Lecture Notes

Blockchain Features

Session 1 – Basic Idea Of Blockchain

In this session you have learnt the following topics:

- Types of blockchains
- Immutability
- Blockchain as a future promise

Types of Blockchains

Blockchain is mainly classified into the following types:

- Public Permisionless: A **public permissionless blockchain** is free for anyone to join or leave. Bitcoin blockchain is the best example of a public permissionless network. This type of network provides **anonymity**, **immutability** and **transparency** but compromises on efficiency.Public Permissioned: A **public permissioned blockchain** is an intermediate between private and public networks. It values **efficiency** and **immutability** over transparency and anonymity, where every participating member is aware of the identities of the other members in the network.
- A **private blockchain** is one which is operated and managed by a single entity. These type of blockchains are generally applicable in case of a conglomerate where the parent company runs the network for the underlying group of companies. In such a situation, they value efficiency over anonymity, transparency and immutability.
- A **consortium blockchain** is largely similar to a private blockchain but differs when you consider who controls or manages the network. Instead of concentrating all power in one entity, authority is distributed across two or more participants.

The following table summarises the key differences between the above-mentioned types of blockchains:

Type	Anonymity	Transparency	Immutability	Efficiency	Confidentiality	Throughput	Finality Turn aroud time (TAT)
Public	Yes	Yes	Yes	No	Low	Low	High
Permissioned	No	No	Yes	Yes	Medium	Low/Medium	Medium/High
Private	No	No	No	Yes	Very high	High	Very Low
Consortium	No	Partial	Yes	Yes	High	High	Low

Immutability in blockchain

To ensure immutability of data in databases, organisations employ various mechanisms ranging from access controls, multiple approvers, reconciliation, audit trails and firewalls. In spite of employing these means, data stored in the database are mutable (i.e not immutable) and can be altered once someone gains access to the data. This is where blockchain technology makes the biggest impact.

As you have learnt earlier, a block header hash is calculated as a double SHA 256 hash of all the block constituents.
Block header hash = SHA256 (SHA256 (previous block hash + Merkle root + timestamp + difficulty target + nonce)).

Blockchain Features-Session 2

Smart Contracts and Architecture of Blockchain

Introduction

The main topics you learnt as part of this session are:

- Smart Contracts
- Components required for deploying a Smart Contract on a blockchain
- Blockchain Architecture vis-à-vis Traditional Architecture

Smart Contract

A **contract** is a legally binding agreement between two entities. For a contract to be valid, the following conditions should be met:
(1) offer between two or more parties
(2) acceptance of the offer
(3) consideration, which is a benefit which must be bargained for between the parties
(4) mutuality of obligation
(5) competency and capacity
(6) a written instrument.

Most contracts are **valid** even when only two conditions are met amongst the 6 mentioned above: one, acceptance and two, consideration.

Further, contracts are of two types:
(1) **Written**: Documented contracts
(2) **Verbal**: Contracts maintained out of goodwill.

A **smart contract** is the **digital version of a contract** that runs on the Blockchain nodes and is executed when a set of **trigger criteria** are met. If designed right, a smart contract will surely be successful as it is driven by a computer system **without human intervention**.

Designing a smart contract is based on the following steps:

Step 1: Understanding the business problem statement
Step 2: Defining the blockchain network and its participants
Step 3: Assessing involved parties of the contract (whether all members of the network or specific roles)
Step 4: Exploring pre-requisites for the Smart Contract
Step 5: Examining trigger event for the Smart Contract (once this condition is met, the contract is executed)
Step 6: Evaluating business logic for the Smart Contract (If the condition is fulfilled, then what course of action will be followed, else what will happen?)

Evolution of Smart Contracts

The Smart contract has evolved through the following phases:

- **State Machine:** The very first version of blockchain was a **state machine**. It was just like an accounting ledger, where it held only the **state** of the node.
- **Smart Contracts:** All the nodes within a computer network have a code associated with these nodes, which can run on its own based on **predefined logic** known as smart contracts.
- **Oracles:** Oracles are external agents which connect the smart contract to the external world. Any external data that is required for the smart contract will be fetched by agents called **oracles.**

Why Evolution of Smart Contract

Evolution of smart contracts happened because:

1. Enterprises wanted to eradicate the need for the smart contract to be run on all nodes separately
2. Enterprises wanted interoperability within or outside their organizations as a feature for their networks without compromising on the confidentiality of their inter-operations.

The only connection between the network and the outside world was **oracles**, and sometimes, the oracles could get corrupted or hacked.

To avoid the sacrifice of confidentiality, a **separate layer to hold the business logic of smart contracts was envisioned** because of the demand from enterprises and was called **cryptlets**.

Like APIs in a three-layer network architecture, there are **cryptlets fabric** which serve the same purpose in a computer network. Cryptlets are of two types:

1. Utility Cryptlets
2. Contract Cryptlets

There are several developments around IDEs and UI tools for development of smart contracts and these developments promise that blockchains are set to evolve to a better state as we go ahead.

Blockchain Architecture

The **web** is a service that allows sharing of data. Due to this feature of the web, it is said to follow the **client-server architecture**. In a client-server architecture, a client can be a machine or a program that allows the user to make requests to the server. For example, WWW is a client-server program. Users can request data and the web will provide it. On the other hand, a server is essentially a program and **NOT A DEVICE**.

There are two types of servers:

1. Web servers like Apache, which serve HTTP requests
2. Database servers like MySQL, which runs database management system queries

A server is always listening for requests. As soon as it receives a request, it responds with an action or a message.
A Client-Server architecture model is an architecture on the web that splits the devices into two sections: devices that request for data and devices that provide the requested data.
In a peer-to-peer network, each device can act as a server and as a client at different points in time. The most suitable example of this is 'Skype'.

2-tier Architecture	3-tier architecture
A 2-tier architecture is basically meant for client-server applications. This means that there is direct communication between client and server.	A 3-tier architecture is usually meant for web-based applications. There is always an intermediary (can be more than one) between server and client.
Servers can not record multiple requests at the same time.	These architectures have a high performance because requests can be cached, and load can be reduced.
It is easy to maintain.	Complexity is increased due to the increase in the number of layers.
It is easy to modify.	It improves data integrity, security and are easy to maintain and modify.

A distributed app, commonly referred to as the D-App, runs on a peer-to-peer network. Assume multiple copies of client-server devices set up on the network, with each server connecting and interacting with other servers. Each of these servers can be mapped to be a node in a blockchain network. D-Apps connect a blockchain via a smart contract, much like APIs in a client-server middleware architecture. The address or domain name for a server is fixed, and clients can connect through that.