MONEYLAB®

امکان‌سنجی ادغام ماشین مجازی اتریوم در شبکه خصوصی با معماری استلار

به سفارش «دایاچین»
شرکت تدبیراندیشان نوین افروز

ویرایش نخست - ۱۶ اردی‌بهشت ۱۴۰۴
میرسهیل نیک‌زاد کلورزی

فهرست عناوین

مقدمه

در این گزارش، قابلیت استفاده از ماشین مجازی اتریوم (EVM) به عنوان یک کلاینت یا «مرورگر» جایگزین برای تعامل با شبکه بلاک‌چین استلار (یک شبکه خصوصی کلون‌شده از استلار، موسوم به شبکه X) مورد بررسی جامع قرار می‌گیرد. هدف این است که یکی از نهادهای انکر (Anchor) در شبکه، به جای استفاده از نرم‌افزار معمول استلار، از اکوسیستم توسعه و تجربه کاربری برتر EVM بهره‌مند شود بدون نیاز به تغییر یا آگاهی سایر اعضای شبکه. این رویکرد امکان برنامه‌نویسی پیشرفته‌تر و استفاده از ابزارهای بالغ اتریوم را برای انکر فراهم می‌کند، در حالی که شبکه زیربنایی همچنان با پروتکل استلار (SCP) به کار خود ادامه می‌دهد. در ادامه، موارد خواسته‌شده شامل مقایسه معماری، روش‌های اتصال EVM به استلار، امکان‌سنجی اجرای موازی، ابزارهای موجود، نمونه‌های مشابه، معماری پیشنهادی، زمان‌بندی توسعه و مزایا و معایب ارائه می‌شود.

1. مقایسه معماری استلار و اتریوم/EVM

هسته و اجزای شبکه: شبکه استلار و اتریوم هر دو یک مدل مبتنی بر حساب (Account-based) دارند، اما طراحی آن‌ها تفاوت‌های بنیادینی دارد. استلار به طور ویژه برای تسهیل خدمات مالی و انتقال دارایی طراحی شده و دارای امکانات داخلی (پرایمیتیوهای) از پیش ساخته برای مدیریت دارایی است. از این رو در شبکه استلار موجودیت‌های خاصی مانند صدور دارایی‌های سفارشی، حساب‌های امانی (Claimable Balances)، دفتر سفارشات غیرمتمرکز (DEX) و استخرهای نقدینگی (AMM) به صورت توکار پشتیبانی می‌شوند. این ویژگی‌ها استلار را در حوزه پرداخت و تبادل ارزش بسیار کارآمد ساخته اما انعطاف‌پذیری آن را (حداقل تا قبل از سال 2024) در اجرای قراردادهای پیچیده محدود کرده بود. در مقابل، اتریوم (و به طور کلی زنجیره‌های مبتنی بر EVM) برای ایجاد برنامه‌های غیرمتمرکز و قراردادهای هوشمند تورینگ کامل طراحی شده‌اند و انعطاف‌پذیری بالایی دارند. این انعطاف به قیمت کاهش سرعت و کارایی تمام شده است؛ اتریوم به منظور حفظ عملکرد، نیازمند راهکارهای لایه دوم برای مقیاس‌پذیری بود و کارمزدها نیز بسته به پیچیدگی محاسباتی قرارداد افزایش می‌یابد. مکانیزم اجماع: استلار از پروتکل اجماع استلار یا SCP (Stellar Consensus Protocol) استفاده می‌کند که گونه‌ای توافق فدراتیو بیزانس (FBA) است. این روش را می‌توان «Proof of Agreement» نامید که در آن گره‌های شبکه بر اساس مجموعه‌های اعتماد متقابل (quorum slices) به اتفاق نظر می‌رسند. نتیجه این است که استلار هر ۵ تا ۶ ثانیه یک بلاک جدید (در استلار اصطلاحاً «ledger» یا دفتر کل) می‌بندد و نهایی‌سازی تراکنش در همین زمان کوتاه به صورت قطعی (deterministic finality) انجام می‌شود. در سوی مقابل، اتریوم در آغاز از اثبات کار (PoW) استفاده می‌کرد و اکنون به اثبات سهام (PoS) مهاجرت کرده است. مکانیزم PoS اتریوم (پس از به‌روزرسانی اتریوم ۲٫۰) منجر به نهایی‌سازی احتمالاتی (probabilistic finality) می‌شود که برای قطعیت بالای زنجیره به تقریباً ۲ دوره (epoch) در حدود ۱۳ دقیقه زمان نیاز دارد. در مجموع، اجماع استلار سریع‌تر و با قطعیت آنی است، در حالی که اجماع اتریوم (PoS فعلی) کمی پیچیده‌تر و زمان‌برتر برای نهایی شدن کامل است. ساختار داده‌ها و مدل حساب: همانطور که گفته شد، هر دو شبکه مبتنی بر حساب هستند اما انواع داده‌ی نگهداری‌شده در دفتر کل آن‌ها متفاوت است. در اتریوم، هر حساب ممکن است یک قرارداد هوشمند با کد و فضای ذخیره‌سازی داشته باشد یا یک حساب معمولی با موجودی. در استلار، علاوه بر حساب‌های اصلی و موجودی لومن (XLM)، انواع Entryهای دیگری در حالت شبکه ثبت می‌شوند: از جمله Trustline (مانده‌حساب توکن‌های منتشرشده توسط انکرها)، Offers برای سفارشات بازار داخلی، استخرهای نقدینگی، بالانس‌های قابل مطالبه برای مکانیزم‌های امانی، و در نسخه‌های جدیدتر قراردادها و داده قرارداد مربوط به پلتفرم هوشمند Soroban. این تنوع داده‌ها باعث پیچیدگی بیشتر مدل استلار نسبت به مدل ساده‌تر اتریوم می‌شود. به بیان ساده، بسیاری از قابلیت‌هایی که در اتریوم باید از طریق قراردادهای هوشمند پیاده‌سازی شوند (مثل صدور توکن، صرافی غیرمتمرکز، و...)، در استلار به صورت ویژگی‌های از پیش‌تعریف‌شده‌ی پروتکل قابل استفاده‌اند. این موضوع یکی از دلایل امنیت و کارایی بالاتر استلار در کاربری‌های مالی پایه است، اما از سوی دیگر اتریوم را از نظر انعطاف برنامه‌نویسی متمایز می‌کند. تعامل با نودها (کلاینت‌های شبکه): در شبکه اتریوم، تعامل برنامه‌ها و کاربران با شبکه عمدتاً از طریق رابط JSON-RPC انجام می‌شود که توسط کلاینت‌های اتریوم (مثل Geth، Besu و ...) فراهم می‌گردد. این رابط استاندارد، امکان ارسال تراکنش (مثلاً eth_sendTransaction) و خواندن اطلاعات (مثلاً eth_getBalance, eth_call) را به صورت برنامه‌نویسی مهیا می‌کند. در نتیجه، ابزارهای متنوعی مانند Web3.js و Ethers.js برای اتصال به گره اتریوم و اجرای دستورات RPC وجود دارند. در مقابل، معماری سنتی استلار از مؤلفه‌ای جداگانه به نام Horizon برای تعامل با شبکه استفاده می‌کند. Horizon یک API وب RESTful است که بین

برنامه‌های کلاینت و شبکه استلار قرار می‌گیرد و امکان ارسال تراکنش‌ها و بازیابی داده‌های تاریخی شبکه را می‌دهد. به عبارتی، Horizon نقش واسطه سمت کاربر را دارد و از طریق پروتکل HTTP/REST، توسعه‌دهندگان را قادر می‌سازد بدون اجرای مستقیم Stellar Core، با شبکه تعامل کنند. افزون بر Horizon، استلار در معماری جدید خود (همزمان با افزوده‌شدن Soroban) یک سرویس RPC JSON معرفی کرده است. این سرویس RPC (مشابه JSON-RPCهای متداول در بلاک‌چین‌ها) امکان تعامل با قراردادهای هوشمند و عملیات زنجیره را به صورت فراخوانی‌های از راه دور فراهم می‌کند. طبق مستندات رسمی، Stellar RPC در حقیقت یک سرور JSON-RPC است که درخواست‌های مربوط به قراردادهای هوشمند را مستقیماً دریافت و به معادل پروتکلی آن در شبکه استلار ترجمه می‌کند. این تحول نشان می‌دهد استلار نیز برای پشتیبانی بهتر از قراردادهای هوشمند و توسعه‌دهندگان، به سمت ارائه APIهای مشابه اتریوم (RPC) حرکت کرده است. به طور خلاصه، تفاوت در تعامل با نودها چنین است: در اتریوم هر گره کامل یک رابط RPC داخلی دارد، ولی در استلار معمولاً از یک لایه جدا (Horizon یا RPC سرور) برای ارتباط با گره‌های هسته (Stellar Core) استفاده می‌شود. جمع‌بندی معماری: با در نظر گرفتن موارد فوق، شبکه استلار سبک‌تر، سریع‌تر و دارای ساختار ماژولار (Core و Horizon) است و بسیاری از امکانات مالی را بدون نیاز به قرارداد هوشمند فراهم می‌کند. اتریوم سنگین‌تر، کندتر (از نظر TPS و finality) اما به شدت قابل برنامه‌ریزی و انعطاف‌پذیر است. از سال 2024، استلار نیز برای افزایش انعطاف‌پذیری، لایه قرارداد هوشمند خود به نام Soroban (مبتنی بر WebAssembly و زبان Rust) را به شبکه افزوده که امکان نوشتن کدهای سفارشی و پیچیده را فراهم می‌کند. با این حال Soroban از EVM استفاده نمی‌کند و استلار تصمیم گرفته مسیر خود را مستقل از روند رایج EVM پیش ببرد. در این پروژه فرضی، هدف آن است که بدون تغییر کل شبکه به Soroban یا اتریوم، یک گره انکر به طور اختصاصی توان بهره‌گیری از EVM را داشته باشد.

2. روش‌های اتصال EVM به شبکه استلار

برای برقراری ارتباط ماشین مجازی اتریوم (EVM) با شبکه‌ای که از پروتکل استلار استفاده می‌کند، چند رویکرد ممکن شناسایی شده است:

۲-۱. ارتباط مستقیم با نود استلار (پیاده‌سازی پروتکل استلار در کلاینت EVM)

در این روش بلندپروازانه، یک کلاینت اتریوم (مثلاً Geth) را طوری تغییر می‌دهیم که مستقیماً به شبکه استلار وصل شود. به عبارت دیگر، کلاینت EVM باید پروتکل همتابه‌همتای استلار و مکانیزم اجماع SCP را پیاده‌سازی یا پشتیبانی کند تا بتواند مانند یک گره بومی استلار در شبکه فعالیت کند. این کار مستلزم درک عمیق ساختار پیام‌های استلار (که مبتنی بر XDR هستند)، منطق انتخاب و بسته‌شدن تراکنش‌ها در هر دور اجماع، و تولید و به‌روزرسانی دفتر کل مطابق با سایر گره‌هاست. الگویی برای این رویکرد را می‌توان در پروژه‌های مشابه جستجو کرد؛ برای مثال Quorum (GoQuorum) که فورکی از Geth بود، توانست با جایگزینی مکانیزم اجماع RAFT/IBFT به جای PoW، یک شبکه اتریومی خصوصی ایجاد کند. بدین شکل، هسته اتریوم (اجرای EVM، مدل حساب و قراردادها) حفظ شد اما چگونگی اجماع و انتشار بلاک‌ها تغییر کرد. در مورد استلار، پیاده‌سازی SCP در یک کلاینت اتریومی از نظر مفهومی شبیه Quorum خواهد بود (جایگزینی PoS اتریوم با SCP استلار)، اما باید توجه داشت که Quorum بر مبنای کد اتریوم موجود ساخته شد در حالی که در اینجا پیاده‌سازی SCP از صفر در یک بستر غیربومی چالش‌برانگیز است. یک مسیر ممکن، اتصال کتابخانه Stellar-Core به کلاینت اتریوم است، به گونه‌ای که گره EVM ما به جای اتصال به شبکه اتریوم، به شبکه استلار جوین شود. در این سناریو، گره EVM به عنوان یک عضو شبکه استلار دیده می‌شود که می‌تواند بلوک‌های (دفتر کل‌های) جدید را از طریق SCP دریافت کرده و تغییرات را به روز کند. نتیجه این رویکرد آن است که کلاینت EVM به صورت بومی و همگام با سایر گره‌ها، داده‌های شبکه استلار را خواهد داشت و می‌تواند تراکنش‌ها را نیز مستقیماً به شبکه بفرستد. مزیت اصلی این روش، عدم نیاز به هیچ واسطه و دستیابی به یکپارچگی کامل با شبکه استلار است؛ بدین معنی که کلاینت EVM ما دقیقاً مانند یک نود رسمی استلار عمل می‌کند. اما نقطه ضعف بزرگ آن پیچیدگی فنی و هزینه توسعه بسیار بالا است. باید تمام جزئیات پروتکل استلار (از لایه انتقال گرفته تا منطق اجماع و اعتبارسنجی) در سوی کلاینت EVM بازتولید یا یکپارچه شود که نیازمند زمان و تخصص زیادی است. همچنین هر تغییر یا به‌روزرسانی در پروتکل استلار (ورژن‌های جدید) باید در این کلاینت سفارشی اعمال گردد که هزینه نگهداشت بالایی خواهد داشت.

۲-۲. استفاده از بریج یا میان‌افزار (واسط) بین کلاینت EVM و نود استلار

در این روش واقع‌گرایانه‌تر، به جای تغییر اساسی کلاینت اتریوم، یک لایه تطبیق‌دهنده (Adapter) میان EVM و شبکه استلار قرار می‌دهیم. به طور خلاصه، کلاینت یا ماشین مجازی EVM به طور مستقل اجرا می‌شود اما برای تعامل با شبکه استلار، درخواست‌ها و تراکنش‌های خود را از طریق یک واسطه برای یک نود استلار موجود ارسال می‌کند و پاسخ‌ها/رویدادها را از آن دریافت می‌کند. این واسط می‌تواند در ساده‌ترین حالت از APIهای موجود استلار (Horizon یا RPC) استفاده کند. به عنوان مثال، وقتی یک تراکنش در محیط EVM ثبت می‌شود، به جای پخش در شبکه اتریوم، از طریق واسط تبدیل به معادل یک تراکنش استلار (حاوی یک یا چند عملیات) شده و با فراخوانی Horizon API یا RPC به شبکه ارسال می‌شود. برعکس، برای خواندن اطلاعات، کلاینت EVM درخواست‌هایی مانند خواندن موجودی یا وضعیت را به واسط می‌دهد و واسط این درخواست را از شبکه استلار (مثلاً از Horizon) بازیابی کرده و نتیجه را در قالبی قابل فهم برای EVM بازمی‌گرداند. چنین پل IRPCای ماهیتاً نقش مترجم را بازی می‌کند: زبان Ethereum JSON-RPC را از سمت ابزارهای EVM می‌گیرد و به زبان REST یا RPC استلار تبدیل می‌کند و بالعکس. به طور مشخص، می‌توان چند سناریوی ممکن برای این پل متصور شد:

سرور RPC سفارشی «پل شفاف»

یک سرور RPC سفارشی که از دید ابزارهای اتریومی (مانند متامسک یا Web3.js) شبیه یک گره اتریوم رفتار می‌کند. این سرور متدهای پرکاربرد JSON-RPC اتریوم (مانند eth_sendTransaction, eth_getBalance, eth_call و ...) را پیاده‌سازی می‌کند اما منطق داخلی آن به جای رجوع به یک گره اتریوم، به یک گره استلار متصل است. برای نمونه، فراخوانی eth_getBalance(address) می‌تواند به یک خواندن موجودی حساب استلار از Horizon ترجمه شود؛ یا eth_sendTransaction که در دنیای اتریوم ارسال اتر یا فراخوانی یک قرارداد است، می‌تواند به ایجاد یک Operation پرداخت یا انتقال دارایی در استلار تبدیل شود. به این ترتیب، ابزارهایی نظیر متامسک ممکن است فکر کنند با یک شبکه اتریوم در تعامل‌اند، در حالی که سرور ما پشت صحنه در حال گفتگو با استلار است.

استفاده از SDK استلار در لایه اپلیکیشن «ترجمه دستی»

روش دیگر، استفاده از SDKهای استلار در داخل برنامه‌های اتریومی است. برای مثال، به جای ساخت کامل یک سرور واسط، می‌توان در لایه برنامه (application layer) توابع کتابخانه‌ای را قرارداد که کار ترجمه را انجام دهند. تصور کنید یک برنامه Node.js که از اترز (Ethers.js) برای کار با قراردادهای Solidity استفاده می‌کند، همزمان بتواند از SDK جاوااسکریپت استلار نیز بهره ببرد. توسعه‌دهنده می‌تواند با نوشتن کدهایی، خروجی یک قرارداد Solidity را گرفته و یک تراکنش استلار متناظر بسازد و ارسال کند. البته این کار دستی‌تر است و نمی‌توان از ابزارهای آماده مثل متامسک به طور شفاف بهره برد. بنابراین سناریوی سرور RPC شفاف‌تر و کاربرپسندتر خواهد بود.

قرارداد API + EVM استلار «محاسبه موازی»

اگر نیاز باشد برخی منطق‌های پیچیده در سمت انکر اجرا شود، می‌توان بخشی از آن منطق را داخل یک قرارداد هوشمند solidity روی EVM لوکال پیاده کرد که خروجی آن صرفاً تصمیم‌گیری یا محاسبات است. سپس نتیجه این قرارداد (مثلاً تأیید کند یک شرط برقرار است) توسط یک ماژول، منجر به فراخوانی API استلار (ارسال تراکنش) می‌شود. در این حالت، قرارداد Solidity به صورت chain-internal اجرا نمی‌شود، بلکه صرفاً یک محاسبه خارج‌زنجیره برای کمک به تصمیم‌گیری است.
مزیت بزرگ معماری بریج/میان‌افزار این است که پیاده‌سازی به مراتب ساده‌تری نسبت به راهکار کاملاً مستقیم دارد. ما به یک نود استلار موجود (یا حتی خدمات API استلار مانند Horizon عمومی) تکیه می‌کنیم و نیازی نیست تمام جزئیات پروتکل استلار را خودمان درگیر شویم. همچنین این روش امکان توسعه سریع‌تر یک نمونه آزمایشی (MVP) را می‌دهد. به عنوان گواه، پروژه‌هایی مانند MetaMask Snap برای استلار نشان داده‌اند که می‌توان با یک لایه‌ی واسط، متامسک را به شبکه‌هایی مثل استلار متصل کرد. در StellarSnap، در واقع متامسک از طریق یک افزونه (Snap) به روش‌های شبکه استلار دسترسی می‌یابد و امکان امضای تراکنش‌های استلار و تعامل با dAppهای استلار فراهم می‌شود. این مثال عملی مؤید امکان‌پذیری رویکرد پل RPC است. البته چالش‌هایی هم وجود دارد؛ از جمله تطابق مدل‌های داده: مثلاً در اتریوم مفهوم Gas و nonce برای تراکنش‌ها وجود دارد در حالی که در استلار مفهوم کارمزد به شکل ثابت/شناور ساده‌تری است و nonce معادل Sequence number حساب‌هاست. یا در اتریوم قراردادهای هوشمند داخل خود EVM ذخیره و اجرا می‌شوند، ولی در استلار سنتی چنین مفهومی نیست (تا قبل از Soroban).

بنابراین اگر برنامه اتریومی سعی کند یک قرارداد را Deploy کند (eth_sendTransaction به آدرسی خالی)، ما باید این درخواست را مدیریت کنیم که طبیعتاً نمی‌توان آن را به عملی در استلار تبدیل نمود (مگر آنکه Soroban فعال باشد که موضوع دیگری است). در نتیجه، پیاده‌سازی واسط باید محدودیت‌ها را بشناسد و فقط زیرمجموعه‌ای از قابلیت‌های EVM که معادل در استلار دارند را پشتیبانی کند. برای مثال، تراکنش‌های انتقال توکن می‌توانند به انتقال دارایی استلار مپ شوند، ولی تراکنش‌های تعاملی پیچیده قرارداد در شبکه استلار پایه جایی ندارند. (در بخش‌های بعد در مورد امکان اجرای برخی منطق‌های قرارداد در خارج زنجیره بحث خواهد شد.)

۳-۲. روش ترکیبی یا هیبریدی

این رویکرد ترکیبی از دو روش بالاست. به این صورت که گره انکر دو جزء کلیدی دارد: یکی ماژول اجماع استلار و دیگری ماژول EVM. ماژول اجماع استلار یا خود یک نود کامل Stellar-Core است یا بخشی از کد آن که فقط وظیفه حفظ همگامی دفتر کل با شبکه را بر عهده دارد. ماژول EVM نیز یک اجرای کامل ماشین مجازی اتریوم (مثلاً geth، یا حتی یک VM سبک مانند evmone) است که توانایی اجرای کدهای Solidity را دارد. بین این دو ماژول یک واسط داخلی برقرار است که داده‌های لازم را مبادله می‌کند. به عنوان نمونه، ماژول اجماع استلار می‌تواند پس از هر بستن دفتر کل جدید، رویداد یا پیغامی به ماژول EVM بفرستد و اطلاع دهد که وضعیت شبکه به‌روز شد. سپس ماژول EVM می‌تواند اطلاعات مورد نیاز (مثلاً موجودی‌ها، یا نتیجه تراکنش‌هایی که انکر فرستاده بود) را از ماژول استلار بخواند (احتمالاً از طریق APIهای لوکال Stellar-Core یا از دیتابیس history) و در صورت لزوم قراردادهای مرتبط را داخل خود EVM شبیه‌سازی کند تا تصمیم‌گیری‌های لازم را انجام دهد. هر زمان هم که نیاز به ارسال تراکنش جدیدی باشد، ماژول EVM درخواست را به ماژول استلار تحویل می‌دهد تا در قالب یک تراکنش استلار واقعی منتشر شود. این معماری هیبرید در واقع سعی می‌کند بهترین‌های هر دو دنیا را با هم ترکیب کند: از یک سو گره انکر یک عضو معتبر شبکه استلار است (چون ماژول Stellar-Core دارد و در اجماع SCP شرکت می‌کند)، از سوی دیگر یک محیط EVM ایزوله‌شده در داخل خود دارد که می‌تواند هر کد Solidity دلخواهی را اجرا کند (البته نتایج این کدها تنها برای خود انکر و تصمیماتش معتبر است، نه اینکه مستقیماً روی زنجیره ذخیره شوند). چنین معماری‌ای را می‌توان شبیه به اجرای یک sidechain/relay شخصی تصور کرد که به صورت یکطرفه یا دوطرفه به شبکه اصلی چسبیده است. نمونه‌هایی در سایر بلاک‌چین‌ها وجود داشته که یک زنجیره ثانویه EVM را به یک شبکه اصلی متصل کرده‌اند. برای مثال، شبکه Velo رویکرد دو زنجیره‌ای اتخاذ کرده است: استفاده از استلار برای سرعت و هزینه کم تراکنش‌ها، و یک زنجیره دیگر به نام Nova برای قراردادهای هوشمند EVM، و ارتباط این دو از طریق مکانیزم‌های بریج. در مورد انکر ما، چون تنها یک گره است، زنجیره EVMش می‌تواند حتی نیاز به اجماع توزیع‌شده نداشته باشد (یک زنجیره محلی تک-گرهی). از این رو شاید لفظ «زنجیره» دقیق نباشد و بهتر است آن را یک لایه اجرا (Execution Layer) خصوصی بنامیم که روی دفتر کل استلار سوار شده است.

به طور کلی، روش‌های ممکن برای اتصال EVM به استلار همین سه دسته (مستقیم، با واسط، هیبرید) را شامل می‌شوند. در عمل، ممکن است راه‌حل نهایی تلفیقی از این‌ها باشد. برای مثال، ابتدا یک واسط سبک Horizon-based توسعه دهیم (روش ۲-۲) و در ادامه برای بهبود کارایی یا استقلال، تدریجاً بخش‌هایی از منطق Stellar-Core را مستقیماً در کلاینت اضافه کنیم (حرکت به سمت ۲-۳ یا ۲-۱).

3. امکان‌سنجی استفاده‌ی موازی از EVM و نرم‌افزار استلاری

یکی از نگرانی‌های مهم، ریسک‌ها و چالش‌های اجرای چنین سیستمی در شبکه موجود است. برای کاهش ریسک، می‌توان دوره‌ای از اجرای موازی (Parallel Run) را در نظر گرفت که در طی آن انکر مورد نظر، هم‌زمان از زیرساخت معمول استلار و زیرساخت جدید مبتنی بر EVM بهره ببرد. این استراتژی اجازه می‌دهد عملکرد رویکرد جدید به صورت بلادرنگ و در شرایط واقعی شبکه مورد ارزیابی قرار گیرد، بدون آنکه وابستگی کامل به آن ایجاد شده باشد. نحوه‌ی اجرای موازی: انکر می‌تواند همچنان یک نود استلار استاندارد (Stellar Core به همراه Horizon) را اجرا کند تا وظایف حیاتی خود را تضمین کند (مثلاً انتشار و تبادل دارایی‌ها به روش سنتی). در کنار آن، گره/سیستم EVM جدید نیز راه‌اندازی می‌شود که به شبکه گوش می‌دهد و تراکنش‌های انکر را نیز مانیتور می‌کند. برای مدتی، این سیستم EVM به صورت غیرفعال/خواندن کار خواهد کرد؛ یعنی بدون ارسال تراکنش واقعی، فقط وضعیت شبکه را دریافت کرده و فرآیندهای درونی خود (مثلاً اجرای قراردادهای Solidity مربوط به انکر) را انجام می‌دهد. نتیجه این فرآیندها در محیط

EVM می‌تواند با خروجی‌های سیستم فعلی مقایسه شود تا اطمینان حاصل شود که تصمیمات و عملکرد هر دو منطبق است. به عنوان مثال، فرض کنیم انکر مسئول انجام پرداخت‌های خودکار بر اساس شرایط خاصی است. در سیستم فعلی، این منطق شاید توسط یک برنامه متمرکز یا اسکریپت روی Horizon انجام می‌شود. همزمان می‌توان همین منطق را در قالب یک قرارداد Solidity در محیط EVM نوشت. در دوره موازی، قرارداد Solidity در فواصل مشخص اجرا می‌شود و تصمیماتی (مثلاً «آیا پرداخت را انجام بدهم یا خیر») می‌گیرد. این تصمیمات با اعمالی که سیستم قدیمی انجام داده مقایسه می‌شود. تا زمانی که خروجی هر دو یکسان است، نشان‌دهنده صحت عملکرد EVM است. هر جا اختلافی مشاهده شود، فرصتی برای اشکال‌زدایی و بهبود سیستم جدید است. همچنین می‌توان برخی تراکنش‌های شبکه را دو بار و با دو روش انجام داد: یک بار از طریق سیستم معمول استلار و یک بار (به موازات آن) از طریق سیستم EVM (اما مثلاً تراکنش دوم را روی یک شبکه تست استلار یا به شکل شبیه‌سازی‌شده ارسال کنیم). این کار عملکرد end-to-end رویکرد EVM (از تولید تراکنش تا ارسال و دریافت پاسخ) را در شرایط واقعی می‌سنجد. عدم تداخل با اعضای شبکه: طی این دوره تست، سایر گره‌ها و اعضای شبکه نیازی به دانستن چیزی از وجود سیستم EVM ما ندارند. چون انکر کماکان تعهدات خود را با سیستم قدیمی انجام می‌دهد و تراکنش‌های معتبر مطابق پروتکل استلار به شبکه می‌فرستد. در واقع، EVM ما یا تراکنشی به شبکه نمی‌فرستد (در فاز مانیتورینگ) و یا اگر هم بفرستد، تراکنشی استاندارد و قابل پردازش توسط همه گره‌ها خواهد بود (چرا که نهایتاً از طریق Stellar Core/Horizon ارسال می‌شود). بنابراین از دید سایرین، هیچ تغییر غیرعادی رخ نمی‌دهد. سوئیچ تدریجی: پس از کسب اطمینان در مرحله موازی، انکر می‌تواند به صورت تدریجی بار عملیاتی را به سیستم جدید منتقل کند. برای مثال ابتدا ۱۰٪ تراکنش‌های خود را مستقیماً با سیستم EVM تولید و ارسال کند و ۹۰٪ همچنان از سیستم قدیم باشد. به مرور این سهم افزایش یابد تا نهایتاً سیستم قدیمی صرفاً به عنوان پشتیبان باقی بماند. این روش «بلوغ تدریجی» ریسک اختلال را کمینه می‌کند. همچنین در طول این دوره، شاخص‌های عملکرد (زمان پاسخ، نرخ موفقیت تراکنش، تطابق وضعیت‌ها و ...) باید پایش شوند تا تفاوتی بین دو سیستم مشاهده نگردد یا اگر هست، توجیه‌پذیر باشد. چالش‌های همزمانی: اجرای دو سیستم به طور همزمان نیازمند دقت در جلوگیری از اعمال دوگانه یا تناقض در وضعیت است. لذا احتمالاً باید یک‌طرفه بودن سیستم جدید را حفظ کرد تا وقتی کاملاً جایگزین شود. یعنی در دوره تست، سیستم EVM تصمیم‌گیر نهایی نیست و صرفاً توصیه‌کننده یا ناظر است. پس از کسب اطمینان، نقش تصمیم‌گیری به آن منتقل می‌شود. به طور خلاصه، استفاده موازی از هر دو رویکرد، کاملاً امکان‌پذیر و حتی توصیه‌شده است تا قبل از سوئیچ کامل، کارایی و درستی سیستم EVM در میدان واقعی اثبات شود. این رویکرد همچنین به تصمیم‌گیران فنی و مدیریتی امکان مشاهده تدریجی مزایا و معایب را می‌دهد.

4. ابزارها، کتابخانه‌ها و فریم‌ورک‌های مفید

برای پیاده‌سازی این پروژه بلندپروازانه، می‌توان از مجموعه‌ای از ابزارها و کتابخانه‌های موجود بهره برد تا روند توسعه تسهیل شود:

گث (Geth) یا سایر کلاینت‌های اتریوم

Geth (پیاده‌سازی Go-lang اتریوم) یک گزینه طبیعی برای پایه کار است، چرا که متن‌باز بوده و به خوبی مستندسازی شده است. می‌توان از Geth به عنوان هسته EVM استفاده کرد و در صورت نیاز تغییرات دلخواه (در اجماع، شبکه، RPC) را روی آن اعمال کرد. برای مثال، GoQuorum که توسط جی‌پی‌مورگان توسعه داده شد، یک fork از Geth بود که اجماع RAFT را اضافه و ویژگی‌های جدیدی مانند حریم خصوصی تراکنش‌ها را فراهم کرد. ما نیز می‌توانیم از تجربه Quorum بهره برده و Geth را fork کنیم تا پروتکل استلار را صحبت کند یا حداقل RPCهای دلخواه ما را پشتیبانی کند. جایگزین Besu ،Hyperledger Geth (به زبان جاوا) یا Nethermind (سی‌شارپ) نیز هست، اما Geth بدلیل جامعه کاربری بزرگ و نزدیکی به موارد استفاده مشابه (Quorum) مزیت دارد.

ماشین مجازی اتریوم سبک (evmone و دیگران)

اگر هدف این باشد که یک VM کارآمد به صورت کتابخانه در پروژه خود ادغام کنیم، پروژه evmone می‌تواند مفید باشد. evmone یک اجرای بسیار سریع و بهینه از EVM (توسعه‌یافته توسط Ethereum C++ team) است که به صورت یک کتابخانه (C++) در دسترس است. با evmone می‌توانیم کدهای باینری قراردادهای Solidity را در برنامه خود (مثلاً در یک service داخل نود انکر) اجرا کنیم. این در سناریوی معماری هیبرید (بخش ۲-۳) به کار می‌آید که بخواهیم بدون اجرای یک گره اتریوم کامل، توانایی اجرای OPCodeهای EVM را داشته باشیم. افزون بر evmone، ماشین مجازی‌های دیگر نظیر SputnikVM (Rust) یا EthereumJS VM

(برای جاوااسکریپت) نیز هستند که بسته به تکنولوژی انتخابی می‌توان استفاده کرد. هدف از این ابزارها، عدم نیاز به پیاده‌سازی مستقیم یک ماشین مجازی اتریوم از صفر است.

کتابخانه‌ها و SDKهای استلار

در سمت تعامل با شبکه استلار، برای سهولت می‌توان از SDK رسمی استلار در زبان‌های مختلف (مانند JavaScript, Python, Go و ...) استفاده کرد. این SDKها وظیفه ساخت تراکنش‌های استلار، امضا و ارسال به Horizon یا مستقیماً به Stellar-Core را ساده می‌کنند. مثلاً JS) stellar-sdk) این امکان را می‌دهد که با چند خط کد، یک تراکنش پرداخت یا ایجاد دارایی ساخته و امضا شود. به خصوص اگر رویکرد واسط (۲-۲) مدنظر باشد، داشتن SDK استلار در سمت سرور واسط می‌تواند ترجمه‌ی درخواست‌های EVM به تراکنش‌های استلار را سریعتر عملی کند. همچنین کتابخانه‌های مربوط به XDR استلار (برای سریال‌سازی/دسریال‌سازی پیام‌ها) و در صورت لزوم کلاینت‌های سطح پایین‌تر برای ارتباط با خود Stellar-Core (از طریق دستوراتی که فراهم می‌کند) کاربرد خواهند داشت.

ethers.js و web3.js

این دو کتابخانه محبوب جاوااسکریپتی، جزء لاینفک توسعه نرم‌افزارهای اتریومی هستند. هر دوی این کتابخانه‌ها قابلیت اتصال به یک گره اتریوم از طریق URL RPC را دارند و مجموعه کاملی از توابع برای ارسال تراکنش، فراخوانی قرارداد، خواندن eventها و ... ارائه می‌دهند. در پروژه ما، هرگاه بخواهیم یک رابط کاربری یا سرویس بیرونی به محیط EVM ما متصل شود، می‌توانیم با راه‌اندازی یک end-point RPC سازگار، استفاده از اترز یا وب۳ را برای توسعه‌دهندگان مقدور کنیم. برای مثال، برنامه وب یا موبایل انکر می‌تواند به جای استفاده از Horizon SDK، از اترز.js استفاده کند و به سرور واسط ما وصل شود که آن هم درخواست‌ها را ترجمه می‌کند. نتیجه این خواهد بود که توسعه‌دهندگان فرانت‌اند، تجربه کار با شبکه را مثل یک شبکه اتریومی (مثلاً xDai یا BSC) خواهند داشت و نیازی نیست جزئیات استلار را بدانند. علاوه بر این‌ها، ابزارهای توسعه مانند Hardhat یا Truffle نیز که مبتنی بر همین کتابخانه‌ها هستند می‌توانند برای نوشتن تست‌ها و اسکریپت‌های استقرار به کار گرفته شوند.

ابزارهای تبدیل و پل‌های ارتباطی

بسته به طراحی دقیق سیستم، ممکن است به ابزارهای خاص برای تبدیل داده‌ها نیاز داشته باشیم. برای نمونه، اگر قرار باشد یک تراکنش اتریوم (RLP انکود شده با امضای secp256k1) مستقیماً توسط یک ماژول خوانده و به تراکنش استلار تبدیل گردد، به توابعی برای دسریال‌سازی RLP و استخراج فیلدهای آن نیاز است (کتابخانه‌های RLP متعددی در دسترس هستند). همچنین برای تولید تراکنش استلار، باید از کتابخانه XDR استلار استفاده کنیم تا ساختار دنباله‌بایتی مناسب ساخته شود. این کار در سطح پایین است، اما همان‌طور که گفته شد می‌توانیم در سطوح بالاتر با SDKها آن را ساده‌تر کنیم. ابزار دیگر، MetaMask Snapها هستند که در بالا اشاره شد؛ در صورت تمایل به استفاده از متامسک بدون حتی داشتن یک سرور RPC واسط، می‌توان Snap اختصاصی نوشت تا متامسک مستقیماً از طریق آن Snap با Horizon صحبت کند. این رویکرد توسط پروژه StellarSnap تست شده و امکان امضای تراکنش‌های استلار و مشاهده موجودی حساب استلار را در متامسک فراهم کرده است. هر چند در معماری ما وجود یک سرور واسط منعطف‌تر و قدرتمندتر خواهد بود، اما Snaps متامسک نمونه‌ای از ابزارهای پیام‌تبدیل هستند که می‌توانند مفید باشند.

سایر ابزارهای بلاک‌چینی متن‌باز

پروژه‌های شناخته‌شده‌ای نظیر Cosmos SDK و Substrate نیز وجود دارند که اجازه می‌دهند بلاک‌چین‌های سفارشی بسازیم. اگر در آینده تصمیم به ایجاد یک زنجیره کاملاً جداگانه با ترکیب ویژگی‌های استلار و EVM باشد، این فریم‌ورک‌ها ارزش بررسی دارند. به طور خاص، Cosmos SDK قبلاً با پروژه Ethermint نشان داده که می‌تواند EVM را روی اجماع Tendermint اجرا کند. Substrate (محصول Parity) نیز پالت‌هایی برای EVM Compatibility دارد (مثال: Moonbeam روی پولکادات). گرچه فعلاً هدف ما حفظ شبکه استلار است نه ایجاد زنجیره جدید، اما داشتن دید نسبت به این پلتفرم‌ها می‌تواند در تصمیم‌گیری بلندمدت مفید باشد.

5. نمونه‌ها و پروژه‌های مشابه یکپارچه‌سازی EVM با سایر پروتکل‌ها

ایده‌ی استفاده از EVM در شبکه‌ها یا شرایط غیراتریومی بی‌سابقه نیست و در سال‌های اخیر چندین پروژه سعی در ایجاد سازگاری EVM در بسترهای مختلف داشته‌اند که مرور آن‌ها مفید است:

Ethermint / Evmos Cosmos

یکی از اولین پروژه‌های موفق در این زمینه، Ethermint بود که بعداً در قالب Evmos ادامه یافت. این پروژه نشان داد که می‌توان ماشین مجازی اتریوم را روی یک موتور اجماع متفاوت (Tendermint) اجرا کرد. Ethermint به زبان Go و بر پایه Cosmos SDK ساخته شد و توانست یک بلاک‌چین PoS با فاینالیتی سریع ارائه دهد که کاملاً با Solidity و ابزارهای اتریوم سازگار است. این یعنی توسعه‌دهندگان می‌توانستند قراردادهای خود را بدون تغییر، روی یک شبکه مبتنی بر Tendermint (همان اجماع کازموس) اجرا کنند. موفقیت Ethermint/Evmos گواهی است بر امکان جداسازی لایه اجرای EVM از لایه اجماع اتریوم و تطبیق آن با پروتکل‌های دیگر.

شبکه‌های سازگار با EVM مثل Polygon، Avalanche C-Chain، Tron، BSC

بسیاری از زنجیره‌های مطرح کنونی در واقع فورک یا بازنویسی اتریوم با اجماع متفاوت و تنظیمات اختصاصی هستند. بایننس اسمارت چین (BSC) نمونه‌ای است که با تغییر به یک مدل Proof of Authority با تعداد ولیدیتور محدود، توان عملیاتی اتریوم را بالا برد ولی EVM را حفظ کرد. ترون (Tron) نیز در ابتدا با ماشین مجازی اختصاصی شروع شد اما خیلی زود سازگاری با EVM را پیاده‌سازی کرد تا از انبوه ابزارها و قراردادهای اتریوم بهره‌مند شود. Tron Virtual Machine امروز تقریباً معادل یک محیط EVM عمل می‌کند. اولانچ Avalanche در زنجیره قرارداد هوشمند خود (Chain C) یک EVM کامل اجرا کرده که با اجماع Snowman (نسخه‌ای از Avalanche) کار می‌کند. پالیگان (Polygon) در ابتدا یک زیرساخت لایه۲ بود اما اکنون دارای زنجیره‌های مستقل (مانند Polygon POS) است که EVM را روی مکانیزم اجماع خودش اجرا می‌کنند. این نمونه‌ها همگی به ما نشان می‌دهند که EVM به عنوان یک استاندارد اجرای قرارداد می‌تواند در محیط‌های متفاوت به کار گرفته شود، کافیست لایه هماهنگ‌سازی (Consensus) و پارامترهای شبکه تنظیم شوند.

پروژه‌های قدیمی‌تر: Quorum، Hyperledger Burrow

پیشتر اشاره شد که Quorum (محصول JP Morgan) یک نمونه موفق از سازگار کردن اتریوم با نیازهای کنسرسیومی بود که اجماع RAFT (و بعد IBFT) را جایگزین ساخت. Hyperledger Burrow نیز تلاش دیگری (البته کم‌فروغ‌تر) بود که از کدبیس Tendermint بهره گرفت تا یک ماشین مجازی اتریوم ارائه دهد (Burrow در واقع حاصل همکاری Monax و Tendermint بود که بعدها زیاد رشد نیافت).

همگرایی زنجیره‌های غیر اتریومی با EVM

در سال‌های اخیر حتی زنجیره‌هایی که قبلاً خود را کاملاً متفاوت معرفی کرده بودند نیز به سمت EVM متمایل شدند. تزوس (Tezos) که زبان قرارداد Michelson مخصوص خود را داشت، یک استاندارد سازگاری به نام Edo و بعدتر پروپوزال‌هایی برای اجرای EVM (مثلاً از طریق رول‌آپ) ارائه داد. ایاس (EOS) با وجود داشتن EOS VM، یک لایه سازگار با EVM به نام TrustEVM را بررسی کرد تا قراردادهای سالیدیتی روی EOS قابل اجرا شوند. جالب‌تر از همه، ریپل (XRP Ledger) که سال‌ها ضدیت با قراردادهای تورینگ‌کامل داشت، اخیراً از راه‌اندازی یک زنجیره جانبی EVM خبر داده است (پروژه‌هایی مانند Flare و پیشنهاد Hooks در XRP). حتی IOTA که ساختارش UTXOگونه و مبتنی بر گراف جهت‌دار (DAG) است، با پروژه Assembly و Shimmer در صدد فراهم کردن محیط EVM برآمد. تمام این موارد نشان می‌دهد که استاندارد EVM به دلیل تسهیل توسعه و جامعه بزرگ، بسیار مورد توجه شبکه‌های مختلف قرار گرفته است و مهاجرت یا همزیستی با EVM تبدیل به یک روند در صنعت شده است.

Soroban در استلار

اگرچه Soroban یک VM متفاوت (WASM) است، اما نباید از ذکر آن غافل شد زیرا تلاش اصلی استلار برای هوشمندسازی شبکه‌اش محسوب می‌شود. Soroban به نوعی رقیب EVM است در اکوسیستم استلار، اما هنوز در ابتدای راه می‌باشد. پروژه ما از این جهت خاص است که به جای استفاده از Soroban، می‌خواهد مستقیماً EVM را وارد ماجرا کند.

پروژه Velo (دو زنجیره‌ای)

در اکوسیستم استلار، پروژه Velo شبکه‌ای ساخت که دو بلاک‌چین مکمل داشت؛ یکی بر بستر استلار برای مزایای سرعت و هزینه (انتقال توکن‌های VELO و دارایی‌های متصل به فیات)، و دیگری یک بلاک‌چین جدا به نام Nova برای پشتیبانی از قراردادهای هوشمند سازگار با اتریوم. Velo از بریج برای ارتباط این دو استفاده می‌کند به نحوی که دارایی‌ها بین استلار و Nova منتقل می‌شوند. این معماری دوگانه به Velo امکان داد از توان هر دو پلتفرم استفاده کند. شباهت ایده ما با Velo در بهره‌گیری از مزایای استلار (در لایه پایه) و قابلیت‌های EVM (در لایه کاربردی) است، با این تفاوت که در پروژه فعلی، هر دو لایه در کنترل یک گره (انکر) خواهند بود و نیازی به دو شبکه عمومی جداگانه نیست.
با مطالعه پروژه‌های فوق، دو درس مهم گرفته می‌شود: اول آنکه پیاده‌سازی EVM در محیط‌های جدید کاملاً ممکن است و بارها انجام شده (چه با تغییر کد اتریوم، چه با قرارداد یا زنجیره جانبی). دوم آنکه دلایل گرایش به EVM معمولاً بهره‌گیری از جامعه بزرگ توسعه‌دهندگان و ابزارها بوده است. در تصمیمات طراحی خود می‌توانیم از این تجربیات بهره ببریم (مثلاً انتخاب استفاده از fork Geth + SCP شبیه رویکرد Quorum، یا Dual-chain شبیه Velo).

6. طراحی معماری پیشنهادی

پس از بررسی رویکردهای ممکن، در این بخش یک معماری پیشنهادی برای پیاده‌سازی EVM در نقش کلاینت شبکه استلار ارائه می‌شود. این طراحی تلاش می‌کند تا حد امکان ساده، قابل پیاده‌سازی و توسعه‌پذیر باشد و ریسک‌های فنی را کاهش دهد، در عین اینکه اهداف پروژه (استفاده از ابزارهای EVM و عدم نیاز به تغییر در شبکه اصلی) را برآورده کند.
شمای کلی از معماری پیشنهادی برای یکپارچه‌سازی EVM با شبکه استلار. در این معماری، گره انکر شامل یک بخش EVM (کلاینت اتریوم سفارشی) و یک بخش ارتباطی با شبکه استلار است. رابط JSON-RPC امکان تعامل ابزارهای اتریومی (مثل متامسک یا Web3.js) را با این گره فراهم می‌کند. گره از طریق آداپتر پروتکل استلار یا Horizon، تراکنش‌ها را به شبکه استلار ارسال و دفاتر کل را دریافت می‌کند. رویدادها و به‌روزرسانی‌های دفتر کل نیز به بخش EVM منعکس می‌شود تا وضعیت داخلی به‌روز بماند.

مولفه‌های اصلی معماری

۱) لایه کاربرد و ابزارهای کاربری

این لایه شامل همان برنامه‌ها، کیف‌پول‌ها و اسکریپت‌هایی است که کاربر یا توسعه‌دهنده با آن سروکار دارد. فرض بر این است که این ابزارها سازگار با اتریوم هستند؛ مانند کیف پول MetaMask، کتابخانه Ethers.js، رابط‌های Web3، یا حتی برنامه‌های غیرمتمرکز (DApp) موجود. به دلیل معماری ما، این ابزارها نیاز به تغییر اساسی ندارند و تصور می‌کنند با یک شبکه اتریوم تعامل می‌کنند. برای مثال، یک برنامه وب که از web3.js استفاده می‌کند تا به شبکه ما متصل شود یا MetaMask که به‌جای شبکه اتریوم به شبکه خصوصی انکر (Network X با EVM) متصل می‌شود.

۲) گره سفارشی انکر (کلاینت EVM سفارشی)

این بخش، قلب راه‌حل است. گره سفارشی از دید بیرونی شبیه یک نود اتریوم عمل می‌کند اما در باطن با پروتکل استلار صحبت می‌کند. می‌توان آن را به دو زیرمولفه تقسیم کرد:

۲-الف) ماژول EVM و هسته اتریوم

شامل یک ماشین مجازی اتریوم کامل و اجزای مرتبط با آن (اجرای OPCODEها، مدیریت حساب‌ها و ذخیره‌سازی قراردادها، APIهای JSON-RPC اتریوم و غیره). این همان کدی است که می‌تواند قراردادهای Solidity را پردازش کند و تراکنش‌های اتریومی را بفهمد. استفاده از کد یک کلاینت موجود (مانند Geth) برای این بخش بسیار کمک‌کننده خواهد بود. در این ماژول، ما نیازی به مکانیزم اجماع PoW/PoS اتریوم نداریم چون قصد نداریم بلاک‌چین اتریوم جداگانه‌ای تشکیل دهیم. به جای آن، این ماژول هر زمان که تراکنشی دریافت می‌کند، آن را جهت اجرا/شبیه‌سازی به EVM می‌دهد و سپس خروجی (تغییر حالت مورد نیاز) را استخراج می‌کند. به عنوان مثال، اگر قرارداد Solidity یک event تولید کند یا قصد ارسال توکن ERC20 داشته باشد، ماژول EVM این را تشخیص می‌دهد.

۲-ب) ماژول تطبیق‌دهنده استلار

این بخش مانند پلی است که گره EVM را به شبکه استلار وصل می‌کند. وظایف این ماژول عبارت‌اند از: دریافت بلاک‌ها/دفتر کل‌های جدید از شبکه استلار، استخراج اطلاعات مرتبط (مثلاً تراکنش‌هایی که انکر فرستاده یا وضعیت دارایی‌های تحت نظارت)، و ارائه‌ی آنها به ماژول EVM؛ همچنین دریافت درخواست برای ارسال تراکنش از ماژول EVM و تبدیل آن به قالب تراکنش استلار و انتشار آن در شبکه. این ماژول خودش می‌تواند یکی از دو حالت را داشته باشد: یا مستقیماً بخشی از Stellar-Core (اجرای کامل یک نود) باشد، یا به یک نود استلار خارجی متصل شود (از طریق Horizon یا RPC). تصمیم بین این دو به پیچیدگی و اولویت‌های طراحی بستگی دارد. در فاز MVP ممکن است اتصال از طریق Horizon انتخاب شود (سادگی بیشتر)، اما در نسخه نهایی شاید یکپارچه‌سازی کامل با Stellar-Core (اجرای داخلی) صورت گیرد تا وابستگی خارجی حذف و سرعت بیشتر شود.

۳) شبکه استلار (گروه ولیدیتورها و SCP)

این قسمت همان شبکه بلاک‌چین خصوصی است که سایر اعضا اداره می‌کنند. گره انکر ما در این شبکه یا به عنوان یک اعتبارسنج (ولیدیتور) حضور دارد یا حداقل یک عضو مشاهده‌گر (Observer) است. شبکه طبق روال معمول خود (اجماع SCP) بلاک‌ها را تولید می‌کند و کلیه تراکنش‌های ارسالی از جمله تراکنش‌های انکر ما را پردازش می‌کند. برای شبکه تفاوتی نمی‌کند تراکنش انکر چگونه ایجاد شده؛ طالما که امضای معتبر داشته و از نوع قابل فهم است (مثلاً پرداخت، ایجاد حساب، تراست‌لاین، ...). دفتر کل‌های جدید شبکه نهایتاً مرجع نهایی وضعیت دارایی‌ها و تراکنش‌ها هستند.

تعامل بین مولفه‌ها (سناریوی عملکرد):

تعامل کاربر با گره انکر

کاربر یا برنامه، از طریق ابزارهای اتریومی (مثلاً متامسک یا اترز.js) یک درخواست ایجاد می‌کند. این می‌تواند ارسال یک تراکنش (مثلاً درخواست انجام مبادله دارایی) یا فراخوانی یک تابع قرارداد (مثلاً check کردن چیزی) باشد. این درخواست به شکل JSON-RPC به آدرس گره انکر (ماژول EVM) ارسال می‌شود. برای مثال، متامسک ممکن است از کاربر بخواهد تراکنشی را امضا کند؛ این تراکنش طبق فرمت اتریوم خواهد بود (شامل nonce، gas price=0 احتمالا، data یا value و...) و پس از امضا، متامسک آن را با متد eth_sendRawTransaction به گره ما می‌فرستد.

پردازش در گره انکر (EVM)

ماژول EVM تراکنش دریافتی را مانند یک نود اتریوم عادی ابتدا decode و بررسی می‌کند. اگر این تراکنش مثلاً یک تماس به قرارداد باشد (دارای EVM ،(data آن را بر روی وضعیت داخلی خود اجرا می‌کند. این وضعیت داخلی در ابتدا احتمالاً فقط شامل چند قرارداد مربوط به انکر است (چون سایر حساب‌ها شاید نیازی نباشد مدل‌سازی شوند). اجرای تراکنش به EVM امکان می‌دهد بداند چه اثری مورد انتظار است. اما توجه کنیم که وضعیت داخلی EVM الزاماً آینه کامل استلار نیست – تنها بخش‌هایی که ما نیاز داریم می‌توانیم مدل کنیم. برای نمونه، اگر قرارداد Solidity لازم است بداند یک کاربر چه میزان از یک توکن دارد، می‌توانیم در لحظه آن را از شبکه استلار استعلام کرده و به EVM بدهیم (مثلاً EVM را pause کرده، RPC به ماژول استلار بزنیم و نتیجه را برگردانیم – هرچند این کار بسیار پیچیده است چون EVM معمولاً چنین توقف‌هایی را پیش‌بینی نکرده). طراحی ساده‌تر آن است که قراردادهای Solidity ما در محیط انکر، خیلی وابسته به وضعیت بقیه نباشند و عمدتاً منطق کسب‌وکار انکر (که برای خودش قابل دسترس

است) را اجرا کنند. به عنوان خروجی، EVM تشخیص می‌دهد که مثلاً انکر باید ۱۰ توکن X را به حساب Y منتقل کند. این خروجی را ماژول تطبیق‌دهنده دریافت می‌کند.

ارسال تراکنش به شبکه استلار

ماژول تطبیق‌دهنده، با دانستن هدف مورد نظر (مثلاً انتقال توکن X)، یک یا چند Operation معادل در استلار می‌سازد. در این مثال، احتمالاً یک Operation از نوع پرداخت (Payment) یا پرداخت مسیردار (PathPayment) ساخته می‌شود که دارایی X را از حساب انکر به حساب Y منتقل کند. این Operation درون یک تراکنش استلار قرار می‌گیرد، با کلید خصوصی انکر امضا می‌شود و به شبکه ارسال می‌شود. ارسال می‌تواند از طریق Horizon (متد /transactions در REST) یا مستقیم از طریق اتصال به یک همتای SCP صورت گیرد. در هر حال، شبکه استلار تراکنش را دریافت کرده و طی یک یا دو سیکل اجماع آن را در دفتر کل بعدی می‌گنجاند. در این حین، ماژول تطبیق‌دهنده می‌تواند وضعیت تراکنش را دنبال کند (مثلاً از طریق /transactions/{hash} در Horizon) تا از موفقیت یا شکست آن مطلع گردد.

به‌روزرسانی وضعیت و رخدادها در محیط EVM

پس از بسته‌شدن دفتر کل، ماژول استلار یک رویداد (Event) داخلی به ماژول EVM ارسال می‌کند، یا خود ماژول EVM به صورت polling متوجه می‌شود که دفتر کل جدیدی آمده است. در این لحظه، مهم است که وضعیت داخلی EVM با واقعیت شبکه تنظیم (sync) شود. اگر ما تنها موجودی‌ها و برخی اطلاعات محدود را در EVM نگهداری می‌کنیم، به‌روزرسانی آنها ساده است (مثلاً جدول موجودی توکن X برای کاربر Y را آپدیت می‌کنیم). اگر قراردادی در EVM منتظر رخدادی بوده (مثلاً event که در Solidity تعریف شده)، ما می‌توانیم با یک ترفند آن را شبیه‌سازی کنیم: به عنوان مثال، وقتی انتقال توکن در استلار موفق شد، خود ماژول EVM یک رویداد Log) Solidity) ایجاد کند تا اگر کلاینت بیرونی منتظر events بود، بتواند آن را دریافت کند. البته این سطح از تطبیق پیچیده است و شاید در نسخه‌های اولیه پشتیبانی نشود. حداقل، قراردادهای Solidity مستقر در انکر می‌توانند حالت داخلی خود را با نتایج شبکه هماهنگ کنند (مثلاً شمارنده‌ای افزایش یابد یا رکوردی ثبت شود که فلان پرداخت انجام شد).

پاسخ‌دهی به کاربر/برنامه

در نهایت، کاربری که درخواست اولیه را فرستاده بود (مثلاً متامسک یا DApp)، انتظار نوعی پاسخ یا نتیجه دارد. برای متامسک، ارسال موفق eth_sendRawTransaction معمولاً یک TX hash برمی‌گرداند. ما می‌توانیم هش تراکنش استلار (یا یک هش پوششی شبیه اتریوم) را به عنوان پاسخ برگردانیم. اگر بعداً کاربر درخواست eth_getTransactionReceipt داد، گره ما می‌تواند با مراجعه به Horizon، وضعیت آن TX را ببیند و پاسخی حاوی وضعیت success/fail و شاید لاگ‌های رخداد (در صورت شبیه‌سازی) ارائه دهد. هدف این است که چرخه تعامل برای کاربر کاملاً شبیه تجربه اتریوم باشد، هرچند در پشت صحنه از استلار استفاده شده است.

جزئیات بیشتر: طراحی بالا را می‌توان بسته به نیاز دقیق ساده‌تر یا پیچیده‌تر نمود. یک تصمیم معماری مهم، میزان همگام‌سازی حالت بین EVM و استلار است. در یک طراحی مینیمال، ممکن است EVM فقط برای امتحان شرایط و منطق‌ها استفاده شود و تمام حالت‌های اصلی (دارایی‌ها، حساب‌ها) مستقیماً از استلار خوانده شود. در طراحی‌های پیچیده‌تر، ممکن است یک قرارداد Solidity حتی نسخه توکنیزه‌شده دارایی‌های استلار را درون خود نگه دارد. مثلاً یک قرارداد ERC20 نماینده یک دارایی استلار باشد که هر تغییر در استلار، معادل آن در این ERC20 به‌روز شود. این باعث می‌شود برنامه‌های سازگار با اتریوم واقعاً تصور کنند در حال تعامل با یک ERC20 هستند، در حالی که آن ERC20 فقط لایه واسطی است و دارایی واقعی روی استلار جابجا می‌شود. چنین الگویی در بریج‌های بین‌زنجیره‌ای رایج است (صدور نماینده دارایی یک شبکه در شبکه دیگر) و می‌توان از آن بهره برد. از حیث مقیاس‌بندی داخلی، چون انکر ما احتمالاً حجم زیادی تراکنش ندارد (وابسته به业务 انکر)، یک سرور نسبتاً معمولی با اجرای موازی کافی برای هر دو بخش EVM و استلار کفایت می‌کند. ممکن است تصمیم بگیریم Stellar-Core و کلاینت EVM را روی دو فرایند جداگانه اجرا کنیم که از طریق RPC داخلی یا message queue با هم در ارتباط باشند. این طراحی Microservice مانع تداخل می‌شود و استفاده مجدد از کامپوننت‌ها را بالا می‌برد (مثلاً می‌توان از یک Stellar-Core استاندارد استفاده کرد). یا برعکس، می‌توان بهینه‌سازی کرد و کد EVM را مستقیماً به کد Core اضافه کرد (fork واحد). این جزئیات در فاز پیاده‌سازی و با توجه به توان تیم قابل نهایی‌سازی است.

7. زمان‌بندی پیشنهادی برای توسعه MVP و نسخه پایدار

با توجه به گستردگی و نوآوری این پروژه، لازم است فازبندی مناسبی برای آن تعریف شود. در ادامه یک برنامه زمانی تقریبی (با فرض وجود یک تیم توسعه‌دهنده خبره در حوزه بلاک‌چین) ارائه می‌گردد:

1. مطالعه و طراحی اولیه (۴ هفته):

- آشنایی دقیق تیم با پروتکل استلار، ساختار Stellar-Core و Horizon.
- مرور نمونه‌های مشابه (Quorum, Ethermint, Stellar Snap و ...) و استخراج نکات فنی کلیدی.
- تهیه سند طراحی معماری (مشابه بخش 6 این گزارش) و تصمیم‌گیری درباره تکنولوژی‌های مورد استفاده (انتخاب کلاینت اتریوم پایه، زبان برنامه‌نویسی اصلی، نحوه اتصال به استلار و ...).
- شناسایی مخاطرات فنی (Risk) و برنامه‌ریزی برای رفع آنها. برای مثال، تصمیم‌گیری اینکه Soroban (در صورت فعال بودن روی شبکه خصوصی) در این پروژه نقشی خواهد داشت یا خیر.

2. پیاده‌سازی نمونه‌ی آزمایشی (MVP) – حدود ۳ ماه:

- فاز ۲-الف: توسعه واسط ساده: در ۴ تا ۶ هفته اول، ابتدا یک سرور واسط سبک راه‌اندازی می‌شود که یک یا دو متد JSON-RPC اتریوم (مثلاً eth_getBalance و eth_sendTransaction پایه) را پشتیبانی کند و آنها را به Horizon ترجمه نماید. این MVP اولیه می‌تواند بسیار محدود باشد (مثلاً فقط انتقال توکن XLM را پوشش دهد) اما اثبات می‌کند که حلقه ارتباطی کار می‌کند.
- فاز ۲-ب: یکپارچه‌سازی EVM: در ۶ تا ۸ هفته بعد، بخش EVM واقعی به پروژه اضافه می‌شود. این شامل راه‌اندازی یک ماشین مجازی اتریوم (احتمالاً با استفاده از geth در حالت private network) و ایجاد مکانیزمی برای دریافت رویداد دفتر کل استلار و به‌روزرسانی state می‌باشد. در پایان این فاز، انتظار می‌رود MVP قادر باشد یک قرارداد Solidity ساده (مثلاً قراردادی که با یک شمارنده، رسیدن پرداخت‌ها را پیگیری می‌کند) را اجرا کرده و هنگام وقوع تراکنش‌های واقعی استلار، آن شمارنده را افزایش دهد. همچنین از سمت کاربر، بتوان با Web3.js متصل شد و وضعیت آن شمارنده را پرسید (در پشت صحنه با RPC ما و ترجمه به Horizon).

3. تست و اصلاح MVP (۶-۴ هفته):

- اجرای سناریوهای تست روی MVP: تست انتقال دارایی، تست انطباق موجودی‌ها، تست استرس با تعداد تراکنش بالا به واسط و بررسی performance.
- دریافت بازخورد از تیم انکر و حتی اجرای یک دوره Parallel Run محدود. در این فاز، MVP روی شبکه خصوصی (یا شبکه تست مشابه) نصب شده و در کنار روش فعلی، به طور آزمایشی عمل می‌کند تا مشکلات عملی آشکار شود.
- مستندسازی عملکرد MVP و هرگونه محدودیت آن (مثلاً اگر MVP فرض کرده کارمزدها صفر باشد یا nonceها را ساده مدیریت کرده).

4. توسعه نسخه پایدار (Production-Ready) – حدود ۳ ماه دیگر:

- پس از اطمینان از مسیر تکنیکال، MVP گسترش می‌یابد: پوشش بیشتر متدهای JSON-RPC اتریوم، پشتیبانی از انواع عملیات بیشتر استلار (Trustline, Offer, ... در صورت نیاز)، بهبودهای امنیتی (مثلاً اعتبارسنجی بیشتر ورودی‌های کاربر)، افزودن قابلیت‌های مانیتورینگ و logging برای نظارت بر سیستم.
- در این فاز، اگر تصمیم به عمیق‌تر کردن یکپارچگی باشد، ممکن است به جای Horizon، گره انکر مستقیم به Stellar-Core متصل شود. پیاده‌سازی ماژول SCP در کلاینت EVM (حداقلی) یا استفاده از کتابخانه رسمی

استلار برای اتصال به overlay network می‌تواند جزو کارها باشد. این موضوع احتمالاً زمان‌بر است، لذا شاید تقسیم شود و در نسخه 1.0 مستقیماً نیاید.

- تست‌های واحد و یکپارچه‌ی مفصل نوشته و اجرا می‌شوند تا هر تغییر جدید پایداری را مختل نکند.

5. آزمایش نهایی و استقرار (۴ هفته):

- انتخاب یک بازه زمانی کم‌تراکم در شبکه برای روشن کردن موازی سیستم جدید در حالت فعال. یعنی انکر شروع به استفاده واقعی از سیستم EVM برای درصدی از عملیات کند.
- مشاهده رفتار شبکه و اطمینان از عدم وجود ایراد (باگ) بحرانی. در صورت رویت مشکل، سیستم به حالت قبلی برگردد و اصلاحات انجام شود.
- آماده‌سازی مستندات فنی و آموزشی برای تیم‌های مرتبط (تیم عملیات انکر، توسعه‌دهندگان فرانت‌اند که حالا باید به جای Horizon API از RPC جدید استفاده کنند و ...).

با این برنامه، برآورد می‌شود در مجموع حدود ۷ تا ۸ ماه برای رسیدن به نسخه تقریباً پایدار زمان نیاز باشد. البته برخی عوامل می‌توانند زمان را افزایش دهند، از جمله پیچیدگی‌های پیش‌بینی‌نشده در سازگاری پروتکل‌ها یا کمبود نیروی متخصص در بخشی از کار. بنابراین بازه‌ای در حدود ۹ تا ۱۲ ماه واقع‌بینانه است تا محصول نهایی کاملاً تست‌شده و قابل اتکا به بهره‌برداری برسد. در ضمن، می‌توان به موازات توسعه فنی، بحث‌های حقوقی/رگولاتوری (در صورت وجود) و مستندسازی را نیز پیش برد تا هنگام آماده‌شدن محصول، موانع غیر فنی جلوگیر نباشند.

8. مزایا و معایب رویکرد پیشنهادی در مقایسه با گزینه‌های دیگر

در پایان، لازم است رویکرد پیشنهادی (استفاده از EVM در نقش کلاینت استلار برای انکر) را از منظر استراتژیک با باقی گزینه‌ها مقایسه کنیم: ۱) باقی‌ماندن در معماری صرفاً استلار بدون قرارداد هوشمند، ۲) مهاجرت شبکه به قابلیت قرارداد هوشمند Soroban (و یا گزینه‌های مشابه).

مزایای رویکرد EVM یکپارچه‌شده:

استفاده از اکوسیستم غنی اتریوم

بزرگ‌ترین مزیت، بهره‌گیری از سال‌ها تجربه، ابزار و جامعه توسعه‌دهندگان اتریوم است. با این رویکرد، انکر و شرکای فنی‌اش می‌توانند از ابزارهای استاندارد مانند متامسک، Truffle/Hardhat، کتابخانه‌های سالیدیتی و ده‌ها ابزار دیگر استفاده کنند. این به معنای کاهش هزینه‌های آموزشی و توسعه است؛ زیرا یافتن توسعه‌دهنده سالیدیتی بسیار آسان‌تر از توسعه‌دهنده آشنا به پلتفرم خاص (چه استلار خالص، چه Soroban) است. همچنین بسیاری از الگوهای قرارداد و کامپوننت‌های آماده (مثلاً قراردادهای چند امضایی، کیف پول امن، کتابخانه‌های ریاضی و ...) در دنیای EVM وجود دارد که قابل بهره‌برداری‌اند.

افزایش قابلیت برنامه‌ریزی (Programmability) برای انکر

انکر با داشتن یک ماشین مجازی تورینگ کامل (EVM)، می‌تواند منطق‌های پیچیده مالی یا کسب‌وکاری خود را کدنویسی کند. در معماری استلار سنتی، بسیاری از فرایندهای پیچیده باید خارج از زنجیره (off-chain) انجام شوند و نتیجه‌شان به صورت تراکنش‌های ساده روی زنجیره اعمال شود. ولی با EVM، انکر می‌تواند بخشی از این منطق را به شکل قرارداد درآورد که نیمه‌خودکار بر اساس رویدادهای زنجیره عمل کند. به عنوان مثال، امکان نوشتن یک قرارداد هوشمند برای مدیریت وثیقه‌ها و وام‌دهی فراهم می‌شود که مستقیماً با تراکنش‌های استلار (انتقال دارایی) تعامل می‌کند. این ترکیب انعطاف زیادی به انکر می‌دهد تا خدمات جدیدی ارائه کند که قبلاً در محیط استلار ممکن نبود.

عدم تاثیر بر سایر اعضای شبکه

یکی از شروط اساسی پروژه این بود که سایر اعضا نفهمند یا نیاز نباشد بدانند انکر چه تغییراتی در سیستم خود داده است. رویکرد ما این شرط را برآورده می‌کند؛ چرا که پروتکل مشترک شبکه همچنان SCP و فرمت تراکنش‌های استلار باقی می‌ماند. به بیان دیگر، ما

Backwards Compatible هستیم. اگر به جای این کار تصمیم می‌گرفتیم کل شبکه را به مثلاً یک فورک اتریوم منتقل کنیم، همه باید تغییر می‌کردند که عملاً نقض این شرط بود.

پتانسیل ارائه خدمات برون‌مرزی و نوآورانه

با داشتن EVM، انکر می‌تواند پل‌های متنوعی با دنیای اتریوم و EVM برقرار کند. حتی می‌تواند به عنوان درگاه میان استلار و اتریوم عمل کند؛ مثلاً قراردادهای ما امکان تعامل با شبکه اتریوم اصلی از طریق پیام‌های متقابل (با استفاده از اوراکل‌ها یا بریج‌ها) داشته باشند. این امر می‌تواند کسب‌وکار انکر را توسعه دهد (مثلاً پذیرش وثیقه در قالب ETH یا ERC20 جهت صدور معادل آن در استلار و بالعکس).

سرعت نهایی شدن و کارمزد پایین (ارث رسیده از استلار)

برخلاف اجرای قرارداد روی اتریوم که ممکن است کارمزد بسیار بالا و تایید طولانی داشته باشد، ما اینجا بهترین حالت را داریم که اجرای قرارداد در محیط EVM داخلی سریع است و نهایی شدن تراکنش واقعی روی استلار نیز ظرف ۵ ثانیه با هزینه بسیار اندک انجام می‌شود. این برای تجربه کاربری نهایی عالی استمثلاً کاربر ظرف چند ثانیه نتیجه عمل خود (مثل مبادله) را خواهد دید، چیزی که در خود شبکه اتریوم ممکن است دقیقه‌ها طول بکشد. از این منظر، می‌توان گفت ما سرعت استلار را با قدرت EVM ترکیب کرده‌ایم.

معایب و چالش‌های رویکرد EVM:

پیچیدگی فنی و هزینه پیاده‌سازی

روشن است که پیاده‌سازی چنین سیستم نامتعارفی بسیار پیچیده‌تر از استفاده ساده از راهکارهای موجود است. اگر انکر صرفاً در معماری استلار باقی بماند، کافیست از ابزارهای اثبات‌شده SDF (نظیر Anchor Platform رسمی) یا قراردادهای ساده استلار استفاده کند که تقریباً آماده و آزمایش‌شده‌اند. یا اگر Soroban را انتخاب کند، با اینکه تازه است ولی حداقل رسمی و پشتیبانی‌شده است. در مقابل، رویکرد EVM ما کاملاً سفارشی است و تیم باید توان تخصصی در هر دو حوزه اتریوم و استلار داشته باشد. این نه تنها زمان توسعه را زیاد می‌کند بلکه ریسک باگ‌های پنهان را نیز افزایش می‌دهد (چون مسیری که کمتر کسی رفته احتمال خطا و گوشه‌های کشف‌نشده دارد). هزینه نگهداشت نیز قابل توجه خواهد بود؛ هر به‌روزرسانی حیاتی در پروتکل استلار یا کتابخانه‌های اتریوم ممکن است نیاز به اصلاحات در سیستم ما داشته باشد.

عدم بهره‌گیری از بهبودهای Soroban

استلار با معرفی Soroban در واقع در حال اضافه کردن قابلیت قرارداد هوشمند به شبکه است، البته به روشی متفاوت (WASM و نه EVM). یکی از دلایل این تصمیم، بهبود کارایی و امنیت بوده است. Soroban از زبان Rust استفاده می‌کند که ایمنی حافظه بهتری دارد و طراحی آن لحاظ‌کننده تجربه اتریوم و رفع نقایص آن است. اگر انکر EVM را بر Soroban ترجیح دهد، ممکن است از مزایای پیشرفت‌های آتی استلار محروم شود. برای مثال، شاید SDF در آینده ابزارهای مخصوص Soroban برای انکرها توسعه دهد یا مشوق‌هایی برای پروژه‌های Soroban محور ارائه کند. رویکرد ما چون غیررسمی است، احتمالاً خارج از این اکوسیستم باقی خواهد ماند.

مقیاس‌پذیری محدود به یک گره

یکی از تفاوت‌های ظریف، تفاوت بین یکپارچگی فردی و استاندارد شبکه است. اگر EVM به صورت رسمی وارد شبکه می‌شد (مثلاً همه می‌توانستند قرارداد Solidity اجرا کنند)، مزیت آن به کل اکوسیستم می‌رسید (توسعه‌دهندگان بیشتری جذب پلتفرم می‌شدند و غیره). اما در راهکار ما، EVM صرفاً در اختیار یک انکر است. این انکر طبیعتاً می‌تواند پروژه‌های خودش را روی آن توسعه دهد اما نمی‌تواند اکوسیستم توسعه‌دهندگان خارج را برای مشارکت مستقیم دعوت کند، چون سایر اعضا دسترسی به این لایه ندارند. از این جهت، ما یک حباب خصوصی می‌سازیم. این لزوماً بد نیست (چون هدف هم همین بوده که دیگران تغییر نکنند)، ولی از منظر خلق ارزش شبکه‌ای، به اندازه افزودن Soroban برای کل شبکه اثرگذار نخواهد بود.

مسائل احتمالی سازگاری و دوباره‌کاری

برخی از قابلیت‌های اتریوم ممکن است هرگز معادل واقعی در استلار نداشته باشند. مثلا قراردادهای خود-انگیز (که با دریافت Ether کار می‌کنند) یا DEFI پیچیده مبتنی بر mining liquidity. این موارد در محیط انکر قابل شبیه‌سازی است اما چون خروجی نهایی باید تراکنش استلار باشد، با محدودیت مواجه می‌شود. ممکن است تیم توسعه بعد از صرف زمان متوجه شود که برخی سناریوهای دلخواه اساساً قابل پیاده‌سازی روی استلار نیستند حتی با این حقه EVM. در این صورت، تلاش‌ها باید به سمت ساده‌سازی انتظارات یا تغییر بخشی از معماری برود. این ریسک در استفاده از Soroban کمتر است، چون Soroban عملاً رسمیت دارد و اگر کمبودی باشد SDF شاید آن را رفع کند، اما در رویکرد ما خودمان تنها هستیم.

دوباره‌کاری احتمالی در آینده

اگر روزی در آینده کل شبکه (با اجماع همه) تصمیم بگیرد رسماً EVM را وارد کند یا Soroban را همه‌گیر نماید، راه‌حل ما ممکن است بلا استفاده شود. البته شاید همین پروژه محرکی شود برای بررسی رسمی‌تر این ایده توسط SDF؛ ولی در هر حال، چون استاندارد شبکه نیست، عدم قطعیت بیشتری دارد.

مزایا/معایب در قیاس با Soroban:

Soroban به عنوان گزینه دیگر، مزیت یکپارچگی با شبکه را دارد (هر گره ولیدیتور اجرای WASM را خواهد داشت) و امنیتی که توسط طراحی SDF تضمین شده. همچنین Soroban با استفاده از WASM توان بالقوه اجرای انواع VMها از جمله EVM را هم داشت (حداقل تئوری، می‌توان یک مفسر بایت‌کد EVM را روی WASM سوار کرد، هرچند ناکارآمد است). اما معایب Soroban برای انکر ما عبارتند از: جدید بودن و کمبود ابزار – بسیاری از امکاناتی که در اکوسیستم Solidity طی سال‌ها ساخته شده (مثل کتابخانه‌های قرارداد OpenZeppelin، دیباگرهای بالغ، انجمن‌های پرسش/پاسخ)، برای Soroban در ابتدای راه است. زبان Rust نیز منحنی یادگیری خودش را دارد که شاید برای تیم انکر زمان‌بر باشد. بنابراین رویکرد EVM برای انکر، یک مسیر میانبر برای بهره‌گیری فوری از سالیدیتی است بدون انتظار برای بلوغ Soroban.

مزایا/معایب در قیاس با استلار صرف (بدون قرارداد):

اگر اصلاً قرارداد هوشمندی در کار نباشد و انکر صرفاً از قابلیت‌های پایه استلار استفاده کند، توسعه ساده‌تر است اما محدودیت شدید دارد. عملاً بسیاری از فرایندها را نمی‌توان خودکار و درون‌زنجیره‌ای کرد (مثلاً وام‌دهی با شرایط دلخواه، معاملات اهرمی، قراردادهای مشتقه و...). انکر ممکن است بتواند راه‌حل‌های off-chain برای بعضی را تدارک ببیند ولی آن راه‌حل‌ها هرگز شفافیت و قدرت قراردادهای هوشمند را ندارند. بنابراین اگر انکر چشم‌انداز ارائه خدمات مالی پیشرفته را دارد، ماندن در حالت سنتی احتمالاً کافی نخواهد بود. افزون بر آن، استفاده صرف از قابلیت‌های استلار، انکر را محدود به ابزارهای موجود (Horizon و غیره) می‌کند که تجربه کاربری توسعه‌دهندگان چندان غنی نیست؛ برای مثال، در دنیای استلار سنتی هیچ معادلی برای چیزی مانند Metamask + Hardhat جهت توسعه سریع یک DApp وجود ندارد. در عوض، توسعه‌دهنده باید خیلی چیزها را خود بسازد. این ممکن است سرعت نوآوری را کاهش دهد.

جمع‌بندی مزایا/معایب

رویکرد یکپارچه‌سازی EVM پیشنهادی، یک راهکار ابتکاری و قدرتمند است که می‌تواند بهترین‌های دو دنیا (شبکه سریع استلار و اکوسیستم غنی اتریوم) را برای انکر فراهم کند. اما این راه پرهزینه و چالش‌برانگیز است و نیاز به تعهد جدی فنی دارد. در مقابل، ماندن در معماری فعلی کم‌هزینه‌تر ولی با فرصت‌های محدودتر است. استفاده از Soroban نیز حد وسطی است که برخی قابلیت‌های برنامه‌پذیری را می‌دهد ولی جامعه و ابزار کوچکتری دارد. تصمیم نهایی باید بر اساس استراتژی بلندمدت انکر و شبکه گرفته شود: اگر هدف، بهره‌برداری سریع از قابلیت‌های DeFi موجود و پیوند با دنیای اتریوم است، پروژه EVM می‌تواند مزیت راهبردی ایجاد کند. اما اگر هدف، کمینه کردن ریسک و همسویی با مسیر رسمی استلار است، شاید سرمایه‌گذاری روی Soroban انتخاب بهتری برای میان‌مدت باشد. در هر صورت، این گزارش نشان داد که امکان فنی انجام پروژه EVM وجود دارد و حتی در دنیای بلاک‌چین غیر معمول هم نیست که یک شبکه غیر اتریومی به EVM روی آورد

ارجاعات

1. EVM vs Stellar Comparison - Dune Docs: https://docs.dune.com/data-catalog/stellar/evm_v_stellar
2. The importance of JSON-RPC in Ethereum - CoinsBench: https://coinsbench.com/the-art-of-communicating-with-blockchain-a-detailed-exploration-3448073b60b4
3. Ethereum - Ankr: https://www.ankr.com/docs/rpc-service/chains/chains-api/ethereum/
4. Learn About the Core Components and Architecture of the Stellar Network | Stellar Docs: https://developers.stellar.org/docs/learn/fundamentals/stellar-stack
5. Stellar sparks smart contract upgrade — and it's not an EVM - Blockworks: https://blockworks.co/news/stellar-soroban-smart-contract-update
6. A comparison of Ethereum clients:
7. https://blog.web3labs.com/a-comparison-of-ethereum-clients
8. GitHub - paulfears/StellarSnap: Stellar on Metamask: https://github.com/paulfears/StellarSnap
9. Understanding Velo: A Comprehensive Overview - Messari https://messari.io/report/understanding-velo-a-comprehensive-overview
10. GitHub - evmos/ethermint: Ethermint is a Cosmos SDK library for running scalable and interoperable EVM chains: https://github.com/evmos/ethermint
11. What is Stellar Blockchain? A Complete Guide for Beginners: https://www.leewayhertz.com/what-is-stellar-blockchain/
12. EVM vs Stellar Comparison - Dune Docs: https://docs.dune.com/data-catalog/stellar/evm_v_stellar

منابع

1. docs.dune
2. coinsbench
3. ankr
4. developers.stellar
5. blockworks
6. blog.web3labs
7. github
8. messari
9. leewayhertz