



GladMeds - Medical Report

Basic Information:

Full Name:	Suryansh Rohil
Email:	sr738@snu.edu.in
Age:	19
Gender:	FEMALE
Blood Group:	AB+
Phone:	9355294184
Address:	Haryana
Registered On:	19/6/2025, 12:41:19 pm



Extended Medical Profile:

Date Of Birth: Fri Nov 11 2005 00:00:00 GMT+0530 (India Standard Time)

Marital Status: SINGLE

Id Proof Type: Aadhaar

Id Proof Number: 111111111111

Occupation: Student

Nationality: Indian

Guardian Name: Person 1

Guardian Relation: Parent

Guardian Phone: 9244718134

Religion: Hindu

Preferred Language: Hindi, English, Punjabi

Known Allergies: none

Chronic Conditions: none

Past Surgeries: intestine

Current Medications: none

Immunization Status: vaccinated

Family Medical History: none

Lifestyle Details: simple

Menstrual Info: sdasd

Javascript Task Sheet 1

Date: 21 May, 2025

✓ Task 1: Rebuild `map`, `filter`, `reduce` — Without Native Methods

What to Build:

Recreate JavaScript's `.map()`, `.filter()`, and `.reduce()` functions by writing your own versions from scratch. Do not use any native array methods.

```
myMap(array, callback)
myFilter(array, callback)
myReduce(array, callback, initialValue)
```

Constraints:

- ✗ No `.map`, `.filter`, `.reduce`, `.forEach`, or `.flatMap`
- ✓ Use only `for`, `while`, or recursion

Why This Matters:

These three functions are the foundation of functional programming in JS. You'll discover how callbacks are invoked, how accumulators work, and how iteration transforms data. This makes future React data rendering logic intuitive.

AI Trap: AI will give you working code, but won't help you **explain it, optimize it, or design multiple implementations with constraints**. You'll write 3 different styles (loop-based, recursion-based, and mutation-based) and compare them.

✓ Task 2: Fake Terminal CLI in the Browser

What to Build:

Simulate a terminal/command-line UI inside a web browser. Commands to support:

- `add task <text>`
- `list tasks`
- `clear tasks`
- `delete task <id>`

Display command input/output in a styled `<div>`, not `<input>`.

Why This Matters:

You'll learn to:

- Parse strings
- Build internal state (like a task manager)
- Dynamically create and update DOM like a terminal
This builds a strong mental model for interpreting inputs and building custom renderers (a key React skill).

AI Trap: AI might help generate a basic UI, but parsing commands, handling edge cases, and building a versioned internal state cannot be easily copy-pasted.

✅ Task 3: Tab System Without `classList` or `style.display`

What to Build:

Create a tab component with three tabs (e.g., Home, About, Contact). Switching tabs should:

- Hide other sections
- Show only selected content
But: You cannot use `.classList` or inline styles to do this.

Why This Matters:

You'll be forced to think creatively:

- How else can we show/hide elements?
- Can you reorder the DOM? Use attributes? Custom logic?
This builds problem-solving skills when libraries are unavailable or break.

AI Trap: AI will usually suggest `.classList.toggle()` or `style.display = 'none'`. You must deliberately bypass this with custom logic.

✅ Task 4: Manual ToDo App with Debug State Panel

What to Build:

A complete ToDo app with add, delete, toggle-complete functionality.

- A floating state panel that:
- Shows the current task array
- Logs every change (add/delete) with timestamp

Why This Matters:

This is how **Redux DevTools** works. You'll learn:

- Manual state handling
- DOM re-rendering
- Logging state changes for debugging

AI Trap: The task includes building a *debugger*, not just the app — AI won't know what you mean by this unless you do the thinking.

✓ Task 5: Stopwatch & Timer — With Artificial Lag Injection

What to Build:

- Stopwatch (start, stop, lap, reset)
- Countdown timer (user inputs time)
- Digital clock (real time)

Inject **delays or race conditions** by wrapping logic inside random timeouts:

```
setTimeout(() => updateTimer(), Math.random() * 1000)
```

Why This Matters:

Teaches timing issues, event loop, and controlling concurrent intervals — essential for React **useEffect**, Node's async behavior, and debugging bugs caused by unintentional async behavior.

AI Trap: You're deliberately making things **break** and fixing them — something AI isn't good at doing blindly.

✓ Task 6: Modal with Keyboard Accessibility & Focus Trap

What to Build:

Build a modal that:

- Auto-focuses the first input
- Disallows focus leaving the modal when opened
- Closes on:
 - Escape key
 - Clicking outside modal

Why This Matters:

Manual DOM control like this is core to accessibility and UI frameworks. This teaches you what `focus trap` libraries do under the hood.

AI Trap: AI won't implement a correct focus trap without you understanding `document.activeElement`, `tabindex`, and keyboard listeners.

✓ Task 7: Live Preview + Rule-Driven Validation Engine

What to Build:

Form fields:

- Name, email, password, bio
- Profile image upload
- Display live preview next to form

Also:

- Create a `validateField()` that takes rules from JSON like:

```
{ name: ['required'], password: ['min:8'] }
```

Why This Matters:

You'll build your own form engine — a crucial abstraction used in every enterprise app.

AI Trap: Copy-pasting won't teach you how to separate rules, evaluate them, and show proper UX.

✓ Task 8: Manual Hash-Based Router with History Stack

What to Build:

Single Page App with 3 pages (`#home`, `#about`, `#contact`) using hash routing. Maintain:

- A manual navigation stack
- Back/forward tracking
- Scroll position restoration per view

Why This Matters:

You'll understand how React Router or Next.js routing works under the hood, and why SPA routing is harder than it looks.

AI Trap: AI won't generate a full-stack-aware version of history + scroll + state unless you design it.

✓ Task 9: Component System with `createComponent()`

What to Build:

Create a function `createComponent(tag, props, children)`

Use this to build:

- Button
- Avatar
- Card
- Input

Then implement a system to:

- Re-render only when props change
- Compare old and new props

Why This Matters:

You're simulating React's **component + virtual DOM + diffing** model.

AI Trap: AI will generate a simple factory function — but reconciling changes, props diffing, and selective DOM updating must be reasoned through.

✅ Task 10: Sortable Table with Debug Panel

What to Build:

A data table:

- Sortable by age/name/score
- Filter by text
- A floating panel shows:
 - Current filters
 - Current sort direction
 - Visible row count

Why This Matters:

This is what **dashboard tables and admin panels** are built on. State reflection is core to debugging production apps.

AI Trap: AI can help sort, but won't tell you **why your table didn't re-render**, or why a filter broke the sort order.

✅ Task 11: Virtual DOM Diff Engine + Visual Logs

What to Build:

- Define 2 virtual DOM JSON objects
- Write `diff(oldTree, newTree)` to:
 - List additions, removals, updates
 - Apply patch to DOM
- Log:

- + for new nodes
- ~ for changed nodes
- - for removed nodes

Why This Matters:

This is how React performs DOM updates efficiently. You'll simulate reconciliation manually.

AI Trap: You can't solve this without **understanding tree comparison**, keying, and mutation.

✅ Task 12: Fetch with Retry + Backoff Strategy

What to Build:

Use `fetch()` to hit OpenWeatherMap API:

- On failure, retry after:
 - 1s → 2s → 4s → Stop after 5 tries
- Show loading, success, error states
- Add a retry button

Why This Matters:

Resilience, retries, and exponential backoff are real-world practices in all frontend/backend apps.

AI Trap: You'll have to *design a strategy*, not just write a `fetch()` function.

✅ Task 13: Drag & Drop Kanban Board — With Keyboard Support

What to Build:

- A **Kanban board** with 3 columns:
 - To Do | ● In Progress | ● Done
- Each task card should be:
 - Draggable using native `dragstart`, `dragover`, `drop` events

- Droppable into another column

Required Features:

- Manual state management (object with column arrays)
- Persist state in `localStorage`
- Support **keyboard movement**:
 - Focus on task → Use `←` `↑` `→` `↓` or **WASD** to move tasks between columns

Why This Matters:

- You'll learn complex **DOM event coordination**
- Simulates advanced UI used in apps like Trello, Jira
- Keyboard support teaches **accessibility**, important for real users and compliance

AI Trap:

AI can help with drag-and-drop basics, but:

- Won't handle **keyboard accessibility**
- Won't track full column logic
- Won't persist state properly unless you design it

✅ Task 14: Lazy-loaded Image Gallery with Cache Detection

What to Build:

- Grid of 20+ images (with categories like "Nature", "People", "Architecture")
- Filter images by tag (e.g., buttons or dropdown)
- Lazy-load images only when visible (IntersectionObserver or scroll logic)

- On click, show modal preview

Advanced Twist:

- Use `performance.getEntriesByType('resource')` to check if image was loaded from **cache** or **network**
- Log this in a debug panel (e.g., “13 images loaded from cache”)

Why This Matters:

- Teaches **performance awareness**, image loading strategies
- Introduces you to browser performance APIs
- Bridges frontend design with **browser internals**

AI Trap:

AI won't think of **cache detection**, will likely skip **IntersectionObserver**, and won't explain **why certain images load faster** unless you design the logic.

✅ Task 15: Notepad Lite – With Rich Text + Undo/Redo Stack

What to Build:

- A text editor (like a mini Notepad) in the browser
- Features:
 - Bold / Italic / Underline buttons
 - Word and character count
 - Download as `.txt` file

Advanced Twist:

- Implement **Undo/Redo functionality**
 - Use a stack-based system: every keystroke is a change

- Provide keyboard shortcuts (Ctrl+Z, Ctrl+Y)

Why This Matters:

- You'll learn to manage **complex internal state**
- Undo systems are core in collaborative apps like Google Docs, Figma

AI Trap:

AI may write a basic editor, but building **undo history**, applying operations in the right sequence, and syncing button states requires you to design a state machine.

✓ Task 16: Analog Clock + Timezone Switching UI

What to Build:

- A **real-time analog clock** using CSS + JS
- Render:
 - Hour, minute, and second hands that move every tick
- Add:
 - Dropdown with 5 timezones (UTC offsets)
 - On change, update the clock to reflect the new timezone

Why This Matters:

- Reinforces **DOM transforms**, time math, and `setInterval`
- Teaches how timezones affect date/time calculations
- Foundation for **server/client rendering consistency**

AI Trap:

AI may get the clock running, but timezone conversions require **manual Date math**, and adjusting for local offset is a reasoning-heavy task.

✓ Task 17: Custom Carousel with Infinite Loop + Smooth Animations

What to Build:

- A responsive carousel with:
 - 5 visible images (or slides)
 - Next/Prev buttons
 - Dot indicators
 - Auto-play every 3 seconds
- Implement:
 - Smooth scroll transitions
 - Infinite loop behavior (rewind to start seamlessly)

Advanced Twist:

- Inject 1000 slides/images
- Use **virtualized rendering** (only render 3–5 visible slides)

Why This Matters:

- Reinforces **scroll, animation, and state control**
- Virtualization is key for building **high-performance apps** (e.g., tables, image viewers)

AI Trap:

Basic carousels are AI-friendly, but **loop logic, performance optimization, and virtualization design** are too contextual for AI to do well.

✓ Task 18: Toast Notification System with Priority Queue

What to Build:

- Create your own `toast()` notification system (like in Slack, Gmail)

Call API like:

```
toast.success('Message sent')
toast.error('Network error')
```

-

System Rules:

- Max 3 visible at a time
- Auto-dismiss after 5s
- Stack top to bottom
- Queue up remaining toasts
- Prioritize by type:
 - error > warning > success > info

Why This Matters:

- You'll simulate a **priority queue + lifecycle system**
- This builds engineering muscle needed for stateful systems like **job queues, alerts, or event logs**

AI Trap:

AI will likely hard-code display logic. **Queuing and prioritization**, combined with **dismiss logic**, requires real architectural design.

✅ Task 19: JSON-Based Dynamic Form Engine + Export

What to Build:

Given a config like:

```
[
  { type: 'text', label: 'Email', required: true },
```

```
[
  { type: 'password', label: 'Password', rules: ['min:8'] },
  { type: 'select', label: 'Gender', options: ['M', 'F', 'Other'] }
]
```

- Render a dynamic form based on the config
- Handle:
 - Validations from `rules`
 - Error messages
 - Labeling
- On submit → Return JSON object of input values

Export Feature:

- Export this form engine as a function: `generateForm(config, containerId)`

Why This Matters:

You'll build a **form engine**, not just a form. This sets you up to build **low-code tools**, **CMS**, and **admin panels** in the real world.

AI Trap:

AI will render a static form — won't help you write a reusable engine unless you plan the API, handle configs, and abstract input logic.



✅ Task 20: Redux-Like State Store with Time Travel Debugger

What to Build:

Create a Redux-inspired store:

```
const store = createStore(reducer, initialState)
store.subscribe(render)
store.dispatch({ type: 'INCREMENT' })
```

Features:

- `getState`, `dispatch`, `subscribe`
- Actions and reducer pattern
- Add:
 - Visual log of past states
 - Time-travel buttons: Prev  | Next 

Why This Matters:

You'll understand:

- What Redux does
- How `useReducer` and `useContext` are powered
- Debugging via **immutable state chains**

AI Trap:

AI might mimic Redux API, but you must:

- Design reducer logic
 - Track full state history
 - Write visual debugger and hook it into DOM
 - Handle edge cases like jumping beyond array bounds
-

Hello, world!

An Efficient Design and Implementation of Blockchain Architecture

Midsem report for the OUR project

Anish Gupta(Student-2nd year)
Btech-Computer Science and Engineering
SoE, Shiv Nadar University
2310110047
ag801@snu.edu.in

Dr. Sweta Kumari(Faculty Advisor)
Assistant professor
Dept-Computer Science and Engineering
Shiv Nadar University
sweta.kumari@snu.edu.in

Abstract—Blockchain technology has revolutionized how we manage secure, decentralized transactions across various industries, from finance to supply chain management. However, traditional blockchains, which use basic structures like linked lists(used in bitcoin [12]), merkle trees(used in hyperledger [11]), face issues with transaction speed, scalability, and energy efficiency. Directed Acyclic Graph-based blockchain marks a shift from traditional blockchains, enabling significantly higher throughput through concurrent storage and execution. We have explored DAG structures like Tangle which is used in IOTA, MorphDAG, LightDAG etc.

I. INTRODUCTION

Blockchain technology is widely used for secure and transparent digital transactions. its decentralized structure ensures trust without the need for intermediaries. However, many existing blockchain systems face issues like slow processing, high energy use, and limited scalability.

This OUR project plans to present a more efficient blockchain design to address these challenges by improving data storage, optimizing transaction processing, and using better security methods; our approach enhances performance while keeping the system secure. Our primary focus will be on speed, however. We also plan to compare our design with existing blockchain models to show its advantages in future.

As from our literature review we found out to work on DAG based structures, we are currently exploring the MorphDAG technology and understanding its code, algorithm and the Maths behind it in Depth before we apply our methodology.

II. LITERATURE REVIEW

Here are a few things I learnt from online resources and research papers available in this field (briefing some of them) to explore more about this topic :-

A. Basic Blockchain terminologies and concepts

Before diving into the world of Blockchain I had the opportunity to learn about basic blockchain working, terminologies and what are all the types of blockchain available in the market.

B. Research Paper-Implementation of directed acyclic graph in blockchain network to improve security and speed of transactions

This paper explores how DAGs type structures improve blockchain networks by enhancing transaction speed and security. Unlike traditional blockchains, they allow parallel transaction processing, reducing delays and fees. This structure eliminates mining, making transactions faster, scalable, and fee-less, ideal for IoT and financial applications.

C. Research Paper-A Comparative Analysis of DAG-based Blockchain Architectures

Since we knew how DAG is very promising from previous paper, this one analyzes DAG-based blockchains like IOTA, Nano, and Byteball, highlighting their scalability, speed, and efficiency over traditional blockchains. It compares their consensus mechanisms, transaction validation, and security to identify best practices and proposes an optimized hybrid DAG model for improved performance.

D. The Merkle Tree structure

A Merkle tree is a data structure used in blockchains to organize and verify transaction data efficiently. it follows a binary tree format, where each leaf node contains a cryptographic hash of a transaction, and each non-leaf node stores the hash of its two child nodes. this process continues recursively until a single hash, known as the Merkle root, is formed at the top. The Merkle root is stored in the block header, allowing efficient verification of transaction integrity without requiring the entire dataset. bitcoin and ethereum use this structure to secure transactions.

E. Why DAG, and not Merkle tree

- **Higher scalability** – DAG enable parallel transaction processing, whereas merkle trees follow a sequential validation approach.
- **Greater throughput**– DAG efficiently process high transaction volumes, while merkle trees may create bottlenecks in large networks.

TABLE I
DIFFERENT DAG TECHNOLOGIES

Comparing Factor	DAG Technologies		
	<i>MorphDAG</i>	<i>Tangle</i>	<i>JointGraph</i>
Scalability	High (Dynamically adjusts DAG structure)	High, but may suffer from orphaned transactions	Moderate (designed for consortium blockchains)
Transaction Speed	Very High (adaptive transaction processing)	Fast, but can slow down in low-participation scenarios	High (but limited by supervisory node bottlenecks)
Storage Efficiency	Elastic storage reduces redundant data	Prone to unbounded DAG growth	Snapshots reduce memory load
Latency	Lowest (adaptive consensus process)	Low, but varies based on network activity	Low (due to direct voting mechanism)
Security Model	Stronger (adaptive conflict resolution)	Secure, but susceptible to network attacks	Strong (malicious nodes removed)
Ideal Use Case	Large-scale decentralized applications	IoT and microtransactions	Consortium blockchains

- **Faster confirmations** – Transactions in a DAG validate each other, reducing delays compared to merkle tree-based proof-of-work systems.
- **No reliance on mining** – Some DAG-based networks remove the need for mining, making them more energy-efficient.

F. Research Paper-Jointgraph: A DAG-based efficient consensus algorithm for consortium blockchains

This paper helped us to understand and learn how combining consensus with DAG structure can be a optimal solution as unlike traditional blockchains, JointGraph allows parallel transaction processing, reducing confirmation delays and improving efficiency. It addresses Byzantine faults by isolating malicious nodes and optimizes memory through periodic snapshots.

G. Research paper- Tangle the Blockchain:Towards Connecting Blockchain and DAG

Tangle is a hybrid Blockchain-DAG integration to improve scalability and efficiency, particularly for IoT. It introduced a connector component that facilitates seamless interaction, using Blockchain for data storage and Tangle for fast, scalable transactions. This system enhances flexibility, reliability in decentralized networks.

H. Research paper- MorphDAG: A Workload-Aware Elastic DAG-Based Blockchain

This is the technology I am currently exploring. I am briefly explaining it here:-

- MorphDAG is a workload-aware DAG-based blockchain designed to improve scalability and storage efficiency.
- It dynamically adjusts its DAG structure based on transaction patterns, optimizing performance for hotspot and cold transactions.
- By implementing elastic storage techniques, MorphDAG reduces memory usage while maintaining fast consensus.
- The system is tested under real-world workloads, demonstrating high throughput, low latency, and adaptive resource allocation, making it well-suited for large-scale decentralized applications.

III. PROPOSED METHODOLOGY/FUTURE WORK

We plan to work on the structural code of MorphDAG and propose a new blockchain structure which would have some additional features which could potentially increase throughput and transaction speed as MorphDAG as workload aware, the new structure could give very promising results. We plan to :-

- To work on the code so that it can **allow cyclic dependencies** in isolated segments of the DAG for temporary local consensus which could increase speed
- Divide the DAG into priority zones like **High-priority zones** process time-sensitive transactions (e.g., real-time payments) and **Low-priority zones** handle bulk, non-urgent transactions (e.g., file storage).
- Allow nodes to create additional dynamic edges between non-parent events based on: **Similarity** (e.g., same sender, timestamp proximity), **Dependencies** (e.g., transactions involving the same assets or participants).
- Optimize the DAG structure for energy-efficient validation so, Nodes calculate the energy cost of validating specific transactions and **prioritize low-cost paths**. Transactions with higher costs are deferred or aggregated for batch processing.
- Also plan on checking that which blockchain **consensus algorithm** can result in highest throughput not compromising the basic security of the system
- I am attaching the MorphDAG code link here for reference [<https://github.com/CGCL-codes/MorphDAG/blob/main/README.md>]

Some existing plans might be modified/ more plans be added depending on our understanding of this code

IV. CONCLUSION

The DAG blockchain structure shows a lot of promise and once we have completed the code modification and testing under real-life workloads like Ethereum we would be able to present a blockchain structure which would result in a higher throughput and wont compromise safety too. For now we continue to work on the code of MorphDAG which is a latest technology we found and has a lot of caliber.

ACKNOWLEDGMENT

I Would like to thank Dr Sweta Kumari (my faculty Advisor) for letting me work on this project, I am learning a lot from this opportunity and hope to continue doing so, I would also like to thank the OUR team for giving me the wonderful opportunity to take up this research in my second year.

REFERENCES

- [1] P. S. Anjana, S. Kumari, S. Peri, S. Rathor and A. Somani, "An Efficient Framework for Optimistic Concurrent Execution of Smart Contracts," 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Pavia, Italy, 2019, pp. 83-92
- [2] Fu, Xiang et al. "Jointgraph: A DAG-based efficient consensus algorithm for consortium blockchains." Software: Practice and Experience 51 (2019): 1987 - 1999.

- [3] H. Hellani, L. Sliman, A. E. Samhat and E. Exposito, "Tangle the Blockchain: Towards Connecting Blockchain and DAG," 2021 IEEE 30th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Bayonne, France, 2021, pp. 63-68
- [4] H. Pervez, M. Muneeb, M. U. Irfan and I. U. Haq, "A Comparative Analysis of DAG-Based Blockchain Architectures," 2018 12th International Conference on Open Source Systems and Technologies (ICOSST), Lahore, Pakistan, 2018, pp. 27-34
- [5] F. M. Benčić and I. Podnar Žarko, "Distributed Ledger Technology: Blockchain Compared to Directed Acyclic Graph," 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, 2018, pp. 1569-1570
- [6] Kotilevets, I. D. et al. "Implementation of directed acyclic graph in blockchain network to improve security and speed of transactions." IFAC-PapersOnLine 51 (2018): 693-696.
- [7] Tokhmetov, A., Lee, V. ., and Tanchenko, L. . (2023). DEVELOPMENT OF DAG BLOCKCHAIN MODEL. Scientific Journal of Astana IT University, 16(16).
- [8] Qin Wang, Jiangshan Yu, Shiping Chen, and Yang Xiang. 2023. SoK: DAG-based Blockchain Systems. ACM Comput. Surv. 55, 12, Article 261 (December 2023), 38 pages.
- [9] Dai, Xiaohai and Wang, Guanxiong and Xiao, Jiang and Guo, Zhengxuan and Hao, Rui and Xie, Xia and Jin, Hai. (2024). LightDAG: A Low-latency DAG-based BFT Consensus through Lightweight Broadcast. 998-1008. 10.1109/IPDPS57955.2024.00093.
- [10] S. Zhang et al., "MorphDAG: A Workload-Aware Elastic DAG-Based Blockchain," in IEEE Transactions on Knowledge and Data Engineering, vol. 36, no. 10, pp. 5249-5264, Oct. 2024
- [11] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In Proceedings of the Thirteenth EuroSys Conference (EuroSys '18). Association for Computing Machinery, New York, NY, USA, Article 30, 1–15. <https://doi.org/10.1145/3190508.3190538>
- [12] (2009). Bitcoin: A Peer-to-Peer Electronic Cash System.
- [13] Wang, Tianyu, et al. "Understanding characteristics and system implications of DAG-based blockchain in IoT environments." IEEE Internet of Things Journal 9.16 (2021): 14478-14489.
- [14] Park, Seongjoon, Seounghwan Oh, and Hwangnam Kim. "Performance analysis of DAG-based cryptocurrency." In 2019 IEEE International Conference on Communications workshops (ICC workshops), pp. 1-6. IEEE, 2019.
- [15] Yang, Wenhui, Xiaohai Dai, Jiang Xiao, and Hai Jin. "LDV: A lightweight DAG-based blockchain for vehicular social networks." IEEE Transactions on Vehicular Technology 69, no. 6 (2020): 5749-5759.