Operations: Turin Complete & Gas Fee

Table of Content

Introduction

Consider the humble pocket calculator and home computer. Between the two, which gadget is more functional? The obvious answer is the home computer, as it can run programs for a multitude of computational problems — this makes it Turing complete. As for the calculator, it's only equipped to handle a limited range of mathematical operations and is thus considered Turing incomplete.

As the above illustration demonstrates, Turing completeness refers to the extent of a system's ability to execute any computational task. The more computational functions a system is able to perform, the more "Turing complete" it becomes.

Turing completeness has been applied to the world of blockchain to assess the capabilities of any given blockchain. Between Bitcoin and Ethereum, for example, Ethereum is Turing complete as it enables developers to write multifaceted programs in Solidity and run them on the Ethereum Virtual Machine (EVM).

The History of Turing Complete Systems

As a concept, Turing completeness is rooted in the early days of theoretical computer science, long before the advent of today's sophisticated computers.

In 1936, British mathematician and logician Alan Turing published a paper introducing a conceptual framework for a machine capable of executing an arbitrary set of coded instructions. This hypothetical device later came to be known as the Turing machine.

The Turing machine was originally imagined as having an infinite tape, segmented into boxes. Each box contains a simple instruction, coded as a one or a zero. The machine reads these instructions sequentially before replacing the original symbol with a new one, also either a one or a zero. The machine then updates its internal state, which represents a snapshot of its computational process at a given moment.

One can view Turing's hypothetical machine as the prototype of the programmable computer, paving the way for the future development of universal, programmable computers. In today's

computing landscape, Turing complete systems are ubiquitous, with the term itself used as a benchmark to gauge a system's computational capabilities.

What is Turing completeness?

Turing-completeness is a word defined by Alan Turing which it describes the idea that some computing machines are capable of performing any task a computer can perform. Turing complete refers to the idea that given infinite time, any program in one language could be written (albeit perhaps inefficiently) in another. In Ethereum, Turing complete means using conditional statements and loops to program smart contracts.

- The concept of Turing-completeness is one at the heart of software and application development, where it allows code to be written without having to check beforehand if it will work or not.
- In other words, one can write their program without worrying about what else is allowed for it to do.
- This is essential in determining usability as well as many other aspects of the software. It is also important to know what "Turing-complete" means and how it relates to Ethereum.
- In Turing's paper, the concept of Turing-complete machines is used to disprove the possibility of true artificial intelligence.
- For instance, whether a machine can eventually imitate the behaviors of a human. In practical terms, this means that "Turing-complete" allows programmers to write code that can be used by any computer to achieve any result.
- It is necessary for introducing new techniques and ideas into software programming such as functional programming or even for understanding ideas about universal computation with regard to general computing.

Does a system have to be Turing Complete to be useful in Blockchain?

A system has to be Turing complete in order to be useful in blockchain, but it can have all the other desirable properties of a blockchain, such as decentralization and trustless transactions.

- A system is Turing complete if it can simulate an arbitrary computer program. Turing complete systems have to be able to run any possible computation, which includes the most complex types of computation such as those found in blockchain. This type of system also often has better performance than other systems because it can use a set of rules which are more efficient when solving problems with many steps.
- In addition to being Turing complete, a system must also be decentralized and allow trustless transactions according to consensus in order for it to be useful in the blockchain. These properties are necessary for the security and consistency that a blockchain needs in order for its data records or "blocks" to have value and meaning.

Ethereum: The First Turing Complete Blockchain

Ethereum was the first blockchain platform to achieve Turing completeness, expanding the realm of possibilities in blockchain technology. Ethereum's Turing completeness comprises two key elements:

- The Ethereum Virtual Machine (EVM), responsible for executing these smart contracts, is also Turing complete.

- Solidity, the programming language used for building smart contracts on Ethereum, is Turing complete. It is a general-purpose language tailored specifically for the Ethereum ecosystem.

Ethereum's ability to execute a stored program, in a state machine called the Ethereum Virtual Machine, while reading and writing data to memory makes it a Turing-complete system and therefore a UTM. Ethereum can compute any algorithm that can be computed by any Turing machine, given the limitations of finite memory.

The EVM's architecture enables it to handle an expansive variety of smart contracts, even those with functionalities yet to be envisioned. This flexibility propelled Ethereum into a revolutionary phase, bringing blockchain technology from a niche utility to a versatile platform with nearly limitless applications.

Ethereum's groundbreaking innovation is to combine the general-purpose computing architecture of a stored-program computer with a decentralized blockchain, thereby creating a distributed single-state (singleton) world computer. Ethereum programs run "everywhere," yet produce a common state that is secured by the rules of consensus.

Turing Completeness as a "Feature"

Hearing that Ethereum is Turing complete, one might arrive at the conclusion that this is a feature that is somehow lacking in a system that is Turing incomplete. Rather, it is the opposite. Turing completeness is very easy to achieve; in fact, the simplest Turing-complete state machine known has 4 states and uses 6 symbols, with a state definition that is only 22 instructions long. Indeed, sometimes systems are found to be "accidentally Turing complete."

However, Turing completeness is very dangerous, particularly in open access systems like public blockchains, because of the halting problem we touched on earlier. For example, modern printers are Turing complete and can be given files to print that send them into a frozen state. The fact that Ethereum is Turing complete means that any program of any complexity can be computed by Ethereum. But that flexibility brings some thorny security and resource management problems. An unresponsive printer can be turned off and turned back on again. That is not possible with a public blockchain.

Practical Limitations to Ethereum's Turing Completeness

While Ethereum's Turing completeness is revolutionary, it's essential to acknowledge the platform's practical constraints. Theoretically, Turing completeness posits that a system can run any computation, given infinite time and resources. However, in the real-world blockchain environment, each Ethereum transaction consumes "gas," a unit that measures the computational work involved. Should a smart contract enter an infinite loop—a situation theoretically possible in a Turing complete system—it would eventually deplete its gas reserves.

This limitation is intentional. Infinite loops could cripple a public blockchain network, which operates on finite computational resources. Consequently, each transaction on Ethereum mandates a gas limit, specifying the maximum computational effort assignable to that operation. If a transaction fails to complete within this limit, it's automatically terminated.

Moreover, the vast majority of Ethereum smart contracts rarely utilize the full breadth of Turing completeness, including features like recursive loops.

Implications of Turing Completeness

Turing proved that you cannot predict whether a program will terminate by simulating it on a computer. In simple terms, we cannot predict the path of a program without running it. Turing-complete systems can run in "infinite loops," a term used (in oversimplification) to describe a program that does not terminate. It is trivial to create a program that runs a loop that never ends. But unintended never-ending loops can arise without warning, due to complex interactions between the starting conditions and the code. In Ethereum, this poses a challenge: every participating node (client) must validate every transaction, running any smart contracts it calls. But as Turing proved, Ethereum can't predict if a smart contract will terminate, or how long it will run, without actually running it (possibly running forever). Whether by accident or on purpose, a smart contract can be created such that it runs forever when a node attempts to validate it. This is effectively a DoS attack. And of course, between a program that takes a millisecond to validate and one that runs forever are an infinite range of nasty, resource-hogging, memory-bloating, CPU-overheating programs that simply waste resources. In a world computer, a program that abuses resources gets to abuse the world's resources. How does Ethereum constrain the resources used by a smart contract if it cannot predict resource use in advance?

To answer this challenge, Ethereum introduces a metering mechanism called gas. As the EVM executes a smart contract, it carefully accounts for every instruction (computation, data access, etc.). Each instruction has a predetermined cost in units of gas. When a transaction triggers the execution of a smart contract, it must include an amount of gas that sets the upper limit of what can be consumed running the smart contract. The EVM will terminate execution if the amount of gas consumed by computation exceeds the gas available in the transaction. Gas is the mechanism Ethereum uses to allow Turing-complete computation while limiting the resources that any program can consume.

The next question is, 'how does one get gas to pay for computation on the Ethereum world computer?' You won't find gas on any exchanges. It can only be purchased as part of a transaction, and can only be bought with ether. Ether needs to be sent along with a transaction and it needs to be explicitly earmarked for the purchase of gas, along with an acceptable gas price. Just like at the pump, the price of gas is not fixed. Gas is purchased for the transaction, the computation is executed, and any unused gas is refunded back to the sender of the transaction.

Drawbacks of Turing Completeness in Blockchain

The infinitive programmability of Turing complete systems, while empowering, comes with specific vulnerabilities — this is especially true for public blockchains, where code is visible to anyone. The flexibility to craft any computation creates a broad array of possible outcomes, not all of which can be foreseen. Consequently, this leaves room for disruptions like software bugs and unintended functions that could hamper the protocol's effectiveness.

In centralized systems, unforeseen issues can be quickly rectified by the entity controlling the code through prompt updates. However, the decentralized architecture of blockchain complicates this process. Any modification to the code must undergo a community-driven voting process, delaying the implementation of critical patches.

As an example, consider The DAO, a decentralized venture capital fund set up on Ethereum in 2016. While often termed a 'hack,' the incident was more an exploitation of a previously overlooked vulnerability in the contract's code. This exploit allowed a malevolent actor to

siphon over $150 million from an Ethereum smart contract, necessitating the rewind of the Ethereum blockchain to recover the stolen assets. This incident eventually gave way to the Ethereum Classic fork.
This incident was essentially a reentrancy attack, where the attacker exploited a flaw in the smart contract's code to withdraw funds illegally. Since The DAO debacle, the Ethereum community has made strides in improving coding best practices to avert similar vulnerabilities. However, the fluid nature of Turing complete systems—where new code is being constantly created—ensures that the threat of emerging vulnerabilities remains a persistent concern.

Ethereum Gas Fees

Introduction

If you have ever paid a toll on a highway, then you already know something about Ethereum gas fees. Ethereum gas fees are like paying a "toll" to use the Ethereum blockchain. Highway tollbooths may be operated by one person, but the Ethereum blockchain involves many decentralized operators.

What is Ethereum gas?

Gas is a term used in Ethereum to describe a computational unit that measures the amount of computational work needed to perform specific operations on the network. Unlike Bitcoin, where transaction fees only consider the size of a transaction, Ethereum accounts for every computational step performed by transactions and smart contract code execution. In other words, every single operation that is performed on Ethereum requires a certain amount of gas.

Complexity

The amount of gas required for an operation depends on its complexity. More complex operations require more computational resources and therefore require more gas to be executed. For example, a simple transaction that involves sending ETH from one address to another may require less gas than a complex smart contract that executes multiple operations or interacts with multiple other contracts.

State of the Network

Gas costs can also vary depending on the state of the network, or more specifically, how congested it is. When there are more transactions waiting to be processed than the network can handle, it will prioritize transactions based on the gas price that was set by the user, meaning that higher gas prices are more likely to get processed first. When the network is congested, gas prices increase to encourage more efficient use of the network's resources and decrease when network usage is lower. This dynamic pricing mechanism ensures that the Ethereum network remains accessible and functional for all users, while also incentivizing responsible and efficient use of the network's resources.

Why is gas necessary?

Turing Completeness

As we've learned, Ethereum is a Turing-complete platform, which means that any program that can be represented in code can theoretically be expressed and executed on the network. This opens up the door to countless different types of applications that can be built, but it also creates

the possibility that malicious or inefficient code can clog up the network, potentially leading to denial-of-service attacks, network spam, and other problems.

Preventing Infinite Loops

Gas to the rescue! To prevent accidental or intentional infinite loops in smart contract code, Ethereum requires that every transaction specify a gas limit. The gas limit establishes the maximum amount of gas that the transaction can consume, and they ensure that transactions are executed within a predetermined amount of computational resources, preventing the execution of code that might consume too much computation power and potentially cause the network to freeze or crash. Without gas, Ethereum's Turing completeness would be insecure and inefficient.

Autonomous Execution

It's also important to note that gas enables the execution of smart contracts without the need for a central authority to monitor their execution. The gas system provides a mechanism for regulating the resources required to execute the code of these contracts as well. In other words, without gas, it would be difficult to guarantee that smart contracts could operate autonomously, fairly and efficiently.

Concepts related to Ethereum Gas

Ethereum Denominations

Before diving into the inner workings of gas, it's important to understand a few of the most common denominations used in Ethereum.

Ether (ETH)

Ether is the native cryptocurrency of the Ethereum network. Gas fees are paid in ETH.

Wei

Wei is the smallest denomination of Ethereum and is equivalent to 10^-18 ETH. It is used to represent very small amounts of ETH, usually gas prices and transaction fees. To put 10^-18 into perspective:

1 ETH = 1,000,000,000,000,000,000 wei

1 wei = 0.000000000000000001 ETH

Gwei

Gwei is commonly used to express the price of gas. One gwei is equivalent to one billionth of one ETH or 10^-9 ETH.

1 ETH = 1,000,000,000 gwei

1 gwei = 0.000000001 ETH

Gas Price

Gas price on the network is denominated in gwei, and the gas fee is calculated as the product of the gas price and the amount of gas required for an operation. For example, if the gas price

is 50 gwei, and an operation requires 100,000 units of gas, the gas fee would be 0.005 ETH (50 gwei x 100,000 gas = 0.005 ETH).

Gas Limit

Gas limit is an essential component of the gas system in Ethereum. It defines the maximum amount of gas a user is willing to spend for a transaction to be processed. This gas limit is set by the sender of the transaction and represents the upper limit of computational resources that the transaction can consume. The Ethereum Virtual Machine (EVM) starts deducting the amount of gas used from the gas limit as soon as it starts processing the transaction.

Gas Estimation

Gas estimation is another key concept to understand. It refers to the process of predicting the amount of gas that will be required to execute a transaction. This is important because as we've seen in our example, the gas limit of a transaction needs to be set before it can be broadcasted to the network. If the gas limit is set too low, the transaction may fail to execute, while if it is set too high, the sender may end up paying more in transaction fees than is necessary.

There are several methods that can be used for gas estimation. One common method is to use historical gas prices and gas limits as a reference point, and to estimate the gas needed for a new transaction based on the gas used in similar past transactions. Another method is to simulate the execution of the transaction in a test environment to determine the actual amount of gas that would be used.

Gratefully, most Ethereum wallet applications have built-in gas estimation algorithms that can automatically calculate an appropriate gas limit for a transaction based on the network conditions at the time the transaction is initiated. This helps to prevent a transaction from failing from the gas limit being too low while optimizing for the best possible cost for the sender.

How Ethereum Gas Works

Ethereum underwent an upgrade in August 2021 known as the London Upgrade, which altered the way that ETH gas fees are calculated.

Pre London Upgrade

Before the London Upgrade, ETH gas worked like this:

- Assume Alice wants to pay Bob 1 ETH. The gas limit is 21,000 units, while the gas price is 200 gwei.
- The total fee is calculated as: (gas units (limit) x gas price per unit). In this example, that would equal: 21,000 x 200 = 4,200,000 gwei, or 0.0042 ETH.
- When Alice sends the ETH, 1.0042 ETH comes from her Ethereum wallet. Bob receives 1.0000 ETH. An Ethereum miner receives 0.0042 ETH.

Post London Upgrade

The London Upgrade was introduced in an effort to make Ethereum's fees more predictable for users. It also introduced a burn mechanism into Ethereum, to offset issuance of new ETH (there is no limit to how much ETH can be minted).

As of this upgrade, each block has a base fee, which is calculated by the network based on current demand for block space. This base fee gets burned (destroyed), so users are now expected to include a tip or priority fee with each ETH transaction — the greater the tip, the hope is, the more the transaction will gain priority.

This tip provides compensation to miners; many expect that most crypto wallets will integrate a feature that sets the tip fee automatically.

After the London Upgrade, gas works like this:

- Assume Alice wants to send Bob 1 ETH. The gas limit is 21,000 units, the base fee is 100 gwei, and Alice includes a tip of 10 gwei.
- The new formula is: gas units (limit) x (base fee + tip). This can be calculated as 21,000 x (100 + 10) = 2,310,000 gwei, or 0.00231 ETH.
- When Alice sends the ETH, 1.00231 ETH will be subtracted from her wallet. Bob will receive 1.0000 ETH. A miner will receive the tip of 0.00021 ETH. And 0.0021 ETH will be burned.

Alice also has the ability to set a maximum fee for the transaction. The difference between the max fee and actual fee will be refunded. This allows users to set a maximum amount to pay for transactions without having to worry about overpaying.

This makes things more predictable, as under the old transaction fee model, fees could wind up being higher than anticipated during times of extreme network congestion.

How do Ethereum Gas fees relate to transactions?

The way Ethereum gas fees relate to transactions is pretty simple: Each transaction requires a fee to be paid to miners as an incentive for processing the transaction. The general concept is not unlike that of other cryptocurrencies.

The only difference with ETH gas is that because the Ethereum Virtual Machine (EVM) is also a state machine, additional fees are required for more complex transactions, such as those involving smart contracts.

What Is the Ethereum Gas Limit?

The standard limit on an Ethereum gas fee is 21,000 units. The ether gas limit refers to the maximum amount of gas a user can consume to conduct a transaction.

Transactions involving smart contracts are more complicated, and require more computational power to execute. So these transactions need a higher gas limit than simpler transactions like sending payments.

Setting a gas limit too high is fine — the EVM will refund what doesn't get used. But setting a gas limit too low could result in a user losing some ETH and having their transaction declined.

If a user were to place an Ether gas limit of 50,000 for an ETH transfer, for example, the EVM would consume 21,000 and refund the remaining 29,000. But if someone were to set a gas limit of 20,000, and the transaction were to require 21,000 units, the EVM could consume 20,000 gas units as it tries to fulfill the transaction, but the transaction won't complete.

In this case, the user would hold on to the ETH they tried to send, but their 20,000 gas units would be lost because the EVM consumed it trying to complete the failed transaction.

What is the Benefit of a Gas Fee?

The benefit of an ETH gas fee post London Upgrade is that users can better anticipate what their total transaction cost will be. They can also send higher tips to miners to prioritize their transactions. This can be useful when someone wants to send money right away and doesn't want to wait too long for the transaction to confirm.

Another benefit of an adequate ETH gas fee is that it ensures a transaction will be accepted by the network. A too-low fee can result in a transaction being rejected, in which case a user could lose the gas they spent and not have their transaction go through.

Summing up

Gas is a vital component of Ethereum. It's what regulates the execution of all transactions and smart contracts, and it plays a significant role in the proper functioning and security of the network. Without gas, Ethereum's Turing-complete architecture would be inefficient and vulnerable to attacks. Gas also ensures that smart contracts can operate autonomously, fairly, and efficiently without the need for a central authority to monitor their execution. Understanding how gas works is essential for anyone who wants to develop applications or smart contracts on the Ethereum network.

Reference:

https://blog.defichain.com/what-is-turing-complete/

https://www.scribd.com/document/661022628/Turing-Machine-in-Blockchain

https://docs.base.org/base-camp/docs/introduction-to-ethereum/gas-use-in-eth-transactions/