public blockchains are subject to the particular issue of incentive mismatch between the applications and the blockchain nodes. This is because public blockchains are driven by different economic incentives than the applications that run on them, and are not under the applications' control. Because of this, an application may also fail if its

underlying blockchain collapses, unless it is not just portable but migratable.

These blockchain application survivability issues have motivated us to investigate application portability and migration. In this paper, we have presented middleware designs that facilitate application migration between blockchains, together with more efficient alternative designs for applications with fewer requirements.

Making applications portable and migratable is both generally desirable and a potential solution to the incentive mismatch problem with public blockchains. It would be better still for the blockchain to be supported by the applications themselves, not the nodes' economic incentives. We can call it an *incentive-matched blockchain* if the public blockchain is always supported solely by the applications. However, how to design such a mechanism remains an open problem.

Bodies such as the ISO have begun standardizing distributed ledgers, and the IETF and W3C have discussed such matters as well. However, it is challenging just to design an effective common blockchain API and migration between blockchains lies even further in the future. Today, software is implemented first and its specification is written after it gains much popularity. It is implementation-first, not specification-first. For instance, the first Bitcoin Improvement Proposal (BIP) came more than two and a half years after the Bitcoin network was launched. Following this paper, our effective next step will be to demonstrate applications that are portable and migratable.

In future work, we plan to study the portability of program code and data for smart contracts, a topic this paper has been barely touched on. Even on the same middleware platform, it is difficult to update the data format or program code while retaining existing data.

ACKNOWLEDGMENTS

We thank Yuto Takei, Masashi Hojo, and Shigeya Suzuki for discussions on a number of topics, such as the incentive mismatch problem and future incentive-matched public blockchains. We also thank Toshio Koide for pointing out the significance of portability on private blockchains. This work was supported by Kaula, Inc., the SECOM Science and Technology Foundation, the New Energy and Industrial Technology Development Organization (NEDO), and JSPS KAKENHI Grant Numbers 2570008 and 16K12406.

REFERENCES

[1] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Proc. CRYPTO'92*, pages 139–147, August 1992.

[2] QuantumMechanic. Proof of stake instead of proof of work. https://bitcointalk.org/?topic=27787.0.

[3] Hyperledger Sawtooth. https://sawtooth.hyperledger.org/.

[4] Arthur Gervais, Hubert Ritzdorf, Ghassan O. Karame, and Srdjan Čapkun. Tampering with the delivery of blocks and transactions in Bitcoin. In *Proc. ACM CCS 2015*, October 2015.

[5] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on Bitcoin's peer-to-peer network. In *Proc. USENIX SEC'15*, pages 129–144, August 2015.

[6] Kenji Saito and Takeshi Kubo. BBc-1 : Beyond Blockchain One – an architecture for promise-fixation device in the air –. Available electronically at https://beyond-blockchain.org/public/bbc1-design-paper.pdf.

[7] Daniel Ford, François Labelle, Florentina I. Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan. Availability in globally distributed storage systems. In *Proc. OSDI'10*, October 2010.

[8] Takehiro Miyao, Hiroya Nagao, and Kazuyuki Shudo. A method for designing proximity-aware routing algorithms for structured overlays. In *Proc. IEEE ISCC'13*, 2013.

[9] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. In *Proc. ACM SIGCOMM 2004*, September 2004.