Turing Completeness and Active Networks

Bhagyashree Hemant Parkar

Rutgers, The State University of New Jersey

Abstract

Beginning around 2008, when it was first referred to, blockchain innovation addresses a development from both a primary and application perspective. From that point forward, because of its eccentricities and abilities of executing shrewd agreements, blockchain innovation has gone through a solid advancement in various application spaces. The interest around this innovation likewise brought to the meaning of a few stages working with its utilization and application. Due to their assortment, , picking the most reasonable blockchain stage to help a particular business need addresses an essential issue. This study aims to figure out the Blockchain technology, concept of Turing complete programming languages and Active Networks, relationship between them and all the issues that are related to both of them. We are going to explore all these things with respect to blockchain technology.

Introduction

Blockchain innovation has accomplished an ever increasing number of interests somewhat recently. It was proposed for supporting and overseeing trade and installment frameworks dependent on cryptographic forms of money [1], generally impacting the monetary business. Taking influence of its fundamental benefits, like decentralization, permanence, and straightforwardness [2], from the start, it has been taken advantage of to foster a huge assortment of different applications, for various use cases and application spaces, for example, monetary administrations, chain supply, IoT, banking, producing, etc, and changed much in the way normal regular articles (e.g., cell phones, vehicles) and individual information (e.g., individual recognizable proof, medical care) are utilized [3].

Because of the huge assortment of utilization cases and application areas, explicit blockchain innovation will most likely be unable to meet the necessities for all client situations. Along these lines, picking the most appropriate blockchain stage to execute savvy contracts in a particular business setting is an essential issue. Every stage has its own highlights that make it pretty much appropriate to be utilized for explicit necessities.

Before current PCs, Alan Turing estimated that there would one day be a machine that could tackle any issue. This machine became known as the Turing Machine. Alan envisioned his machine as a long piece of tape with data composed on it as paired code (1s and 0s). The machine would likewise have a perused/compose head that moves along the tape perusing each square, individually. The code would ask the machine a computational issue, and the tape would be insofar as was expected to accomplish an answer.

As the head moves along the tape, the machine adheres to basic guidelines that oversee how it responds. It peruses the tape, adheres to the guidelines,

and plays out a specific activity to compose another code as it moves along. This new example of code is the response to the issue. Turing's theoretical machine could answer any computational issue that could be communicated in code.

A gadget or programming language is viewed as Turing Complete when it can repeat a Turing Machine by running any program or tackling any issue the Turing Machine could run or settle. Then again, assuming that a gadget or programming language can't do it, then, at that point, it is supposed to be Turing Incomplete. Turing Completeness says "this vehicle can go anywhere any other vehicle can go." That is, you can compute all the same functions. Input to output, start to end. A straightforward number cruncher is an illustration of a framework that is Turing Incomplete since it can just do a couple of kinds of estimations. Conversely, a programmable logical mini-computer (ready to play out a wide range of estimations) can be considered as a Turing Machine [4].

Active information networks are fit for conveying modified projects to the hubs of the organization. Conventional information networks inactively move bits starting with one framework then onto the next. In a perfect world, the client information is moved darkly, that is, the organization doesn't focus on the pieces it conveys and they are consequently moved between end frameworks without adjustment. Processability in such organizations is incredibly restricted, e.g., header handling in bundle exchanged organizations or potentially motioning in association arranged organizations.

Active Networks break with custom by permitting the organization to perform tweaked calculations on the client information. For instance, a client of a functioning organization could send a tweaked pressure program to a hub inside the organization (e.g., a switch) and solicit that the hub executes that program when handling their parcels [6]. These organizations are "Active" in two ways:

Switches and switches inside the organization effectively process, i.e., perform calculations on the client information coursing through them.

Individual clients as well as executives can infuse modified projects into the organization, in this way fitting the hub handling to be client or potentially application explicit.

There are a few building ways to deal with Active organizations. One methodology, which I saw as especially intriguing, replaces the detached parcels of present day structures with Active "cases" - smaller than usual projects that are executed at every switch/switch they navigate. This adjust in design point of view, from aloof bundles to Active containers, at the same time addresses both of the "Active" properties portrayed previously. Client information can be inserted inside and handled by these smaller than usual projects, in much the manner in which a page's substance are installed inside a part of PostScript code. Besides, containers can summon pre-characterized program strategies and additionally plant new techniques inside network hubs.

Our work is persuaded by both innovation "push" and client "pull". The innovation "push" is the rise of "Active advances", aggregated and deciphered, supporting the embodiment, move, mediation, and protected and productive execution of program pieces. Today, Active advancements are applied inside individual end frameworks or more the start to finish network layer; for instance, to permit Web servers and customers to trade program pieces. Our advancement is to use and expand these innovations for use inside the organization - in manners that will essentially change the present model of what is "in" the organization.

The "pull" comes from the impromptu assortment of firewalls, Web intermediaries, multicast switches, portable intermediaries, video entryways, and so on that perform client driven calculation at hubs "inside" the

organization. In spite of structural orders against them, these hubs are thriving, proposing client and the board interest for their administrations.

There are three chief benefits to putting together the organization design with respect to the trading of Active projects, rather than static bundles:

Trading code gives a premise to progressively versatile conventions, empowering more extravagant associations than the trading of fixed information designs;

Cases give a method for executing fine grained application-explicit capacities at vital focuses inside the organization;

The programming reflection gives an amazing stage to client driven customization of the foundation, permitting new administrations to be conveyed at a quicker pace than can be supported by seller driven normalization processes.

Active organizations empower clients to perform network calculations. Thus, the administration of the organization framework becomes disseminated. Also, the organization can perform huge measures of organization handling. In this manner, I accept that dispersed administration and huge in-network handling can productively oversee energy utilization between hubs.

As energy necessities change, the framework won't need to be refreshed physically. Therefore, strong and dependable correspondence between IoT hubs is conceivable. The primary Active organizations showed up in the mid 1990s. Around then, programming characterized organizing (SDN) and organization work virtualization (NFV) didn't exist. In this way, basic plan difficulties caused significant execution troubles.

Active organizations were an extreme idea of permitting untrusted sources to execute codes on network administrators' equipment to permit quicker and simpler sending of different disseminated applications. Subsequently, it

additionally risked security infringement on numerous levels, which basically prevented the vast majority, particularly network administrators or ISPs to definitely convey it. Over the long run, the advocates of Active organizations needed to make some trade off to expand its security and feasibility. This paper resolves various issues in three spaces of Active organizations dependent on the creators' experience of utilizing ANTS, a test Active organizations execution [7].

Literature Review

Turing-complete languages

There are so-called "Turing complete languages," which are those that can compute everything that needs to be computed. Some of them are Java, C, C++, Python, etc. almost all general purpose languages are Turing complete. The exceptions with Turing completeness are things like SQL, HTML, etc.

A Turing completeness is a program that is able to follow a programming language using a basic arrangement of data. In particular, a Turing completeness is an information control framework that can chase programming languages utilizing a simplified set of data, generally, the Turing machine browses complex data sets. Turing machines are additionally helpful in tackling issues and processing the information framework that the human mind has a hard time understanding. Similarly, mini-computers allow people to more efficiently work with basic information Turing machine allows people to browse and comprehend complex information structures.

It is the exceptional arrangement of guidelines and information orders, over and above any other article, that cannot be effectively unraveled without the use of the Turing machine. Such a machine is considered Turing-complete. When it comes to digital currencies, Ethereum is regarded as Turing-complete. There is also a tape, and even blockchain used to be Turing-complete to tackle language issues[5].

Impact of Turing Completeness on Cryptocurrency

On schedule, innovation and programming dialects have achieved an upset in the contributing business. The idea of digital money was made by the utilization of the programming language Turing-finished over the coin. These aides control and ensure the monetary exchanges. The utilization of digital currency permits individuals to share cash and move assets without the inclusion of outsiders (like banks or monetary foundations). Individuals set

aside cash by staying away from outsiders, since there are no extra charges or exchange costs.

Today, most cryptographic forms of money are Turing-finished, with the particular code of dialects composed over the monetary standards requiring the (Turing) machine to do the encoding and perusing. Turing-complete cryptographic money keeps hidden data protected and minimal. In many cryptographic forms of money, the data encoded over the surface is Turing-finished and can be utilized to solve issues with codes.
Virtual monetary standards are something other than creative mind, they take part in reality and have Turing culmination. Yet, not all virtual monetary forms are Turing finished.

Active networking

Active networks can be either discrete or integrated, depending on whether data and programs are transmitted separately or together. Our next contribution is to provide a high-level description of how active nodes might be organized and to describe a node programming model which could facilitate cross-platform interoperability.

A Discrete Approach

The processing of messages may be architecturally separated from the business of injecting programs into the node, with a separate mechanism for each function. Users send their packets through such a node much the way they do today. When a packet arrives, its header is examined and a program is dispatched to operate on its contents. The program actively processes the packet, possibly changing its contents. A degree of customized computation is possible because the header of the message identifies which program should be run - so it is possible to arrange for different programs to be executed for different users or applications.

The separation of program execution and loading might be valuable when it is desirable for program loading to be carefully controlled or when the individual programs are relatively large. For example, program loading could be restricted to a router's operator who is furnished with a "back door" through which they can dynamically load code. This "back door" would at minimum authenticate the operator and might also perform extensive checks on the code that is being loaded. Note that allowing operators to dynamically load code into their routers would be useful for router extensibility purposes, even if the programs do not perform application- or user-specific computations.

Capsules - An Integrated Approach

A more extreme view of active networks is one in which every message is a program. Every message, or capsule, that passes between nodes contains a program fragment (of at least one instruction) that may include embedded data. When a capsule arrives at an active node, its contents are evaluated, in much the same way that a PostScript printer interprets the contents of each file that is sent to it.

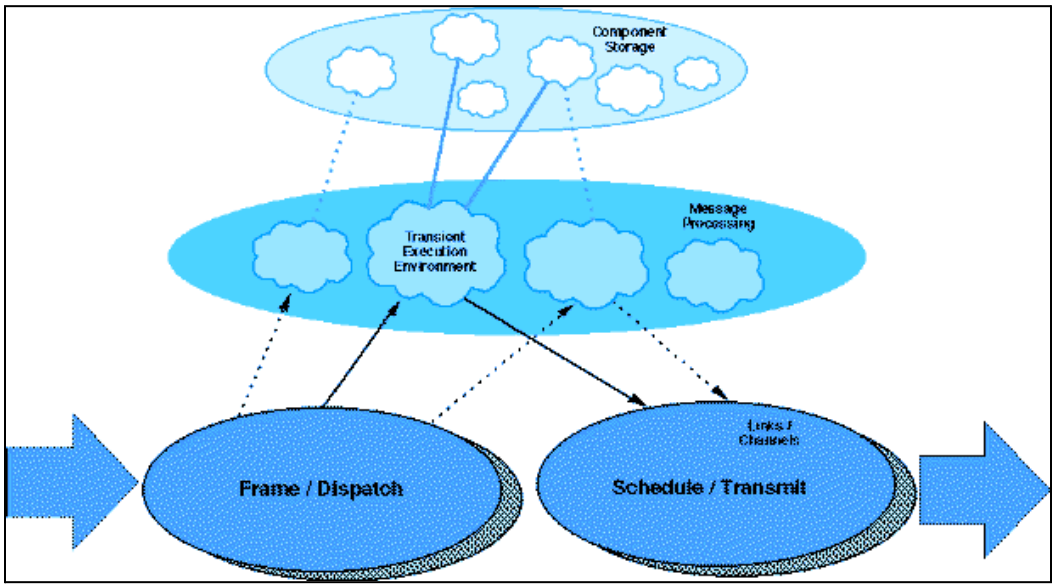

Figure 1: Active Node Organization

Figure 1 provides a conceptual view of how an active node might be organized. Bits arriving on incoming links are processed by a mechanism that identifies capsule boundaries, possibly using the framing mechanisms provided by traditional link layer protocols. The capsule's contents are dispatched to a transient execution environment where they can safely be evaluated. We hypothesize that programs are composed of "primitive" instructions, that perform basic computations on the capsule contents, and can also invoke external "methods", which may provide access to resources external to the transient environment. The execution of a capsule may result in the scheduling of zero or more capsules for transmission on the outgoing links and/or changes to the non-transient state of the node. The transient environment is destroyed when capsule evaluation terminates.

To develop a new service, the first step is to write a new set of forwarding routines implementing the desired behavior using a subset of Java (16KB max code size). The ANTS toolkit provides a core set of APIs that restricts what sort of routines can be developed. Once the code is written, it is signed and put up on a directory service. Anyone, who wants to use that service, can look up the directory, fetch the code and register the service with the local active node. Once the local node calculates the types it is expected to find in capsules for that particular service and disseminate the code throughout the network (through caching and lazy dissemination), the service is ready to go. Capsule processing is pretty straightforward once the code is distributed: after reception, capsules are first demultiplexed using their type field to find references to code, which are then securely executed inside sandboxes, and the process goes on till capsules reach their destinations. There are additional mechanisms such as TTL, fingerprints etc. to provide security at the cost of performance [7].

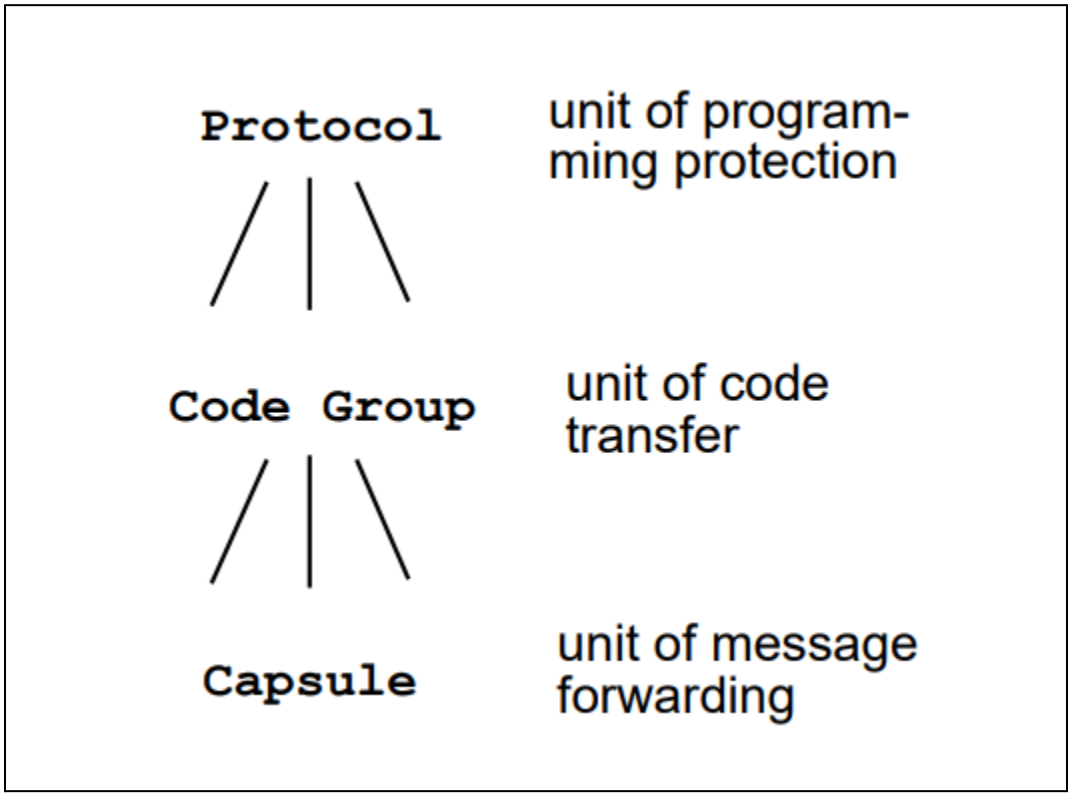

Figure 2: Capsule Composition Hierarchy

In Figure 2, the relationships between capsules, code groups, and Protocols have been illustrated.

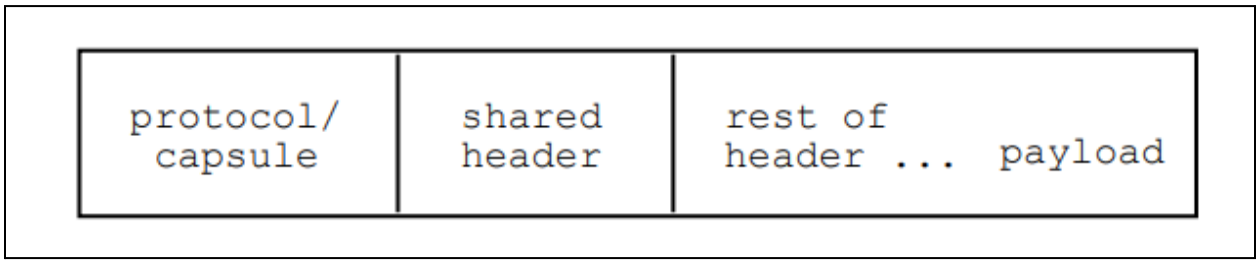

Figure 3: Capsule Format

Figure 3 illustrates that each capsule carries an identifier for its protocol and particular capsule type within that protocol. The identifier is based on a ngerprint (e.g., the MD5 message digest) of the protocol code. It is used for

demultiplexing to a forwarding routine in the same sense as the Ethernet type and IP version and protocol elds [9].

The active networks were rejected by the military because of the issues that were faced. While capsules provide a more secure way of code dissemination (using fingerprints) and can be performance optimized (by carrying references to code instead of carrying the actual code, using caching), there is still a performance tradeoff. It is argued that performance overhead of using capsules is minimal and can be attributed mostly to the use of Java.

Making active networks more open attracts security risks in terms of protection against attacks as well as from a global and local resource management perspective. To mitigate threats, it is proposed using (hierarchical) fingerprinting, restricted API, and strict isolation between different services' codes. To avoid resource management threats, it is proposed to use partial solutions using TTL, restricted API, and eventually fall back to using trusted authority provided digital signatures.

Like traditional networks, active networks are concerned with the authenticity, integrity, and confidentiality of the data going through the network. However, traditional networks are concerned only with possible damage to user data and end nodes. Active networks share these concerns but must also consider possible damage as the active packet moves into each node and EE. Active nodes could be harmed by active code, either because the code modifies the node's state or because it drains resources (essentially launching a denial-of-service attack). Thus, enforcing protections at end nodes is not sufficient for active networks. Securing an active network means that protection mechanisms must move into each node and each EE. Protecting the network as a whole is only possible by building a common protection mechanism into the design of individual nodes and EEs.

This could be achieved by the following sequence of tasks:

1. **Validation**

Ensuring the program is indeed the correct program. This is commonly achieved by encoding the contents of a packet using a cryptographic hashing algorithm such as the MD5 message digest [32], and carrying this encoding in the packet header. Any packet that fails the validation check is subjected to either dropping or some default forwarding behavior.

2. **Authorization**

Ensuring the program comes from an authorized user. Once a program is validated, an Access Control List (ACL) is consulted. Packets that fail the authorization are handled either by some node-specific or user-specified default processing.

3. **Execution**

Based on information from the authorization phase, the run-time environment enforces the resource usage and access limits on the program's execution. These limits include the maximum amount of time a program can spend running at a node and the amount of memory that can be accessed.

4. **Fault Detection**

Any program faults that may occur during execution are seen as attacks, and must be caught and handled efficiently to prevent harm to the correct operation of the node. Handling of the fault/attack should not disrupt services to existing flows.

A strong security architecture should address all the concerns listed above, but still be as lightweight as possible. However, providing an optimal solution is a very complex problem.

With active networks, security protections travel with the packet so that appropriate protections can be chosen dynamically at nodes to suit the environment through which the packet passes. Different users and organizations have different security requirements and as a result, security systems need to support dynamic interoperable security policies to enforce proper security measures and access control for packets. This necessity gives rise to several challenges.

- How can these security policies be defined for the different uses of an active
- network, i.e., different permission levels at a node?
- The ability to negotiate a common set of security services between two or more administrative domains is required. How can differing policies be reconciled?
- How can the security architecture scale to handle a growing population of users with different interests?
- Not only must protection be guaranteed within a node, but protection must also exist on a per-user basis, i.e., it must be possible to protect one user's packets from those of another user.

The requirements on a security mechanism are also dependent on the set of functions a node exports and the expressiveness of the language used to create active network programs. Functions that enable access to node resources and modification to node state are extremely dangerous, requiring tight control over their use. Furthermore, if an expressive language, such as Java, is used to program packets, a node must be able to detect program logic that would behave in a malicious way (e.g., an infinite loop) and prevent the program's execution as early as possible. The ideal solution to security should be both lightweight and scalable. Most of the proposed solutions offer strong security measures but are costly in time, computation, or the number of messages needed to retrieve keys, especially if a public key infrastructure

is in place. Even if the proposed security services are lightweight, this often comes at the cost of less flexibility in what a packet can do at a node. The point is that there is no one perfect, generic solution to the security problem; each solution requires the tradeoffs in one or more areas [9].

The languages behind blockchain

The programming languages used in these blockchains are pretty basic, often no more complex than the simplest code that arrived soon after the first computers were built. Bitcoin Script, for instance, has only a few permitted operations, and the data is pushed and pulled from a virtual stack. There are no higher-level constructs such as loops, a limitation to ensure that the code runs quickly to conclusion.

Programmers who want to write smart contracts or other code on the blockchain must be ready to confront different challenges. First, they must create something concise and extremely efficient, because everyone double-checking the blocks will execute it. Second, they must imagine what happens when the code is run on different nodes at different times.

The consequences for making a mistake can be terrible, because a smart criminal may figure out a way to trigger a payout.

The simplicity of the languages has led some developers to create higher-level, more feature-rich languages that are closer to what is used for most modern code. Some of the latest options are more elaborate and must be compiled into the operations that are embedded in the blockchain.

Oldies but goodies

Not all of the languages underlying the projects are new. Other developers have wondered whether we need to create something different, and they've been experimenting with tweaking and limiting old, general-purpose

languages that deliver the same features as newer languages while maintaining a connection to more familiar syntax. This approach can make it easier for developers to integrate their existing code with blockchains.

Serpent, for example, was one of the original languages for Ethereum; it was a low-level, assembly-like sequence of operands. The language was deprecated recently, and developers have steered away from it. Some of the code written in it may still live on in the blockchain, but it is generally referred to only for historical reasons.

Mutan is another of the original languages for Ethereum that has been set aside since 2015. It was modeled after the popular network language Go, but many of the developers who enjoyed it have since moved to Solidity [10].

Conclusion

This paper does an excellent job in summarizing the active networks landscape as well as providing an overview of the challenges that exist toward its realization, complimented by solution hints. Active and programmable networks were one of the very few really awesome networking concepts that could do a lot in changing the research and commercial landscape. Unfortunately, at that time the available hardware was not fast nor programmable enough. In addition, there was and still is a lot of skepticism and prejudice among network operators regarding opening up their networks to untrusted sources.

However, in the last couple of years, network virtualization has emerged as the solution for a secure, flexible, and extensible future networking platform that combines the programmability of active and programmable networks with concepts of overlay networks and VPNs. The main reasons behind this is, first and most importantly, programmability nowadays is cheaper and faster (with the wide-spread use of FPGAs and/or network processors) and secondly, using so-called “safe” hardware or execution environments do not take as much performance hit as they used to a decade ago.

A lot of research effort is put into development of Active Network or similar technologies. Being a highly dynamic runtime environment that supports a variety of network services, and allows injection of newly designed services into the infrastructure, active network deployment raises a lot of concerns. Various research groups are researching security being the most important of these concerns. Other crucial research topics include routing, resource allocation, network management services and most important of all mobility. A lot of research effort has been put into the deployment of demo active networks and standardization of all these efforts. Most of the effort is involved in parallel deployment of a few different programming models

providing an opportunity to explore alternatives. The possibilities offered by Active Networking technologies have already begun to change our perception of a computer network and would play an important role in the shaping of the future technologies.

Active networks appear to break many of the architectural rules that conventional wisdom holds. However, we believe that they build on past successes with packet approaches, such as the Internet, and at the same time relax a number of architectural limitations that may now be artifacts of previous generations of hardware and software technology.

In summary, passive network architectures that emphasize packet-based end-to-end communication have served us well. However, as our lead users demonstrate, computation within the network is already happening - and it would be unfortunate if network architects were the last to notice. It is now time to explore new architectural models, such as active networks, that incorporate interposed computation. We believe that such efforts will enable new generations of networks that are highly flexible and accelerate the pace of infrastructure innovation.

Most of these languages are new, but the enabling of well-audited code built from simpler constructs is working its way back to more traditional code. The code that runs on the server and handles much of the transaction processing is often written in popular languages such as C++, Rust, Java, or Python. Most programmers who are building smart contracts won't need to explore this system programming layer, but if you're going to be working on enhancing mining, or creating some wallets or other infrastructure, there's a good chance you'll be exploring these code bases and using these languages.

This software for the infrastructure must be developed just as carefully as any other crypto-code, and so developers are starting to ask the same questions

about the underlying code as they do about the code explicitly running in the blockchain.

To conclude, the Active network was too futuristic for its time and no one is still completely sure whether its time has come yet!

References

[1] S. Nakamoto, Bitcoin: a peer-to-peer electronic cash system, https://bitcoin.org/bitcoin.pdf, 2008

[2] F. Casino, T. K. Dasaklis, C. Patsakis. A systematic literature review of blockchain-based applications: current status, classification and open issues. Telematics and Informatics, Vol. 36, pp. 55–81. March 2019. doi: 0.1016/j.tele.2018.11.006

[3] S. N.Khan, F. Loukil, C. Ghedira-Guegan,E. Benkhelifa, A. Bani-Hani "Blockchain smart contracts: Applications, challenges, and future trends", Special Issue on Blockchain for Peer-toPeer Computing, on-line Apr. 17, 2021, https://doi.org/10.1007/s12083-021-01127-0

[4] Turing Complete: https://academy.binance.com/en/glossary/turing-complete

[5] Turing Completeness and cryptocurrency: https://freemanlaw.com/turing-completeness-and-cryptocurrency/

[6] Towards an Active Network Architecture: http://www.sce.carleton.ca/netmanage/activeNetworks/mmcn96.html

[7] Active Network Vision And Reality: Lessons From A Capsule-Based System: https://www.mosharaf.com/blog/2009/10/29/active-network-vision-and-reality-lessons-from-a-capsule-based-system/

[8] ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols David J. Wetherall, John V. Guttag and David L. Tennenhouse. Submitted to IEEE OPENARCH'98, San Francisco, CA, April 1998

[9] A Survey of Active Networks, Adel A. Youssef, Department of Computer Science, University of Maryland, College Park, MD 20742, CS-TR-4422. May 1999

[10] 23 blockchain languages driving the future of programming, Peter Wayner.
https://techbeacon.com/app-dev-testing/23-blockchain-languages-driving-future-programming