Hydra: Kênh trạng thái đẳng cấu nhanh

Manuel MT Chakravarty[1] , Sandro Coretti[1], Matthias Fitzi[1], Peter Gazi[1], Philipp Kant[1], Aggelos Kiayias[2] và Alexander Russell[3]

[1]IOHK. firstname.lastname@iohk.io .

[2]Đại học Edinburgh và IOHK. akiayias@inf.ed.ac.uk .

[3]Đại học Connecticut và IOHK. acr@cse.uconn.edu .

Tóm lược

Các kênh trạng thái (State Channel) là một giải pháp Layer 2 hấp dẫn để cải thiện thông lượng và độ trễ của các Blockchain. Chúng cung cấp giải quyết thanh toán Off-Chain lạc quan và sự phát triển Off-Chain nhanh chóng của các hợp đồng thông minh giữa nhiều bên mà không đặt ra bất kỳ giả định bổ sung nào ngoài các giả định của Blockchain cơ bản. Trong trường hợp có tranh chấp hoặc nếu một bên không phản hồi, bằng chứng mật mã được thu thập trong kênh Off-Chain sẽ được sử dụng để giải quyết trạng thái được xác nhận cuối cùng trên chuỗi, để các hợp đồng đang thực hiện có thể được tiếp tục theo sự đồng thuận của chuỗi chính.

Một nhược điểm nghiêm trọng có trong các giao thức kênh trạng thái Layer 2 hiện tại là cơ sở hạ tầng hợp đồng thông minh Layer 1 hiện có và mã Code hợp đồng không thể được sử dụng lại ngoài chuỗi mà không cần thay đổi.

Trong bài nghiên cứu này, chúng tôi giới thiệu về *Hydra*, một kênh trạng thái đa bên *đẳng cấu*. Hydra đơn giản hóa giao thức Off-Chain và phát triển hợp đồng thông minh bằng cách áp dụng trực tiếp hệ thống hợp đồng thông minh Layer 1, theo cách này cho phép cùng một mã Code được sử dụng cả On-Chain và Off-Chain. Tận dụng *mô hình UTxO mở rộng*, chúng tôi phát triển một giao thức ngoài chuỗi nhanh chóng để phát triển các Hydra *Head* (các kênh trạng thái đẳng cấu của chúng tôi) có độ phức tạp vòng nhỏ hơn tất cả các đề xuất trước đó và cho phép xử lý kênh trạng thái tiến bộ theo yêu cầu, đồng thời và không đồng bộ.

Chúng tôi thiết lập các thuộc tính bảo mật mạnh mẽ cho giao thức. Chúng tôi trình bày và đánh giá các kết quả mô phỏng mở rộng chứng minh rằng Hydra tiếp cận các giới hạn vật lý của mạng về thời gian và thông lượng xác nhận giao dịch trong khi vẫn giữ các yêu cầu lưu trữ ở mức thấp nhất có thể. Cuối cùng, phương pháp thử nghiệm của chúng tôi có thể được quan tâm độc lập trong bối cảnh chung của việc đánh giá các giao thức đồng thuận.

1. Giới thiệu

Các giao thức sổ cái phân tán không được phép gặp phải những hạn chế nghiêm trọng về khả năng mở rộng, bao gồm độ trễ giao dịch cao (thời gian cần thiết để giải quyết một giao dịch), thông lượng thấp (số lượng giao dịch có thể được giải quyết trên một đơn vị thời gian) và bộ nhớ quá mức cần thiết để duy trì trạng thái hệ thống và lịch sử giao dịch của nó, có thể ngày càng gia tăng.

Một số giải pháp đã được đề xuất để giảm thiểu những vấn đề này bằng cách điều chỉnh các chi tiết của các giao thức sổ cái cơ bản. Sự thích ứng trực tiếp như vậy để có khả năng mở rộng thường được gọi là các giải pháp *Layer 1*.

Tuy nhiên, các giải pháp Layer 1 phải đối mặt với một hạn chế cố hữu, vì việc dàn xếp vẫn là một quá trình cồng kềnh bao gồm sự tham gia của một nhóm lớn, những người tham gia năng động và yêu cầu trao đổi lượng dữ liệu đáng kể. Một giải pháp tiếp cận khác để cải thiện khả năng mở rộng, mà chúng tôi nhấn mạnh trong công việc này là các giải pháp *Layer 2* (đôi khi còn được gọi là *Off-Chain*) phủ một giao thức mới lên trên Blockchain (Layer 1). Các giải pháp Layer 2 cho phép các bên chuyển tiền một cách an toàn từ Blockchain sang một phiên bản giao thức Off-Chain, giải quyết các giao dịch trong trường hợp này (gần như) một cách độc lập với chuỗi cơ sở và chuyển tiền một cách an toàn trở lại chuỗi cơ sở nếu cần.

Các giải pháp Off-Chain có lợi thế là chúng không yêu cầu các giả định tin cậy bổ sung về tính trung thực của các bên, ngoài các bên của Blockchain cơ bản. Chúng có thể rất hiệu quả trong trường hợp lạc quan khi tất cả những người tham gia trong phiên bản giao thức Off-Chain hoạt động như mong đợi. Cụ thể, một trường hợp như vậy hoạt động giữa một số ít các bên liên lạc trực tiếp với nhau và theo cách cho phép họ quên đi các giao dịch gần đây ngay khi họ cập nhật (và bảo mật) trạng thái cục bộ của họ.

Giải pháp khả năng mở rộng Off-Chain nổi bật nhất là khái niệm kênh thanh toán (Payment Channel) [9, 33, 16]. Một kênh thanh toán được thiết lập giữa hai bên, cho phép họ thanh toán tiền qua lại trên kênh này; trong trường hợp lạc quan, điều này có thể diễn ra mà không cần thông báo cho giao thức Layer 1. Các kênh thanh toán đã được mở rộng cho các mạng lưới kênh thanh toán, ví dụ: Mạng Bitcoin Lightning [33]. Về nguyên tắc, các mạng như vậy cho phép chuyển tiền ngoài chuỗi giữa bất kỳ bên nào được kết nối thông qua một đường dẫn của các kênh thanh toán.

Một điểm hạn chế là trong mạng lưới kênh thanh toán truyền thống, giao dịch giữa hai bên không chia sẻ kênh thanh toán trực tiếp đòi hỏi sự tương tác giữa tất cả các bên trên đường dẫn kênh thanh toán giữa họ (được gọi là *trung gian*), ngay cả trong trường hợp lạc quan. Các kênh thanh toán *ảo*, ví dụ như Perun [19], giải quyết vấn đề này và không yêu cầu tương tác với các bên trung gian (trong trường hợp lạc quan).

Các kênh trạng thái [5] mở rộng khái niệm kênh thanh toán sang các trạng thái để hỗ trợ các hợp đồng thông minh. Các mạng lưới kênh trạng thái [21, 15, 29] cũng mở rộng khái niệm kênh trạng thái thành mạng lưới (tương tự với phần mở rộng mạng đã thảo luận ở trên). Tuy nhiên, các mạng này chỉ cho phép thiết lập các kênh trạng thái theo cặp qua mạng.

Các kênh trạng thái đa bên đã được giới thiệu trong [31] cùng với mô tả cấp cao của một giao thức tương ứng. Một kênh trạng thái đa bên cho phép một nhóm các bên duy trì trạng thái "chung", trong đó họ có thể tính toán mà không cần tương tác với Blockchain (trong trường hợp lạc quan).

Trong [18], khái niệm về kênh trạng thái *ảo* đa bên đã được đưa ra, kênh trạng thái giữa nhiều bên có thể được thiết lập mà không cần tương tác Blockchain (nó được cho rằng đã tồn tại một biểu đồ kết nối của các kênh trạng thái theo cặp giữa các bên); và một giao thức tương ứng đã được trình bày.

Bất chấp những tiến bộ đáng kể ở trên, những thách thức lớn vẫn còn tồn tại, cả về thiết lập hiệu suất xử lý Off-Chain cao xấp xỉ giới hạn vật lý của mạng cơ sở cũng như về ý nghĩa áp đặt chi phí kỹ thuật và khái niệm đáng kể lên Layer 1 như trạng thái hợp đồng Off-Chain phải được xác minh trong một đại diện không phải là nguyên mẫu (Non-native); lý do là trạng thái của các hợp đồng được phát triển trong một kênh trạng thái cụ thể cần phải được tách biệt và thể hiện ở một hình thức cho phép nó được xử lý bởi cả Off-Chain và hệ thống tập lệnh hợp đồng thông minh On-Chain trong trường hợp xảy ra tranh chấp Off-Chain. Điều này dẫn đến các thiết kế trong đó các phép tính được thực hiện ngoài chuỗi không còn nằm trong đại diện được sử dụng bởi chính sổ cái; tức là họ không phải là nguyên mẫu. Ví dụ, hợp đồng Solidity mẫu [31] của tuần tự hóa trạng thái thành một mảng Byte32. Bản thân các hợp đồng thông minh cần được điều chỉnh tương ứng. Nói cách khác, hệ thống tập lệnh của sổ cái và của các kênh trạng thái gắn liền với sổ cái khác nhau một cách đáng kể, áp đặt một cách hiệu quả hai hệ thống tập lệnh riêng biệt.

Hydra. Trong Hydra, chúng tôi giải quyết cả hai vấn đề, hiệu suất xử lý Off-Chain và biểu thị trạng thái, với việc giới thiệu các kênh trạng thái đa bên *đẳng cấu* (Isomorphic). Đây là các kênh trạng thái có khả năng nhanh chóng sử dụng lại biểu thị trạng thái chính xác của sổ cái cơ bản và do đó, kế thừa hệ thống tập lệnh của sổ cái. Do đó, các kênh trạng thái mang lại hiệu quả các anh chị em sổ cái song song ngoài chuỗi, mà chúng tôi gọi là *các Head* — sổ cái sẽ có nhiều Head. Việc thành lập một Head mới tuân theo một kế hoạch cam kết tương tự như phổ biến ở các kênh trạng thái. Tuy nhiên, khi kênh trạng thái bị đóng, do hợp tác hoặc do tranh chấp, trạng thái của Head sẽ được ghi nhận ngay lập tức vào trạng thái sổ cái cơ bản và mã Code hợp đồng

thông minh đã được sử dụng Off-Chain bây giờ sẽ được sử dụng On-Chain. Điều này có thể thực hiện được, ngay cả khi không cần đăng ký trước các hợp đồng được sử dụng trong Head, vì một và cùng một mã biểu thị trạng thái và hợp đồng (nhị phân) được sử dụng Off-Chain và On-Chain.

Không phải mọi hệ thống tập lệnh Blockchain đều có lợi cho các kênh trạng thái đẳng cấu. Việc xây dựng chúng đòi hỏi phải tạo ra một cách hiệu quả các khối trạng thái Blockchain tùy ý, xử lý chúng một cách độc lập và có thể hợp nhất chúng trở lại một cách hiệu quả vào bất kỳ lúc nào. Chúng tôi nhận thấy rằng mô hình sổ cái UTxO kiểu Bitcoin [6, 34] đặc biệt phù hợp với sự biểu thị thống nhất của trạng thái On-Chain và Off-Chain, đồng thời hứa hẹn tăng tính song song trong xử lý giao dịch bên trong các kênh trạng thái đa bên. Mặc dù hạn chế chính của mô hình UTxO thuần túy theo truyền thống là khả năng viết tập lệnh hạn chế, sự ra đời của *mô hình UTxO mở rộng (EUTxO)* [13] đã gỡ bỏ hạn chế này và hỗ trợ cho các máy trạng thái chung. Các mô hình UTxO mở rộng tạo cơ sở cho các nền tảng hợp đồng thông minh của các Blockchain hiện có, chẳng hạn như Cardano [14] và Ergo [17]; do đó, công việc được trình bày trong bài nghiên cứu này cũng sẽ có liên quan thực tế ngay lập tức.

Cũng giống như cách biểu thị sổ cái UTxO, cách biểu thị sổ cái EUTxO làm cho tất cả các phụ thuộc dữ liệu một cách rõ ràng mà không đưa ra *các phụ thuộc sai*, nói cách khác, hai giao dịch chỉ phụ thuộc trực tiếp hoặc gián tiếp vào nhau nếu có sự phụ thuộc dữ liệu thực tế giữa chúng. Điều này tránh việc tuần tự hóa quá mức các hệ thống phụ thuộc vào trạng thái toàn cục. Do đó, độ dài của đường dẫn dài nhất qua biểu đồ EUTxO trùng với *độ phức tạp về chiều sâu* của khối lượng công việc do quá trình xử lý và xác thực giao dịch kéo theo. Đây là mức tối ưu khi có liên quan đến xử lý giao dịch song song [10].

Khai thác biểu thị sổ cái EUTxO, chúng tôi có thể thiết kế một giao thức Off-Chain với hiệu suất không song song. Đặc biệt, giao thức Hydra Head có khả năng xử lý Off-Chain đồng thời và không đồng bộ giữa các thành viên khác nhau của Head, chỉ sử dụng 3 vòng tương tác để cập nhật. Ngược lại, các hoạt động trước đây trong các kênh trạng thái đa bên yêu cầu một hoạt động đồng bộ hoặc áp đặt 4 vòng để tạo điều kiện thuận lợi cho việc tuần tự hóa các đầu vào và tổ chức trạng thái Off-Chain.

Chi tiết hơn trong Hydra, một tập hợp các bên phối hợp để *cam kết* một tập hợp các UTxO (thuộc sở hữu của các bên) vào một giao thức Off-Chain, được gọi là *giao thức Head*. Bộ UTxO đó tạo thành trạng thái nguyên thủy ban đầu, sau đó các bên có thể phát triển bằng cách xử lý các hợp đồng thông minh và giao dịch với nhau mà không cần tương tác Blockchain trong trường hợp lạc quan.

Do tính chất đẳng cấu của các Hydra Head, việc xác thực giao dịch, bao gồm cả việc thực thi tập lệnh, diễn ra theo các quy tắc giống như On-Chain. Trên thực tế, có thể sử dụng cùng một mã Code xác nhận. Quan điểm này cho rằng ngữ nghĩa của On-Chain và Off-Chain là trùng khớp, dẫn đến việc đơn giản hóa kỹ thuật đáng kể. Trong trường hợp có tranh chấp hoặc trong trường hợp một số bên muốn chấm dứt giao thức Off-Chain, các bên *ngừng cam kết* trạng thái hiện tại của Head để chuyển trở lại Blockchain. Cuối cùng, việc ngừng cam kết sẽ dẫn đến trạng thái Blockchain được cập nhật phù hợp với sự phát triển của giao thức Off-Chain trên bộ UTxO đã cam kết ban đầu. Để giảm chi phí chuỗi chính, chuỗi chính không biết đến lịch sử giao dịch chi tiết của giao thức Head dẫn đến trạng thái cập nhật. Điều quan trọng, thời gian cần thiết để ngừng cam kết không phụ thuộc vào số lượng các bên tham gia vào một Head hoặc quy mô của Head. Hơn nữa, quá trình ngừng cam kết được thiết kế sao cho khi trạng thái mới nhất trong Head rất lớn, trạng thái của Head có thể được ngừng cam kết các phần nhỏ (nhưng song song). Cuối cùng, Hydra cho phép các cam kết và ngừng cam kết tăng dần, tức là các UTxO có thể được thêm vào và loại bỏ khỏi Head đang chạy mà không cần đóng nó.

Mạng lưới Cross-Head. Trong bài nghiên cứu này, chúng tôi chỉ tập trung vào việc phân tích giao thức Hydra Head; tuy nhiên, sự tồn tại của nhiều Head chồng chéo một phần ra khỏi chuỗi chính có thể làm phát sinh giao tiếp Cross-Head (như trong Lightning Network [33]), sử dụng các kỹ thuật tương tự như [21, 18].

Yêu cầu tham gia trực tuyến. Giao thức Hydra Head hướng đến kịch bản mà những người tham gia được yêu cầu xác thực giao dịch trực tuyến và đáp ứng. Chẳng hạn như [33], việc ở chế độ ngoại tuyến sẽ ngăn cản tiến trình, cũng như việc tham gia giải quyết tranh chấp trên chuỗi tiềm năng. Kịch bản mà một số bên thường xuyên ngoại tuyến cũng được quan tâm nhưng không thuộc phạm vi của phiên bản hiện tại.

Phương pháp đánh giá hiệu quả hoạt động và kết quả thực nghiệm. Vì hiệu suất xử lý giao dịch là động lực cơ bản cho các giao thức Layer 2, các thuộc tính này của giao thức Hydra là đặc biệt quan trọng để thiết lập. Mặc dù giao dịch trên giây (TPS) là một con số đáng giá ngay lập tức đối với các hệ thống đã triển khai, nó rất nhạy cảm với những thay đổi cơ bản về phần cứng hoặc mạng lưới; đặc biệt, nó là một phương tiện không tin cậy để so sánh thực nghiệm các đề xuất thuật toán khác nhau trừ khi các thử nghiệm sao chép chính xác môi trường máy tính cũng nhạy cảm với đầu vào của người dùng. Để tránh những khó khăn này và đoán trước các tình huống sử dụng cụ thể, chúng tôi áp dụng phương pháp "đường cơ sở tương đối" (Baseline Relative) để thiết lập các đảm bảo về hiệu suất, chứng tỏ rằng Hydra đạt được hiệu suất đạt được mức tối ưu về mặt lý thuyết cho *bất kỳ giao thức đồng thuận nào*. Kết quả thử nghiệm của chúng tôi

thu được bằng mô phỏng, điều này cũng cho phép khám phá độ chính xác cao về các lựa chọn thiết kế cụ thể được áp dụng bởi giao thức Hydra. Chúng ta hãy xem xét hai loại đường cơ sở chính được xây dựng dưới đây.

Đường cơ sở chung (Universal Baseline). Như đã đề cập ở trên, chúng tôi bắt đầu bằng cách xem xét một đường cơ sở phản ánh các nghĩa vụ yếu nhất của bất kỳ thuật toán đồng thuận nào. Cụ thể, đường cơ sở chung chỉ xem xét chi phí xử lý mỗi giao dịch và lan truyền các giao dịch trên toàn mạng lưới; quan sát thấy rằng bất kỳ thuật toán đồng thuận lặp lại nào mang lại trạng thái đầy đủ tại mỗi Node nhất thiết phải thực hiện cả hai hoạt động. Chúng tôi chứng minh rằng Hydra đạt được hiệu quả mà các đối thủ thậm chí là lý tưởng cho hầu hết các tình huống. Vì đường cơ sở không phụ thuộc vào giao thức này là đường cơ sở mà bất kỳ thuật toán đồng thuận lặp đi lặp lại nào cũng có thể được so sánh, nên mức độ tối ưu gần như tối ưu đối với đường cơ sở này ngầm chứng minh rằng Hydra có khả năng cạnh tranh với bất kỳ lớp đồng thuận nào khác. Trong các thử nghiệm của mình, chúng tôi so sánh Hydra với đường cơ sở chung cho một số trường hợp khác nhau phản ánh hành vi của người dùng.

Đường cơ sở không giới hạn (Unlimited Baseline). Đường cơ sở thứ hai tập trung vào các đặc điểm của chính giao thức. Đặc biệt, nó yêu cầu làm thế nào việc triển khai giao thức so với việc thực thi giao thức được lý tưởng hóa bởi một tập hợp các Node không có tranh chấp cục bộ về tài nguyên. So sánh đường cơ sở này có nghĩa là bổ sung cho đường cơ sở chung và giúp trả lời câu hỏi sau. Bất cứ khi nào có sự khác biệt giữa đường cơ sở chung và việc thực thi giao thức đồng thuận thực tế trong thử nghiệm, thì mức độ phân kỳ này sẽ được quy vào chi phí vốn có của việc chạy giao thức đồng thuận so với chi phí phát sinh do tranh giành tài nguyên trong mỗi Node. Ngay cả những thiết kế giao thức đồng thuận tốt cũng được kỳ vọng sẽ khác xa với đường cơ sở chung: xét cho cùng, cơ chế đồng thuận là một vấn đề khó giải quyết. Tuy nhiên, các thiết kế giao thức tốt luôn phải gần đúng với đường cơ sở không giới hạn của chúng. Trong các thử nghiệm của mình, chúng tôi chứng minh rằng đây là trường hợp của Hydra trong tất cả các tình huống khác nhau của thiết lập thử nghiệm của chúng tôi.

Kết quả thực nghiệm. Chúng tôi đã tiến hành mô phỏng chi tiết hiệu suất của Head trong nhiều tải và mạng lưới khác nhau, bao gồm cả Head được bản địa hóa theo địa lý và Head với những người tham gia trải dài trên nhiều lục địa, gây ra sự chậm trễ mạng lớn. Chúng tôi nhận thấy rằng giao thức Head trong trường hợp lạc quan đạt được tiến bộ cạnh tranh với tốc độ và thông lượng của mạng trong tất cả các cấu hình; điều này được hỗ trợ bởi sự đồng thời được cung cấp bởi thứ tự giao dịch chỉ một phần được cho phép bởi cấu trúc đồ thị sổ cái UTxO cơ bản.

So sánh với công việc trước đây. Một số công trình trước đây nghiên cứu về giao thức kênh trạng thái. Giao thức của Miller et al. [31] cho phép một nhóm các bên khởi tạo một phiên bản hợp đồng thông minh (trạng thái) trên chuỗi và đưa nó ra ngoài chuỗi. Trạng thái sau đó có thể được phát triển ngoài chuỗi mà không cần tương tác chuỗi trong trường hợp hoàn toàn trung thực. Bằng cách xử lý đồng thời các tranh chấp trong một hợp đồng chia sẻ, việc giải quyết tranh chấp vẫn trong thời gian $O(\Delta)$, trong đó Δ là thời gian giải quyết cho một giao dịch On-Chain. Giao thức Off-Chain tiến hành theo các giai đoạn của 4 vòng không đồng bộ trong đó một người lãnh đạo điều phối việc xác nhận các giao dịch mới giữa những người tham gia trong giao thức Off-Chain. Tương tự như Hydra, giao thức cho phép thêm / bớt tiền khỏi hợp đồng Off-Chain trong khi nó đang chạy.

Giao thức của Dziembowski et al. [18] dựa trên các kênh trạng thái theo cặp và cho phép khởi tạo kênh trạng thái đa bên giữa bất kỳ nhóm các bên nào được kết nối bằng đường dẫn của các kênh trạng thái theo cặp, việc khởi tạo kênh đa bên không yêu cầu bất kỳ tương tác nào với chuỗi chính. Giao thức Off-Chain tiến hành theo các giai đoạn của 4 vòng đồng bộ để xác nhận các giao dịch mới mà không cần người lãnh đạo điều phối.

Giao thức Off-Chain Hydra không đồng bộ hoàn toàn; trong trường hợp lạc quan, các giao dịch được xác nhận trong 3 vòng (không đồng bộ) độc lập với nhau và không cần phải có người lãnh đạo. Một người lãnh đạo chỉ được yêu cầu để giải quyết các xung đột giao dịch và để "thu thập rác" định kỳ cho phép giao thức duy trì kích thước trạng thái độc lập với kích thước của lịch sử giao dịch.

So với các giải pháp trước đây được trích dẫn ở trên, Hydra cung cấp thời gian xác nhận nhanh hơn trong giao thức Off-Chain; đây là một lợi thế được kích hoạt bởi tổ chức cấu trúc của các giao dịch trong mô hình EUTxO, trong khi các giao thức trước đó bị cản trở bởi một tổ chức trạng thái nguyên khối. Một lợi thế bổ sung so với [31] và [18] là những điều đó cố định tập hợp các hợp đồng có thể được phát triển trong một kênh trạng thái nhất định tại thời điểm tạo kênh; Hydra không yêu cầu cam kết trước như vậy: các hợp đồng mới có thể được giới thiệu trong Head sau khi khởi tạo bằng ngôn ngữ EUTxO gốc của Blockchain cơ bản. Một sự khác biệt đáng kể khác đối với [18] là giao thức của họ kêu gọi các bên khóa tiền trên chuỗi chính thay mặt cho các bên khác, điều đó gây ra bởi sự bất đối xứng gây ra bởi thành phần dọc theo đường dẫn của các kênh trạng thái theo cặp, trong khi trong Hydra cũng như trong [31], các bên chỉ cần khóa tiền thay mặt cho chính mình. Cuối cùng, Hydra là đẳng cấu và do đó sử dụng lại hệ thống hợp đồng thông minh và mã Code hiện có để tính toán Off-Chain. Đây không phải là trường hợp của [31] và [18]. Ví dụ: nếu chúng ta xem xét hợp đồng

Solidity mẫu của [31], nó sẽ phải triển khai một máy trạng thái có khả năng thực thi mã Bytecode EVM để đạt được việc tái sử dụng hợp đồng (hệ thống) và do đó, các kênh trạng thái đẳng cấu.

Chúng tôi lưu ý rằng cũng có một số lượng lớn các đề xuất được chưa được đánh giá ngang hàng cho các giải pháp dựa trên kênh trạng thái như [28, 15, 29, 3]. Những đề xuất này đi kèm với nhiều mức độ đặc tả chính thức và đảm bảo an ninh có thể chứng minh được và việc hệ thống hóa chúng nằm ngoài phạm vi hiện tại của chúng tôi; quan sát thấy rằng không ai trong số chúng cung cấp thuộc tính đẳng cấu hoặc đi kèm với một phân tích bảo mật chính thức hoàn chỉnh và một đánh giá thử nghiệm.

Hai khái niệm có liên quan nhưng khác biệt với các kênh trạng thái là *Sidechain* (ví dụ [7, 24, 26]) và *Non-custodial Chain* (ví dụ [32, 27, 20, 4]), bao gồm Plasma và Rollup. Sidechain cho phép chuyển các tài sản giữa một chuỗi chính và một Sidechain thông qua cơ chế Pegging, với chuỗi chính được bảo vệ khỏi các lỗi bảo mật Sidechain bởi một "thuộc tính tường lửa"; trái ngược với kênh trạng thái, Sidechain có cơ chế đồng thuận riêng, tiền có thể bị mất trong trường hợp bảo mật của Sidechain bị sụp đổ. Mặt khác, Non-custodial Chain ủy quyền xử lý giao dịch chuỗi chính cho một trình tổng hợp không cần tin cậy và có khả năng như trong các kênh trạng thái để bảo vệ khỏi lỗi bảo mật. Tuy nhiên, trình tổng hợp là một điểm lỗi duy nhất và sự hỏng hóc của nó, trong bối cảnh mà một lượng lớn người dùng được phục vụ bởi cùng một Non-custodial Chain, dẫn đến vấn đề "thoát hàng loạt" (Mass-Exit) (xem ví dụ: [20]); lưu ý rằng, các kênh trạng thái có thể mở rộng đến một số lượng lớn người dùng thông qua mạng lưới kênh trạng thái [21] mà không yêu cầu nhiều người dùng trên mỗi kênh. Cuối cùng, chúng tôi lưu ý rằng công việc đang được tiến hành trên các bản sao lưu lạc quan, được báo cáo trong [4], tuyên bố một tính năng tương tự như thuộc tính đẳng cấu của chúng tôi, tuy nhiên, không có lợi ích về độ trễ trong cách tiếp cận của chúng tôi vì việc giải quyết của chúng vẫn tiến triển với chuỗi chính cơ bản.

2. Sơ bộ

2.1. Đa chữ ký

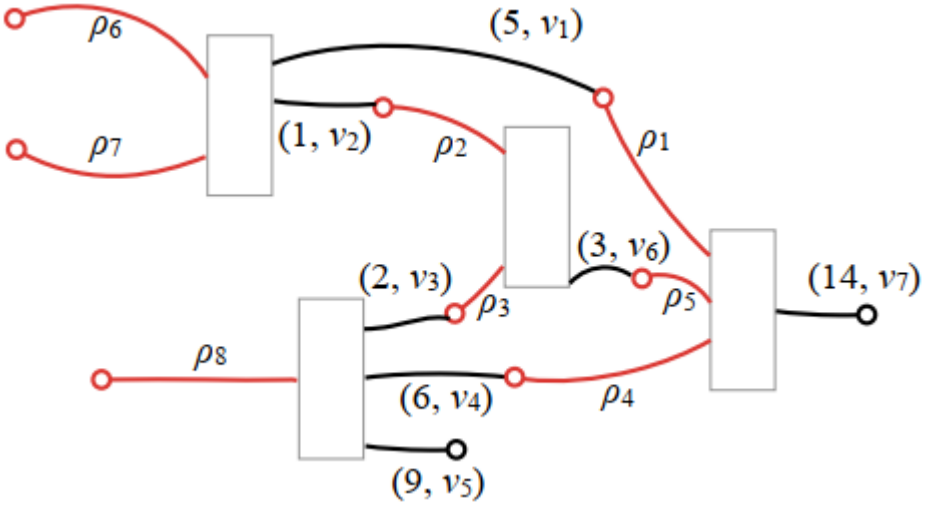
Lược đồ đa chữ ký [25, 30] là một bộ thuật toán **MS** = (**MS-Setup**, **MS-KG**, **MS-AVK**, **MS-Sign**, **MS-ASig**, **MS-Verify**) sao cho $\Pi \leftarrow$ **MS-Setup** (1^k) tạo ra các tham số công khai; với những điều này, (**vk**, **sk**) ← **MS-KG** (Π) có thể được sử dụng để tạo các cặp khóa mới. Sau đó

- $\sigma \leftarrow$ **MS-Sign** (Π, **sk**, *m*) ký một thông điệp *m* bằng cách sử dụng khoá **sk**;
- $\tilde{\sigma} \leftarrow$ **MS-ASig** (Π, *m*, *V*, *S*) tập hợp *S* các chữ ký thành một chữ ký tổng hợp, duy nhất $\tilde{\sigma}$.

Thuật toán **avk** ← **MS-AVK** (Π, *V*) tổng hợp một bộ *V* khóa xác minh **vk** thành một, khóa xác minh tổng hợp **avk** có thể được sử dụng để xác minh: **MS-Verify** (Π, **avk**, *m*, $\tilde{\sigma}$) ∈ {True, False} xác minh chữ ký tổng hợp trong khóa xác minh tổng hợp. Trong phần sau, chúng tôi thường đặt tham số Π ẩn trong các lệnh gọi hàm để dễ đọc hơn.
Về mặt trực quan, tính bảo mật của lược đồ đa chữ ký đảm bảo rằng, nếu **avk** được tạo ra từ nhiều khóa xác minh *V* thông qua **MS-AVK**, thì không có chữ ký tổng hợp $\tilde{\sigma}$ nào có thể vượt qua xác minh **MS-Verify** (**avk**, *m*, $\tilde{\sigma}$) trừ khi tất cả các bên trung thực đều giữ khóa trong *V* ký *m*. Một nghiên cứu đầy đủ xuất hiện trong Phụ lục A.

2.2. Mô hình UTxO mở rộng và máy trạng thái (State Machine)

Cơ sở cho các kênh trạng thái đẳng cấu nhanh của chúng tôi là mô hình sổ cái UTxO của Bitcoin [6, 34]. Nó sắp xếp các giao dịch theo cấu trúc đồ thị vòng có hướng, do đó làm cho tính song song có sẵn rõ ràng: bất kỳ hai giao dịch nào không phụ thuộc trực tiếp hoặc gián tiếp vào nhau đều có thể được xử lý độc lập.



Hình 1: Ví dụ về biểu đồ UTxO đơn giản

UTxO. Các giao dịch trong sổ cái UTxO chứa một tập hợp các đầu vào và đầu ra, trong đó các đầu ra khóa một lượng Crypto, sao cho chỉ những đầu vào được ủy quyền của các giao dịch tiếp theo mới có thể kết nối và chi tiêu những khoản tiền đó. Sự sắp xếp này dẫn đến các đồ thị, chẳng hạn như đồ thị trong Hình 1, trong đó các hộp biểu thị các giao dịch với đầu vào (màu đỏ) ở bên trái và đầu ra (màu đen) ở bên phải.

Mỗi đầu ra khóa một lượng Crypto, có thể được chuyển qua một giao dịch tiếp theo bằng cách sử dụng đầu ra đó với đầu vào mới. Tập hợp các đầu ra treo lơ lửng (không kết nối) là các *đầu ra giao dịch chưa chi tiêu (UTxO)* - có 2 UTxO trong Hình 1. Ngoài

tiền tệ bị khóa, mỗi đầu ra cũng đi kèm với một vị từ *v*, được gọi là *trình xác thực (Validator)*. Trong Hình 1, chúng tôi sử dụng các cặp (*n, v*) để chỉ ra rằng một đầu ra nhất định sẽ khóa *n Crypto* tử với vị từ trình xác thực *v*.

Trong đó các đầu ra mang trình xác thực, mỗi đầu vào đi kèm với một giá trị *Redeemer ρ*. Để xác định xem một đầu vào nhất định của giao dịch *tx* hiện đã được xác thực có được phép kết nối với một đầu ra chưa chi tiêu hay không, chúng tôi xác định xem liệu vị từ trình xác thực *v* của đầu ra đó có áp dụng cho Redeemer *ρ* hay không; hoặc chính thức hơn, chúng tôi kiểm tra $v(\rho, \sigma)$ = True, trong đó *ngữ cảnh xác thực σ* đại diện cho một số thuộc tính của giao dịch mà đầu vào chi tiêu thuộc về, chẳng hạn như giá trị Hash mật mã của giao dịch. Ví dụ: trình xác thực có thể yêu cầu Redeemer có chữ ký trên Hash giao dịch có trong ngữ cảnh *σ* cho một cặp khóa cụ thể, sao cho chỉ chủ sở hữu của khóa cá nhân mới có thể sử dụng đầu ra bị khóa bởi trình xác thực đó.

UTxO mở rộng. *Mô hình UTxO mở rộng (EUTxO)* [13] duy trì cấu trúc này, đồng thời bổ sung hỗ trợ cho các hợp đồng thông minh rõ ràng hơn, đặc biệt là cho các máy trạng thái đa giao dịch, làm cơ sở cho phần chuỗi chính của công việc được trình bày ở đây. Khả năng biểu đạt bổ sung này đạt được nhờ hai thay đổi đối với sơ đồ UTxO đơn giản được nêu trước đây:

- Đầu ra mang theo, ngoài giá trị Crypto *n* và trình xác thực *v*, giờ đây còn là *Datum δ*, trong số những thứ khác, có thể được sử dụng để duy trì trạng thái của các hợp đồng thông minh đang hoạt động lâu dài.
- Ngữ cảnh xác thực *σ* được mở rộng để chứa toàn bộ giao dịch *tx* đã được xác thực cũng như các UTxO được sử dụng bởi các đầu vào của giao dịch đó.

Trong mô hình mở rộng này, việc đánh giá vị từ trình xác thực ngụ ý kiểm tra $v(\rho, \delta, \sigma)$ = True. Bên cạnh việc duy trì trạng thái hợp đồng trong *δ*, thực tế là trình xác thực có thể kiểm tra toàn bộ giao dịch *tx* đã được xác thực thông qua *σ* cho phép trình xác thực thực thi các hợp đồng bất biến được duy trì trên toàn bộ chuỗi giao dịch.

Mặc dù các kết quả chính thức về EUTxO còn khá gần đây, nhưng các mô hình UTxO mở rộng đã tạo cơ sở cho các nền tảng hợp đồng thông minh của các Blockchain hiện có - đặc biệt là Cardano [14] và Ergo [17]. Do đó, giao thức Hydra Head như được trình bày trong bài nghiên cứu này có liên quan thực tế tức thì đối với các hệ thống hiện có này.

Token do người dùng xác định. Ngoài phần mở rộng EUTxO cơ bản, chúng tôi tổng quát hóa các *giá trị* tiền tệ được ghi trên sổ cái từ các số tích phân thành *Token tổng*

quát do người dùng xác định [1]. Nói một cách đơn giản (đủ để hiểu các khái niệm trong bài nghiên cứu này), các giá trị là các bộ theo dõi có bao nhiêu đơn vị Token của loại tiền tệ nào có sẵn. Ví dụ: giá trị $\{\mathbf{Coin} \to \{\mathbf{Coin} \to 3\}, c \to \{t_1 \to 1, t_2 \to 1\}\}$ chứa 3 đồng **Coin** (chỉ có một Token (có thể thay thế) **Coin** để thanh toán tiền tệ **Coin**), cũng như các Token (không thể thay thế) t_1 và t_2, cả hai đều là tiền tệ c. Các giá trị có thể được thêm vào một cách tự nhiên, ví dụ:

$$\{\mathbf{Coin} \to \{\mathbf{Coin} \to 3\}, c \to \{ t_1 \to 1 , t_2 \to 1\}\}$$
$$+ \{\mathbf{Coin} \to \{\mathbf{Coin} \to 1\}, c \to \{ t_3 \to 1\}\}$$
$$= \{\mathbf{Coin} \to \{\mathbf{Coin} \to 4\}, c \to \{ t_1 \to 1, t_2 \to 1, t_3 \to 1\}\}.$$

Trong phần sau, $\emptyset$ là giá trị trống và $\{t_1, ..., t_n\} :: c$ được dùng làm cách viết tắt của $\{c \to \{t_1 \to 1 , ..., t_n \to 1\}\}$.

Sổ cái EUTxO bao gồm các *giao dịch*: Các giao dịch là ngũ phân vị $tx = (I, O, \mathbf{val}_{\text{Forge}}, r, K)$ bao gồm tập hợp *đầu vào I*, danh sách *đầu ra O*, giá trị của *Token giả mạo/đốt* $\mathbf{val}_{\text{Forge}}$, *khoảng cách Slot* $r = (r_{\min}, r_{\max})$ và một tập hợp các khóa công khai *K*. Mỗi đầu vào $i \in I$ là một cặp bao gồm *tham chiếu đầu ra* **out-ref** (bao gồm ID giao dịch và chỉ số xác định đầu ra trong giao dịch) và một *Redeemer* ρ (được sử dụng để cung cấp dữ liệu để xác nhận). Mỗi đầu ra $o \in O$ là một bộ ba ($\mathbf{val}, v, \delta$) bao gồm giá trị **val**, tập lệnh trình xác thực v và Datum δ. Khoảng cách Slot r cho biết các Slot trong đó tx có thể được xác nhận và cuối cùng, K là các khóa công khai mà tx được ký.

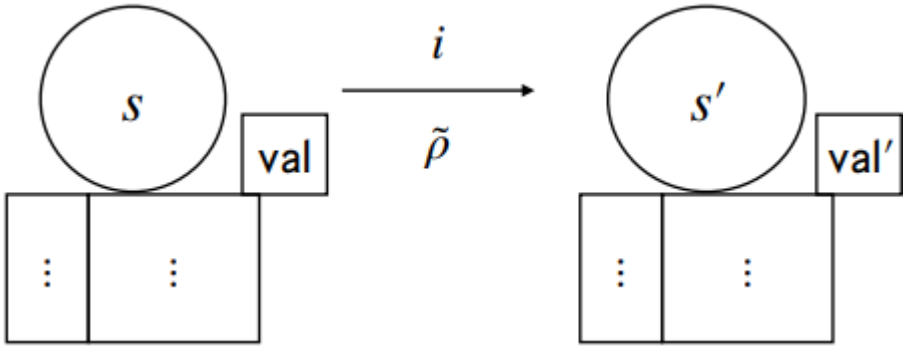
Để xác thực giao dịch tx với tập đầu vào I, đối với mỗi đầu ra $o = (\mathbf{val}, v, \delta)$ được tham chiếu bởi $i = (\mathbf{out\text{-}ref}, \rho) \in I$, trình xác thực tương ứng v được chạy trên các đầu vào sau: $v(\mathbf{val}, \delta, \rho, \sigma)$, trong đó ngữ cảnh xác nhận σ bao gồm tx và *tất cả* các đầu ra được tham chiếu bởi một số $i \in I$ (không chỉ o). Cuối cùng, tx là hợp lệ khi và chỉ khi tất cả các trình xác thực trả về True.

Máy trạng thái (State Machine). Một sự trừu tượng hóa thuận tiện cho các hợp đồng thông minh EUTxO bao gồm một chuỗi các giao dịch liên quan là các máy trạng thái. Cụ thể, chúng tôi áp dụng *các máy phát ra ràng buộc (CEMs)* [13]. Chúng dựa trên máy Mealy và bao gồm một tập hợp các trạng thái S_{CEM}, một tập hợp các đầu vào I_{CEM}, một vị từ $final_{CEM} : S_{CEM} \to Bool$ xác định các trạng thái cuối cùng, và một quan hệ bước $s \xrightarrow{i} (s', \text{tx}^{\equiv})$ đưa một trạng thái s trên đầu vào i đến trạng thái kế thừa s' theo các yêu cầu mà các ràng buộc $\text{tx}^{\equiv}$ được thỏa mãn).

Chúng tôi triển khai CEMs trên sổ cái EUTxO (chuỗi chính) bằng cách thể hiện một chuỗi các trạng thái CEM dưới dạng chuỗi giao dịch. Mỗi giao dịch này có *đầu vào*

máy trạng thái i_{CEM} và *đầu ra máy trạng thái* o_{CEM}, trong đó giao dịch này bị khóa bởi trình xác thực v_{CEM}, thực hiện quan hệ bước. Các ngoại lệ duy nhất là trạng thái đầu tiên và cuối cùng, không có đầu vào và đầu ra của máy trạng thái tương ứng.

Cụ thể hơn, với hai giao dịch tx và tx', chúng đại diện cho các trạng thái liên tiếp $s \xrightarrow{i} (s', \text{tx}^{\equiv})$



Hình 2: Các giao dịch thể hiện các trạng thái liên tiếp trong quan hệ chuyển tiếp CEM $s \xrightarrow{i} (s', \text{tx}^{\equiv})$. Trường val và val' là trường giá trị của đầu ra máy trạng thái và $\tilde{\rho}$ là dữ liệu bổ sung.

- Đầu ra của máy trạng thái o_{CEM} = (**val**, v_{CEM}, s) của tx được tiêu thụ bởi đầu vào máy trạng thái i'_{CEM} = (**out-ref**, ρ) của tx, có Redeemer $\rho = i$ (tức là Redeemer cung cấp đầu vào máy trạng thái) và
- Hoặc $final_{CEM}(s')$ = True và *tx'* không có đầu ra máy trạng thái, hoặc o'_{CEM} = (**val'**, v_{CEM}, s') và tx' đáp ứng tất cả các ràng buộc bởi $\text{tx}^{\equiv}$.

Đôi khi, sẽ hữu ích khi có thêm dữ liệu $\tilde{\rho}$ được cung cấp như một phần của Redeemer, tức là $\rho = (i, \tilde{\rho})$. Chuyển đổi trạng thái của kiểu được mô tả được thể hiện bằng hai giao dịch được kết nối như trong Hình 2. Để đơn giản, các đầu vào và đầu ra của máy trạng thái không được hiển thị, ngoại trừ các trường giá trị **val** và **val'** của đầu ra máy trạng thái.

3. Tổng quan về giao thức

Giao thức Hydra cung cấp chức năng khóa một tập hợp các UTxO trên một Blockchain, được gọi là *chuỗi chính* và phát triển nó bên trong một *Head* Off-Chain, độc lập với chuỗi chính. Tại bất kỳ thời điểm nào, Head có thể được đóng lại với kết quả là tập hợp các UTxO bị khóa trên chuỗi chính được thay thế bằng tập hợp các UTxO mới nhất bên trong Head. Giao thức đảm bảo duy trì toàn bộ tài sản: việc tạo thêm tiền không thể xảy

ra ngoài chuỗi và các bên trung thực tham gia Head không thể mất bất kỳ khoản tiền nào ngoài việc họ đồng ý cho chúng đi.

Ưu điểm của tiến hóa Head từ quan điểm hoạt động là, trong điều kiện tốt, về cơ bản nó có thể tiến hành ở tốc độ mạng lưới, do đó giảm độ trễ và tăng thông lượng một cách tối ưu. Đồng thời, giao thức Head cung cấp các khả năng của hợp đồng thông minh giống như chuỗi chính.

Để tránh quá tải với các chi tiết kỹ thuật, phần chính của bài nghiên cứu trình bày một phiên bản đơn giản hóa của Hydra để truyền đạt các khái niệm và ý tưởng cơ bản của giao thức mới. Cũng trong phần tổng quan, chúng tôi tập trung vào giao thức đơn giản hóa và phác thảo những điểm khác biệt của giao thức đầy đủ trong Phần 3.4. Mô tả chi tiết về quy trình đơn giản hóa được nêu trong Phần 4– 6 và Phụ lục B. Quy trình đầy đủ được mô tả trong Phụ lục C.

3.1. Bức tranh lớn

Để tạo một phiên bản giao thức Head, bất kỳ bên nào cũng có thể đóng vai trò là *người khởi tạo* (Initiator) và thiết lập một số bên (bao gồm cả chính mình), các *thành viên của Head*, tham gia vào Head bằng cách công bố danh tính của các bên.

Sau đó, mỗi bên sẽ thiết lập các kênh được xác thực theo cặp cho tất cả các bên khác hoặc nếu điều này là không thể thì sẽ hủy thiết lập giao thức.[1]

Sau đó, các bên trao đổi, thông qua các kênh được xác thực theo cặp, một số tài liệu khóa công khai. Tài liệu khóa công khai này được sử dụng cho cả việc xác thực các giao dịch On-Chain liên quan đến Head, chỉ giới hạn đối với các thành viên của Head (ví dụ: người không phải là thành viên không được phép đóng Head) và xác nhận sự kiện dựa trên đa chữ ký trong Head.

Sau đó, người khởi tạo thiết lập Head bằng cách gửi một giao dịch *ban đầu* đến chuỗi chính có chứa các tham số của Head và tạo ra (Forge) các *Token tham gia* đặc biệt xác định các thành viên của Head bằng cách gán từng Token cho khóa công khai được phân phối bởi bên tương ứng trong giai đoạn thiết lập. Giao dịch ban đầu cũng khởi tạo một máy trạng thái (xem Hình 3) cho Head quản lý việc “chuyển giao” các UTxO giữa chuỗi chính và Head.

[1]Chúng tôi thường giả định rằng có các cơ chế để thiết lập các kênh được xác thực theo cặp, chẳng hạn như bằng cơ sở hạ tầng khóa công khai.

Khi giao dịch ban đầu xuất hiện trên chuỗi chính, thiết lập trạng thái ban đầu **initial**, mỗi thành viên của Head có thể đính kèm một giao dịch *cam kết*, giao dịch này sẽ khóa (trên chuỗi chính) các UTxO mà họ muốn cam kết với Head.

Các giao dịch cam kết sau đó được thu thập bởi giao dịch *collectCom* tạo ra sự chuyển đổi từ **initial** sang **open**. Sau khi trạng thái mở **open** được xác nhận, các thành viên của Head bắt đầu chạy *giao thức Head* Off-Chain, giao thức này phát triển bộ UTxO ban đầu (liên minh trên tất cả các UTxO do tất cả các thành viên của Head cam kết) độc lập với chuỗi chính. Đối với trường hợp một số thành viên của Head không gửi giao dịch *cam kết*, Head có thể bị hủy bỏ bằng cách chuyển trực tiếp từ **initial** đến trạng thái cuối cùng **final**.

Giao thức Head được thiết kế để cho phép bất kỳ thành viên nào của Head tại bất kỳ thời điểm nào có thể sản xuất chứng chỉ cho bộ UTxO của Head hiện tại mà không cần tương tác. Sử dụng chứng chỉ này, thành viên của Head có thể đưa máy trạng thái về trạng thái đóng **closed**.

Sau khi đóng, máy trạng thái cấp cho các bên một *khoảng thời gian tranh chấp*, trong đó mỗi bên có thể (một lần) tranh chấp việc đóng bằng cách cung cấp chứng chỉ cho bộ UTxO mới hơn của Head. Việc tranh chấp dẫn trở lại trạng thái **closed**. Sau khi thời gian tranh chấp trôi qua, máy trạng thái có thể chuyển sang trạng thái **final**. Máy trạng thái thực thi các đầu ra của giao dịch dẫn đến trạng thái **final** tương ứng chính xác với bộ UTxO mới nhất được thấy trong khoảng thời gian tranh chấp.

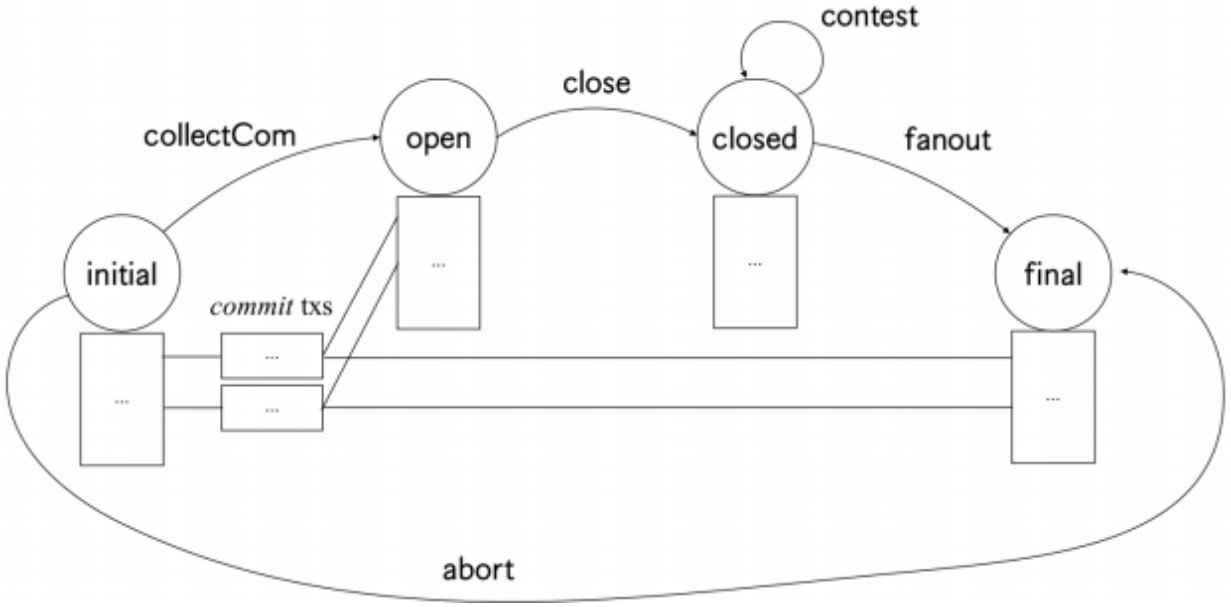
3.2. Máy trạng thái của chuỗi chính

Phần chuỗi chính của giao thức Hydra thực hiện hai chức năng chính: (1) nó khóa các UTxO trên chuỗi chính được cam kết với Head trong khi Head đang hoạt động và (2) nó tạo điều kiện giải quyết trạng thái cuối cùng của Head trở lại chuỗi chính sau khi Head được đóng lại. Khi kết hợp, hai chức năng này có hiệu quả trong việc thay thế UTxO ban đầu của Head được thiết lập bởi UTxO cuối cùng của Head được đặt trên chuỗi chính theo cách tôn trọng nhưng không duy trì toàn bộ các giao dịch của Head.

Máy trạng thái (Hình 3) thực hiện giao thức chuỗi chính bao gồm bốn trạng thái là ban đầu (**initial)**, mở (**open**), đóng (**closed**) và cuối cùng (**final**), trong đó hai trạng thái đầu tiên đảm nhận chức năng đầu tiên (khóa bộ UTxO ban đầu) và hai trạng thái tiếp theo đảm nhận chức năng thứ hai (đưa bộ UTxO cuối cùng đặt trên chuỗi chính).

Máy trạng thái vốn có tuần tự tất cả các hành động liên quan đến trạng thái máy. Điều này giúp đơn giản hóa cả việc lập luận và triển khai giao thức. Tuy nhiên, các

bước có thể được thực hiện song song bây giờ cần phải được tuần tự, điều này có thể ảnh hưởng đến hiệu suất. Đối với các trường hợp mà việc tuần tự này sẽ ảnh hưởng nghiêm trọng đến hiệu suất giao thức, chúng tôi sử dụng một kỹ thuật mới (theo hiểu biết của chúng tôi) để song song hóa tiến trình của máy trạng thái trên chuỗi chính.



Hình 3: Biểu đồ trạng thái chuỗi chính cho phiên bản đơn giản của giao thức Hydra.

Chúng tôi sử dụng kỹ thuật này để song song hóa việc xây dựng bộ UTxO ban đầu của Head. Nếu không có song song, tất cả *n* thành viên của Head sẽ phải đăng các giao dịch cam kết của họ (phần của họ trong bộ UTxO ban đầu) theo trình tự, yêu cầu một chuỗi tuyến tính gồm *n* giao dịch, mỗi giao dịch cho một chuyển đổi trạng thái tại một thời điểm. Thay vào đó, chúng tôi làm cho máy trạng thái sử dụng tất cả *n* giao dịch cam kết trong một chuyển đổi trạng thái duy nhất. Trong Hình 3, chúng tôi thể hiện điều này theo cách sau: giao dịch đại diện trạng thái ban đầu kết nối với giao dịch đại diện cho trạng thái mở không chỉ thông qua chuyển đổi trạng thái *collectCom*, mà còn thông qua một tập hợp các giao dịch cam kết (một giao dịch cho mỗi thành viên của Head).

Điều này đòi hỏi một số chăm sóc thêm. Chúng tôi muốn đảm bảo rằng mỗi thành viên của Head đăng chính xác một giao dịch cam kết và giao dịch **open** thu thập tất cả các giao dịch cam kết một cách trung thực. Chúng tôi đạt được sự đảm bảo này bằng cách phát hành một Token không thể thay thế duy nhất cho mỗi thành viên của Head, chúng tôi gọi đây là *Token tham gia*. Token này phải đi qua giao dịch cam kết của thành

viên Head tương ứng và giao dịch **open**, để hợp lệ, phải thu thập đầy đủ các Token tham gia. Chúng tôi có thể coi Token tham gia là đại diện cho *khả năng* và *nghĩa vụ* tham gia vào giao thức Head.

3.3. Giao thức Head

Giao thức Head bắt đầu với một bộ UTxO ban đầu U_0 giống với UTxO được khóa trên chuỗi.

Giao dịch và trạng thái UTxO cục bộ. Giao thức *xác nhận* các giao dịch riêng lẻ đồng thời hoàn toàn bằng cách thu thập và phân phối phân phối đa chữ ký trên mỗi giao dịch được phát hành một cách riêng biệt. Ngay sau khi giao dịch như vậy được xác nhận, nó không thể đảo ngược trở thành một phần của sự phát triển trạng thái UTxO của Head, đầu ra của giao dịch có thể chi tiêu ngay lập tức trong Head hoặc có thể được chuyển trở lại On-Chain một cách an toàn trong trường hợp đóng Head.

Mỗi bên duy trì quan điểm của họ về trạng thái UTxO cục bộ $\bar{L}$, đại diện cho bộ UTxO hiện tại được phát triển từ bộ UTxO ban đầu U_0 bằng cách áp dụng tất cả các giao dịch đã được xác nhận cho đến nay trong Head. Vì giao thức không đồng bộ nên quan điểm của các bên về trạng thái UTxO cục bộ thường khác nhau.

Ảnh chụp nhanh. Việc xử lý giao dịch ở trên sẽ đủ để phát triển trạng thái của Head. Tuy nhiên, cuối cùng một sự ngừng cam kết trên chuỗi sẽ phải chuyển toàn bộ lịch sử giao dịch trên chuỗi vì sẽ không có cách nào khác để chứng minh tính đúng đắn của bộ UTxO sẽ được khôi phục trên chuỗi.

Để giảm thiểu các yêu cầu lưu trữ cục bộ và cho phép ngừng cam kết trên chuỗi độc lập với lịch sử giao dịch, ảnh chụp nhanh UTxO U_1, U_2, ... liên tục được tạo. Đối với điều này, một *lãnh đạo ảnh chụp nhanh* (Snapshot Leader) yêu cầu chế độ xem của anh ta về trạng thái đã xác nhận $\bar{L}$ phải có nhiều chữ ký như một ảnh chụp nhanh mới, ảnh chụp nhanh Head đầu tiên tương ứng với trạng thái ban đầu U_0. Ảnh chụp nhanh được coi là *đã xác nhận* nếu nó được liên kết với đa chữ ký hợp lệ.

Ngược lại với các giao dịch, các ảnh chụp nhanh được tạo tuần tự. Để có ảnh chụp nhanh mới $U_{i+1} = \bar{L}$ đa chữ ký, lãnh đạo không cần gửi trạng thái cục bộ U_{i+1} của mình, mà chỉ cho biết bằng Hash, tập hợp các giao dịch (đã xác nhận) được áp dụng cho U_i để có được U_{i+1}.

Những người tham gia khác ký vào ảnh chụp nhanh ngay khi họ (cũng) nhìn thấy các giao dịch được xác nhận sẽ được xử lý trên ảnh chụp nhanh trước đó của nó: trạng thái được xác nhận của một bên luôn đi trước ảnh chụp nhanh được xác nhận mới nhất.

Ngay sau khi ảnh chụp nhanh được xác nhận, người tham gia có thể xóa tất cả các giao dịch đã được xử lý trong ảnh chụp nhanh một cách an toàn vì đa chữ ký của ảnh chụp nhanh hiện là bằng chứng cho thấy trạng thái này đã từng tồn tại trong quá trình phát triển Head.

Đóng Head. Một bên muốn đóng Head ngừng trạng thái đã xác nhận $\bar{L}$ của mình bằng cách đăng On-Chain ảnh chụp nhanh U_ℓ đã được xác nhận mới nhất cùng với các giao dịch đã xác nhận chưa được xử lý bởi ảnh chụp nhanh này. Trong thời gian tranh chấp tiếp theo, các thành viên khác của Head có thể đăng các trạng thái được xác nhận cục bộ của họ trên chuỗi.

3.4. Giao thức đầy đủ và các khía cạnh khác

Để cải thiện giao thức cơ bản, chúng tôi thay đổi máy trạng thái chuỗi chính (như được mô tả trong Phụ lục C) để bao gồm

- Các cam kết và ngừng cam kết tăng thêm (thêm vào hoặc xóa UTxO khỏi Head mà không cần đóng lại).
- Đóng Head một bước lạc quan mà không cần tranh chấp trên chuỗi.
- Đóng Head hai bước bi quan với khoảng thời gian tranh chấp $O(\Delta)$, độc lập với n, trong đó Δ là thời gian giải quyết trên chuỗi của một giao dịch.
- Ngừng cam kết phân tách On-Chain của bộ UTxO cuối cùng (trong trường hợp nó quá lớn để phù hợp với một giao dịch duy nhất).

Các khía cạnh khác của giao thức này được tóm tắt trong Phụ lục D:

- Việc xử lý các khoản phí khuyến khích các bên thúc đẩy máy trạng thái Head trên chuỗi chính.
- Việc xử lý các vấn đề về thời gian trong giao thức Head (không đồng bộ).
- Điều chỉnh giao dịch trong Head để tránh trạng thái của Head trở nên quá lớn trong điều kiện bi quan.

4. Thiết lập giao thức

Để tạo một phiên bản giao thức Head, người khởi tạo mời một nhóm người tham gia $\{p_1, ..., p_n\}$ (chính họ cũng thuộc người tham gia) tham gia bằng cách thông báo cho

họ các tham số giao thức: danh sách những người tham gia, các tham số của lược đồ (đa) chữ ký sẽ được sử dụng, v.v.

Sau đó, mỗi bên thiết lập các kênh được xác thực theo cặp cho tất cả các bên khác.

Đối với một số sơ đồ chữ ký điện tử, mỗi bên p_i tạo một cặp khóa ($k_{i,\text{ver}}$, $k_{i,\text{sig}}$) và gửi khóa xác minh $k_{i,\text{ver}}$ tương ứng của mình cho tất cả các bên khác. Lược đồ chữ ký số "tiêu chuẩn" này sẽ được sử dụng để xác thực các giao dịch trên chuỗi chính được giới hạn cho các thành viên của phiên bản giao thức Head.

Đối với lược đồ đa chữ ký (MS) —xem Phần 2.1 — mỗi bên p_i tạo một cặp khóa

$$(K_{i,\text{ver}}, K_{i,\text{sig}}) \leftarrow \textbf{MS-KG}\ (\Pi)$$

và gửi khóa xác minh $K_{i,\text{ver}}$ cho tất cả các bên khác.

Sau đó, mỗi bên sẽ tính khóa tổng hợp của mình từ các khóa xác minh đã nhận:

$$K_{\text{agg}} := \leftarrow \textbf{MS-AVK}\ (\Pi, (K_{j,\text{ver}})\ j\in[n]).$$

Lược đồ đa chữ ký sẽ được sử dụng để xác nhận Off-Chain (và xác minh Off-Chain và On-Chain) của các sự kiện giao thức Head.

Khi kết thúc lần khởi tạo này, mỗi bên p_i sẽ lưu trữ khóa ký của họ và tất cả đều nhận được khóa xác minh cho lược đồ chữ ký,

$$\left(k_{i,\text{sig}},\ \underline{k}_{\text{ver}} := (k_{j,\text{ver}})_{j\in[n]}\right)\ ,$$

và khóa ký của họ, khóa xác minh và khóa xác minh tổng hợp cho lược đồ đa chữ ký,

$$\left(K_{i,\text{sig}},\ \underline{K}_{\text{ver}} := (K_{j,\text{ver}})_{j\in[n]},\ K_{\text{agg}}\right)\ .$$

Nếu bất kỳ điều nào ở trên không thành công (hoặc có bên nào không đồng ý tham gia vào Head ngay từ đầu), bên đó sẽ hủy bỏ quy trình ban đầu và bỏ qua bất kỳ hành động nào khác.[2]

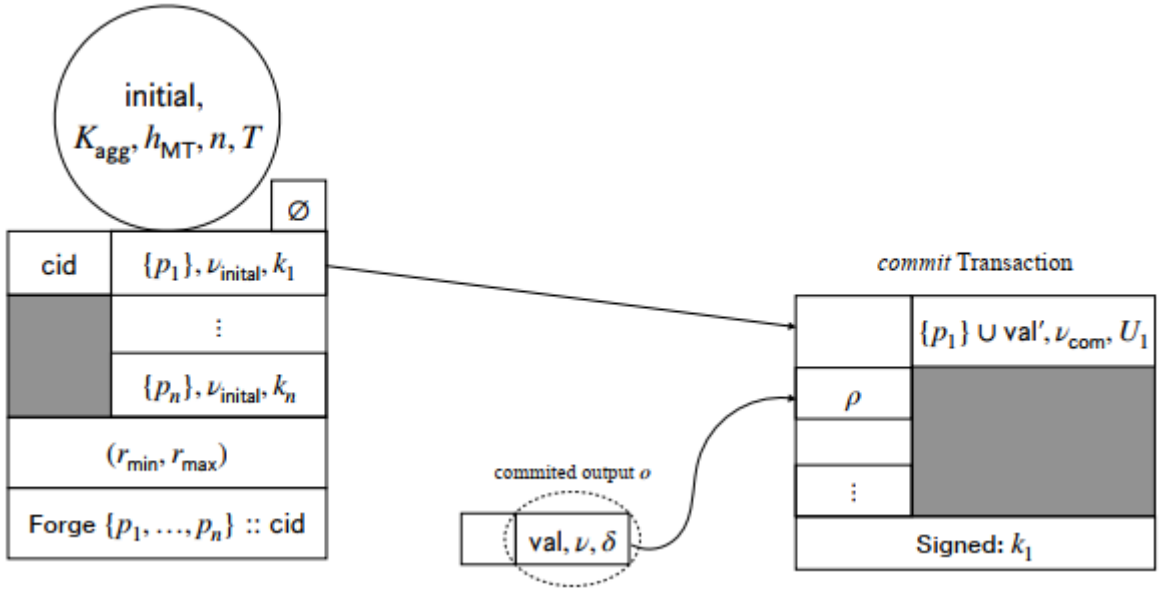
Người khởi tạo hiện đăng giao dịch *ban đầu* trên chuỗi như được mô tả trong Phần 5.

[2]Tất nhiên, việc hủy bỏ khởi tạo có thể đạt được một cách uyển chuyển hơn bằng cách thông báo rõ ràng cho người khởi tạo về việc không tham gia của một người. Các kỹ thuật thậm chí còn được biết đến để kết thúc một cuộc khởi tạo như vậy theo thỏa thuận của tất cả các bên [23].

5. Chuỗi chính

Ở đây chúng tôi mô tả chi tiết về máy trạng thái (SM – State Machine) chuỗi chính kiểm soát Hydra Head (xem Hình 3). Đối với các chuyển đổi trạng thái, mô tả chính thức về các điều kiện trong $tx^{\equiv}$ được bỏ qua để ưu tiên cho các giải thích trực quan trong văn bản và các hình.

Các thuật toán xác minh On-Chain. Trạng thái của Head được duy trì trong một biến η, là một phần của trạng thái SM và được cập nhật bằng *thuật toán xác minh On-Chain (OCV)* **Initial**, **Close**, **Contest** và **Final**. Trong ngữ cảnh của giao thức chuỗi chính, các thuật toán OCV này được cố ý giữ càng chung chung càng tốt; điều này giúp SM chuỗi chính tương thích với nhiều biến thể tiềm năng của giao thức Head. Các thuật toán OCV cụ thể cho giao thức Head được chỉ định trong nghiên cứu này được đưa ra trong ngữ cảnh của chính giao thức Head vì chúng phụ thuộc vào nội bộ của giao thức Head cụ thể: xác minh chứng chỉ giao thức Head và các cập nhật trạng thái On-Chain có liên quan. Như vậy, các thuật toán OCV có thể được coi là các thuật toán chuỗi chính trừu tượng được thực hiện bởi giao thức Head cụ thể. Do đó, việc triển khai OCV cho giao thức Head của chúng tôi được mô tả trong Phần 6.3.1.



Hình 4: Giao dịch *ban đầu* (bên trái) có đính kèm giao dịch *cam kết* (bên phải) và một trong các đầu ra bị khóa (ở giữa).

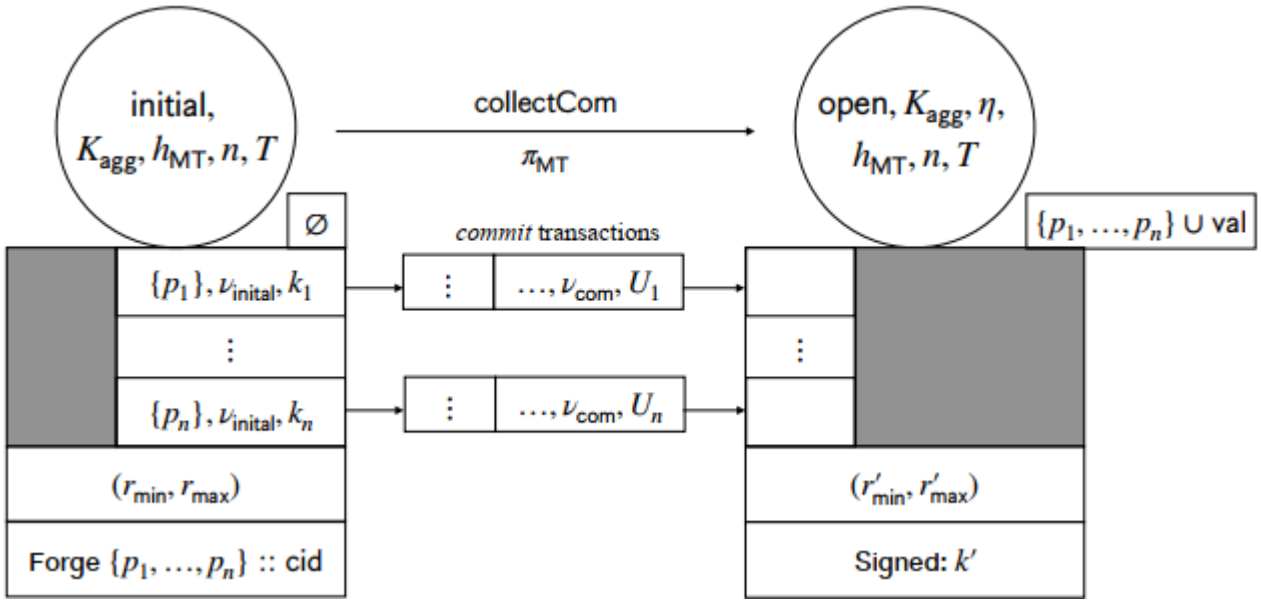
Trạng thái ban đầu. Sau giai đoạn thiết lập của Phần 4, người khởi tạo Head đăng một giao dịch *ban đầu* (xem Hình 4). Giao dịch *ban đầu* thiết lập trạng thái ban đầu (**initial**, K_{agg}, h_{MT}, n, T) của SM, trong đó **initial** là mã định danh trạng thái, K_{agg} là khóa đa chữ ký tổng hợp được thiết lập trong giai đoạn thiết lập, h_{MT} là gốc của cây Merkle cho các khóa xác minh chữ ký $\underline{k}_{ver} = (k_1, ..., k_n)$ được trao đổi trong giai đoạn thiết lập (xác định các thành viên của Head), n là số lượng các thành viên của Head và T là độ dài của giai đoạn tranh chấp. Giao dịch *ban đầu* cũng tạo ra (Forge) n Token tham gia $\{p_1, ..., p_n\}$ :: **cid**, trong đó ID tiền tệ **cid** được cung cấp bởi *tập lệnh chính sách tiền tệ duy nhất* được sử dụng bởi đầu vào **cid**. Tập lệnh là duy nhất vì nó được ràng buộc với một đầu ra và sổ cái ngăn chặn việc chi tiêu gấp đôi. Do đó, chúng ta có thể sử dụng **cid** làm mã định danh duy nhất cho Head mới được khởi tạo.

Điều quan trọng, giao dịch *ban đầu* có n đầu ra, trong đó mỗi đầu ra bị khóa bởi trình xác thực $v_{initial}$ và đầu ra thứ i có k_i trong trường dữ liệu của nó. Trình xác thực $v_{initial}$ đảm bảo những điều sau: hoặc đầu ra được chi tiêu bởi

1. Một giao dịch hủy bỏ SM (xem bên dưới) hoặc
2. Một giao dịch cam kết (được xác định bằng cách có trình xác thực v_{com} trong đầu ra duy nhất của nó) và
    (a) Giao dịch được ký và chữ ký xác minh là hợp lệ với khóa xác minh k_i,
    (b) Trường dữ liệu của đầu ra của giao dịch cam kết là U_i = **makeUTxO**(o_1, ..., o_m), trong đó o_j là đầu ra được tham chiếu bởi đầu vào của giao dịch cam kết và **makeUTxO** các cặp lưu trữ (**out-ref**$_j$, o_j) trong số các đầu ra o_j với tham chiếu đầu ra tương ứng **out-ref**$_j$.

Tính hợp lý và hợp lệ chung của giao dịch *ban đầu* được kiểm tra trên chuỗi chính. Các thành viên của Head cũng kiểm tra xem các tham số Head có khớp với các tham số đã thống nhất trong giai đoạn thiết lập hay không. Trong trường hợp không khớp, việc mở Head coi như không thành công.

Cam kết đầu ra cho một Head. Để khóa đầu ra cho Hydra Head, thành viên thứ i của Head sẽ đính kèm một giao dịch *cam kết* (xem Hình 4) vào đầu ra thứ i của giao dịch *ban đầu*. Trình xác thực v_{com} đảm bảo rằng giao dịch *cam kết* ghi lại chính xác từng phần bộ UTxO U_i mà bên đó đã cam kết.



Hình 5: Giao dịch *ban đầu* (bên trái) với giao dịch *collectCom* (bên phải) và giao dịch *cam kết* (ở giữa).

Tất cả các giao dịch *cam kết* sẽ lần lượt được thu thập bởi một giao dịch SM - *collectCom* hoặc *abort* (xem bên dưới).

Thu thập cam kết. Quá trình chuyển đổi SM từ **initial** sang **open** được thực hiện bằng cách gửi giao dịch *collectCom* (xem Hình 5). Tất cả các tham số K_{agg}, h_{MT}, n và T vẫn là một phần của trạng thái, nhưng ngoài ra, một giá trị $\eta \leftarrow$ **initial**(U_1, ..., U_n) được lưu trữ trong trạng thái. Ý tưởng là η lưu trữ thông tin về bộ UTxO ban đầu, được tạo thành từ các bộ UTxO U_i riêng lẻ thu thập được từ các giao dịch cam kết, để xác minh thông tin trạng thái của Head sau này (xem bên dưới).

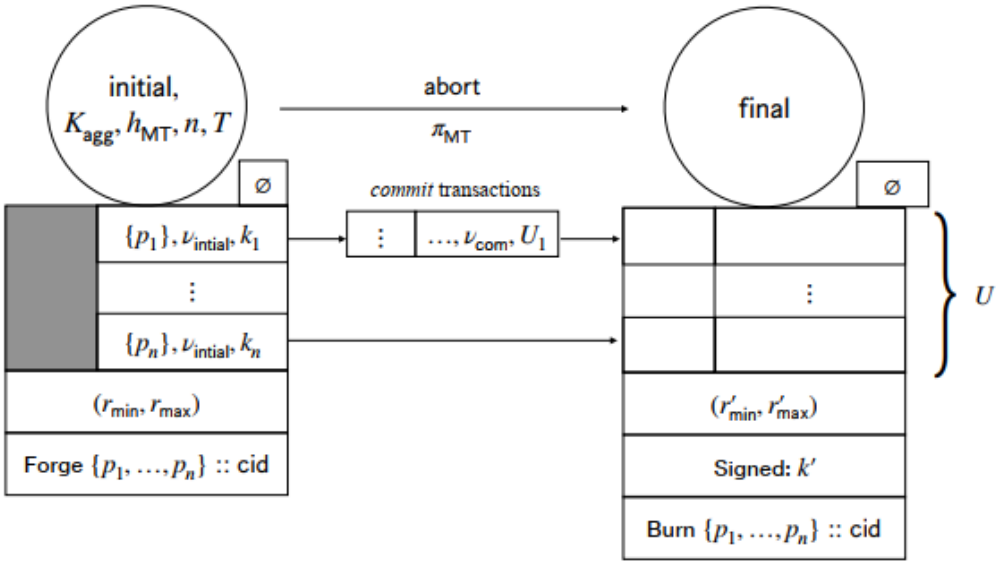
Nó cũng được yêu cầu rằng tất cả n Token tham gia phải có trong đầu ra SM của giao dịch *collectCom*. Điều này đảm bảo rằng giao dịch *collectCom* thu thập tất cả n giao dịch cam kết. Lưu ý rằng vì $\nu_{initial}$ không cho phép giao dịch cam kết SM chi tiêu đầu ra của giao dịch ban đầu, cách duy nhất để gửi giao dịch *collectCom* là nếu mỗi thành viên của Head đã gửi giao dịch cam kết.

Cuối cùng, hãy lưu ý rằng quá trình chuyển đổi yêu cầu bằng chứng π_{MT} rằng người ký *k'* nằm trong Cây Merkle thuộc h_{MT}, điều này đảm bảo rằng chỉ các thành viên của Head mới có thể gửi các giao dịch SM. Đây sẽ là trường hợp cho tất cả các chuyển đổi được xem xét trong nghiên cứu này (và sẽ không được chỉ ra thêm).

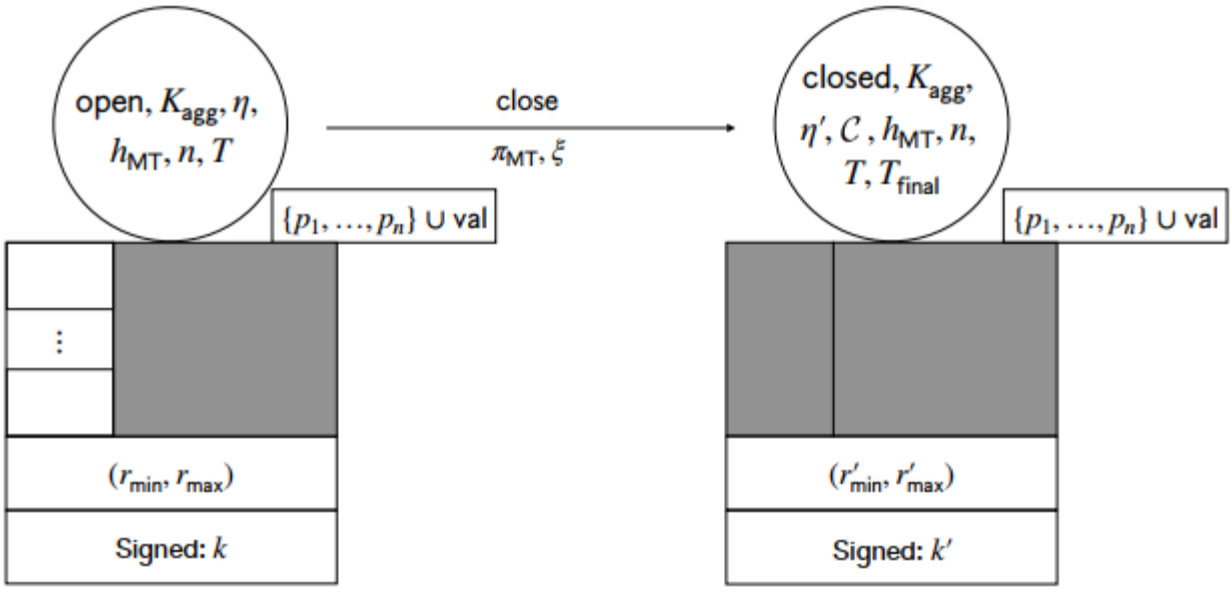
Huỷ bỏ một Head. Giao dịch *hủy bỏ* (xem Hình 6) cho phép một bên hủy bỏ việc tạo Head trong trường hợp một số bên không gửi giao dịch cam kết. Trạng thái cuối cùng không chứa bất kỳ thông tin nào (ngoài số nhận dạng của nó), nhưng nó được đảm bảo rằng (1) đầu ra U tương ứng với sự kết hợp của tất cả các bộ UTxO đã cam kết U_i và (2) tất cả các Token tham gia đều được đốt.

Giao dịch đóng. Để đóng một Head, một thành viên của Head có thể đăng giao dịch *đóng* (xem Hình 7), điều này dẫn đến trạng thái chuyển từ **open** sang **closed**. Để đóng thành công, thành viên của Head phải cung cấp thông tin hợp lệ ξ về (quan điểm của họ về) trạng thái của Head hiện tại. Thông tin này được chuyển qua thuật toán OCV **Close**, dẫn đến trạng thái OCV mới $\eta' \leftarrow$ **Close**(K_{agg}, η, ξ). Thuật toán OCV **Close** sử dụng trạng thái OCV η và K_{agg} trước đó để kiểm tra thông tin ξ của Head. Hãy lưu ý rằng nếu kiểm tra không thành công, **Close** có thể xuất ra $\perp$, nhưng để giao dịch *đóng* có hiệu lực, $\eta' \neq \perp$ là bắt buộc.

Sau khi một giao dịch *đóng* đã được đăng, một *khoảng thời gian tranh chấp* sẽ bắt đầu kéo dài ít nhất là T Slot. Do đó, Slot cuối cùng T_{final} của giai đoạn tranh chấp được ghi lại ở trạng thái, và nó được đảm bảo rằng $T_{final} \geq r'_{max} + T$.



Hình 6: Giao dịch *ban đầu* (bên trái) với giao dịch *hủy bỏ* (bên phải) và giao dịch *cam kết* (ở giữa).



Hình 7: Giao dịch *collectCom* (bên trái) với giao dịch *đóng* (bên phải).

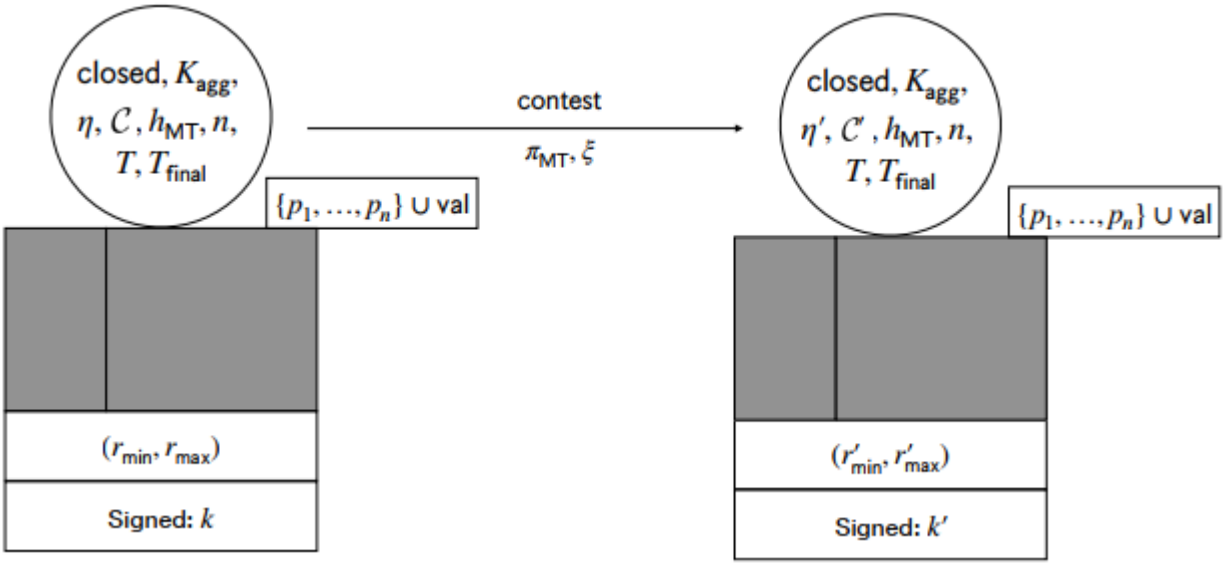
Cuối cùng, trạng thái SM được mở rộng bởi một tập C được khởi tạo cho khóa ký của người đăng, tức là $C \leftarrow \{k'\}$. C được sử dụng để đảm bảo rằng không có bên nào đăng nhiều hơn một lần trong suốt thời gian tranh chấp.

Sự tranh chấp. Nếu bên đầu tiên đóng Head đăng thông tin cũ / không đầy đủ về trạng thái hiện tại của Head, thì bất kỳ bên nào khác có thể đăng giao dịch *tranh chấp* (xem Hình 8), điều này gây ra sự chuyển đổi trạng thái từ trạng thái **closed** sang tranh chấp (**0ontest)**. Quá trình chuyển đổi xử lý thông tin cập nhật ξ bằng cách chuyển nó qua thuật toán OCV **Contest**, dẫn đến trạng thái OCV mới $\eta' \leftarrow$ **Contest** (K_{agg}, η, ξ). Thuật toán OCV **Contest** sử dụng trạng thái OCV η và K_{agg} trước đó để kiểm tra thông tin cập nhật ξ. Tương tự như **Close**, **Contest** có thể xuất ra $\perp$, nhưng để giao dịch *tranh chấp* có hiệu lực thì cần phải có $\eta' \neq \perp$.

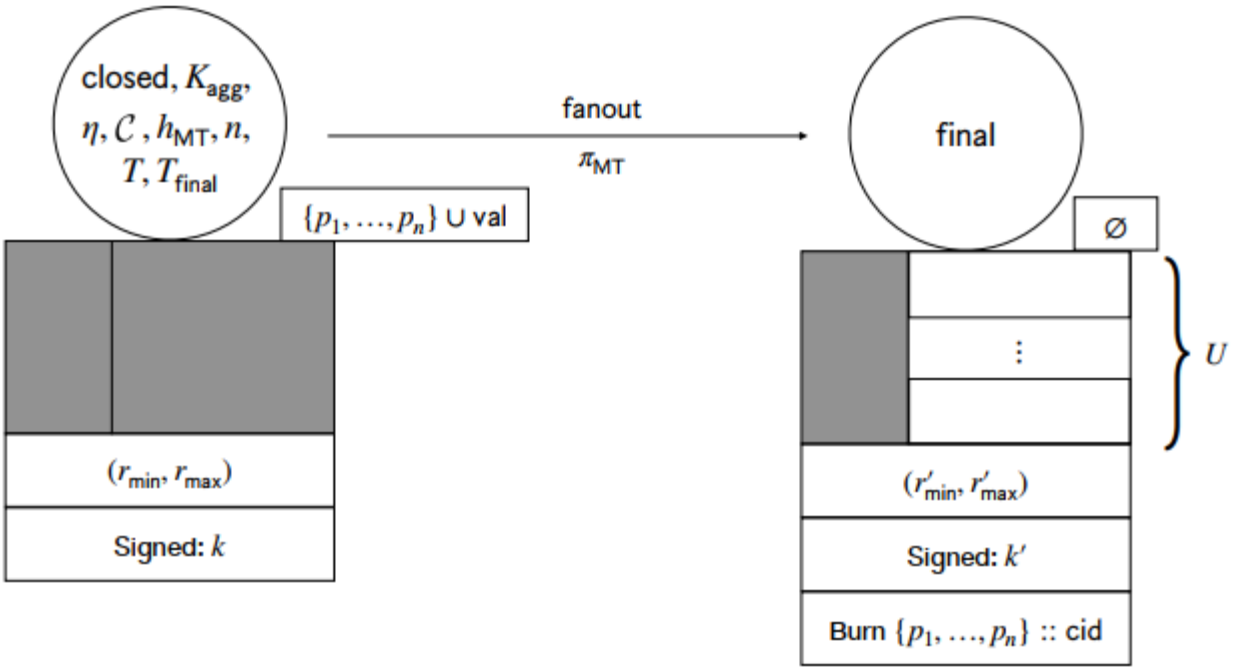
Giao dịch *tranh chấp* chỉ có hiệu lực nếu tập hợp C cũ của các bên đã tranh chấp (hoặc đã đóng) cho đến nay vẫn chưa bao gồm người đăng, tức là $k' \notin C$. Nếu việc kiểm tra này thông qua, tập hợp được mở rộng để bao gồm người đăng của giao dịch *tranh chấp*, tức là, $C' \leftarrow C \cup \{k'\}$. Hơn nữa, các giao dịch *tranh chấp* chỉ có thể được đăng cho đến khi T_{final}, tức là bắt buộc phải $r'_{max} \leq T_{final}$.

Hãy lưu ý rằng trong suốt thời gian tranh chấp, có thể đăng tối đa $n - 1$ giao dịch *tranh chấp* (tất nhiên, tham số T phải được chọn đủ lớn để cho phép mỗi thành viên của Head có khả năng đăng một giao dịch *đóng* / *tranh chấp*).

Trạng thái cuối cùng. Sau khi giai đoạn tranh chấp kết thúc, Head có thể được hoàn thiện bằng cách đăng một giao dịch *fanout*, đưa SM từ trạng thái **closed** đến **final**. Giao dịch *fanout* phải có đầu ra tương ứng với trạng thái của Head gần nhất. Để đạt được điều đó, vị từ OCV **Final** sẽ kiểm tra bộ đầu ra U của giao dịch dựa trên thông tin được ghi trong η. Giao dịch *fanout* chỉ hợp lệ nếu kết quả cuối cùng là True. Hơn nữa, để đảm bảo rằng giao dịch *fanout* không được đăng quá sớm, $r'_{\text{min}} > T_{\text{final}}$ là bắt buộc. Cuối cùng, tất cả các Token tham gia phải được đốt.



Hình 8: Giao dịch *đóng* / *tranh chấp* (bên trái); giao dịch *tranh chấp* (bên phải)



Hình 9: Giao dịch *đóng* / *tranh chấp* (bên trái); giao dịch *fanout* (bên phải)

6. Giao thức Head đơn giản mà không cần giải quyết xung đột

Phần này mô tả phiên bản đơn giản hóa của giao thức Head và không cần giải quyết xung đột, với mục tiêu thể hiện các kiến thức cơ bản về giao thức mà không làm quá tải bản trình bày với quá nhiều chi tiết. Giải quyết xung đột được thêm vào giao thức trong Phụ lục B và giao thức đầy đủ được phác thảo trong Phụ lục C.

Đầu tiên chúng tôi giới thiệu định nghĩa bảo mật cho giao thức Head trong Phần 6.1. Máy giao thức được mô tả trong Phần 6.2, mã Code chuỗi chính dành riêng cho Head trong Phần 6.3 và bằng chứng bảo mật cho giao thức Head được đưa ra trong Phần 6.4.

6.1. Định nghĩa bảo mật

6.1.1. Cú pháp giao thức

Cú pháp giao thức Head là **HP** = (**Prot**, **Initial**, **Close**, **Contest**, **Final**). Thành phần chính là máy giao thức **Prot**, một phiên bản được chạy bởi mọi thành viên của Head. Các thuật toán khác được sử dụng để thiết lập và xác minh trên chuỗi đồng thời tạo giao diện cho chuỗi chính. Đặc biệt,

- $\Sigma \leftarrow$ tạo các tham số toàn cầu,

- $(K_{ver}, K_{sig}) \leftarrow$ **MS-KG** (Σ) cho phép mọi thành viên Head tạo tài liệu khóa công khai / cá nhân mới dựa trên các tham số toàn cầu,
- $K_{agg} \leftarrow$ **MS-AVK** $(\Sigma, (K_{ver,\,i})_i)$ tổng hợp các khóa công khai, và
- **Initial**, **Close**, **Comtest** và **Final** là các thuật toán xác minh On-Chain (xem Phần 5).

Máy giao thức Head **Prot** có giao diện như sau với môi trường:

- input (**init**, i, $\underline{K}_{ver}$, K_{sig}, U_0) được sử dụng để khởi tạo giao thức Head, cho bên có chỉ số i, với Vectơ của tài liệu khóa công khai $\underline{K}_{ver}$, tài liệu khóa cá nhân K_{sig} và một bộ UTxO ban đầu U_0;
- input (**new**, tx) được sử dụng để gửi một giao dịch tx mới;
- output (**seen**, tx) thông báo rằng giao dịch tx đã được nhìn thấy (bởi bên đưa ra thông điệp);
- output (**conf**, tx) thông báo rằng giao dịch tx đã được xác nhận (theo quan điểm của bên đưa ra thông điệp);
- input (**close**) được sử dụng để bắt đầu đóng Head (tạo ra một chứng chỉ ξ); và
- input (**cont**, η) được sử dụng để tranh chấp (tạo ra một chứng chỉ ξ).

6.1.2. Bảo mật giao thức

Định nghĩa bảo mật cho giao thức Head đảm bảo bốn thuộc tính được đưa ra một cách trực quan như sau:

- Tính nhất quán (Consistency): Không có hai bên không bị gián đoạn nào thấy các giao dịch xung đột được xác nhận.
- Tính sống động (Liveness): Nếu tất cả các bên vẫn không bị gián đoạn và đối thủ gửi tất cả các thông điệp, thì mọi giao dịch sẽ được xác nhận vào một thời điểm nào đó.
- Tính hợp lý (Soundness): Bộ UTxO cuối cùng được chấp nhận trên chuỗi chính là kết quả từ một tập hợp các giao dịch đã thấy.
- Tính hoàn chỉnh (Completeness): Tất cả các giao dịch được quan sát như được xác nhận bởi một bên trung thực ở cuối giao thức được xem xét trên chuỗi chính.

Thử nghiệm cho định nghĩa bảo mật. Các thuộc tính bảo mật ở trên được thu thập bằng cách xem xét một thử nghiệm ngẫu nhiên liên quan đến

- Một đối thủ A,

- Một mạng lưới dưới sự kiểm soát toàn bộ lịch trình của A, có thể bỏ qua thoong điệp hoặc trì hoãn chúng một cách tùy ý,
- Giai đoạn thiết lập,
- n bên p_i, có thể thay đổi bởi A, chạy giao thức Head với các tham số từ giai đoạn thiết lập và một tập UTxO ban đầu U_0 được chọn bởi A, và
- Một chuỗi chính trừu tượng (hầu hết) được điều khiển bởi A.

Thử nghiệm kết thúc khi máy trạng thái chuỗi chính chuyển sang trạng thái cuối cùng và đối thủ sẽ thắng nếu một số điều kiện nhất định không được thỏa mãn khi kết thúc thử nghiệm. Chi tiết hơn, thử nghiệm tiến hành như sau:

1. Tham số toàn cầu $\Sigma \leftarrow$ **MS-Setup** được tạo và Σ được chuyển cho A.
2. Đối với mỗi bên p_i, tài liệu khóa $(K_{ver,i}, K_{sig,i}) \leftarrow$ **MS-KG** (Σ) được tạo và Vectơ $\underline{K}_{ver}$ của tất cả khóa công khai và $K_{agg} \leftarrow$ **MS-AVK** $(\Sigma, \underline{K}_{ver})$ được chuyển cho A.
3. Mỗi bên p được khởi tạo bằng (**init**, i, $\underline{K}_{ver}$, $K_{sig,i}$, U_0), trong đó U_0 được chọn bởi A.
4. Giờ đây, đối thủ có thể kiểm soát đầu vào của các bên (ví dụ: giao dịch mới, yêu cầu đóng / tranh chấp) và xem kết quả đầu ra (ví dụ: giao dịch đã thấy và đã xác nhận). Việc ghi sổ kế toán sau đây diễn ra:
    - Khi một bên p_i không bị gián đoạn xuất ξ theo lệnh **close**, bản ghi (**close**, i, ξ);
    - Khi bên p_i không bị gián đoạn xuất ξ theo lệnh (**cont**, η), bản ghi (**cont**, i, η, ξ).

"Song song" với điều trên, thử nghiệm đặt C, $H_{cont} \leftarrow \emptyset$ và thực hiện những điều sau để mô phỏng chuỗi chính:

(a) Khởi tạo $\eta \leftarrow (U_0, 0, \emptyset)$.

(b) Khi A cung cấp (i, ξ): nếu i không bị gián đoạn, ξ được thay thế bằng ξ được ghi trong (**close**, i, ξ) và $H_{cont} \leftarrow H_{cont} \cup \{i\}$. Sau đó, $\eta \leftarrow$ **Close** (K_{agg}, η, ξ) và $C \leftarrow C \cup \{i\}$ được tính toán. Nếu **Close** từ chối, mọi thứ trong bước này sẽ bị loại bỏ và bước này được lặp lại.

(c) Đối thủ liên tục cung cấp (i, ξ) với $i \notin C$; nếu i không bị gián đoạn, ξ được thay thế bằng ξ được ghi trong (**cont**, i, ξ) và $H_{cont} \leftarrow H_{cont} \cup \{i\}$. Sau đó, $\eta \leftarrow$ **Contest** (K_{agg}, η, ξ) và $C \leftarrow C \cup \{i\}$ được tính toán. Nếu **Contest** từ chối, mọi thứ trong bước này sẽ bị hủy.

(d) Khi đối thủ cung cấp U_{final}, $b \leftarrow$ **Final**(η, U_{final}) được tính và thử nghiệm kết thúc.

Giao thức của chúng tôi cung cấp các đảm bảo bảo mật khác nhau tùy thuộc vào mức độ tham nhũng của đối thủ. Nó cung cấp tính đúng đắn độc lập với cả hai, số lượng các bên Head bị hỏng và các điều kiện mạng lưới. Nhưng đảm bảo rằng giao thức đạt được tiến bộ (tức là các giao dịch mới được xác nhận trong Head) chỉ được cung cấp trong trường hợp không có bên Head nào bị hỏng và điều kiện mạng tốt.

Để nắm bắt sự khác biệt này, chúng tôi phân biệt:
Kẻ tấn công đang hoạt động. Kẻ tấn công đang hoạt động A có toàn quyền kiểm soát giao thức, tức là anh ta hoàn toàn không bị hạn chế trong trò chơi bảo mật ở trên.
Kẻ tấn công mạng lưới. Kẻ tấn công mạng lưới Aø không làm hỏng bất kỳ bên Head nào, cuối cùng sẽ gửi tất cả các thông điệp mạng đã gửi (tức là không bỏ bất kỳ thông điệp nào) và không gây ra sự kiện **close**. Ngoài hạn chế này, kẻ tấn công có thể hành động tùy tiện trong thí nghiệm trên.

Sự kiện bảo mật. Hãy xem xét các biến ngẫu nhiên sau:

- $\hat{S}_i$: tập hợp các giao dịch tx cho bên p_i, *trong khi không bị gián đoạn*, ouput (**seen**, tx);
- $\bar{C}_i$: tập hợp các giao dịch tx cho bên p_i, *trong khi không bị gián đoạn*, output (**conf**, tx);
- H_{cont} : tập hợp (tại thời điểm) các bên không bị gián đoạn đã đưa ra ξ theo yêu cầu đóng / tranh chấp và ξ được áp dụng để sửa η (xem ở trên); và
- H: nhóm các bên không bị gián đoạn.

Tính bảo mật của giao thức Head được nắm bắt bằng cách xem xét các sự kiện sau, mỗi sự kiện tương ứng với một trong các thuộc tính bảo mật được giới thiệu ở trên:

- Tính nhất quán (Head): Khi có kẻ tấn công đang hoạt động, điều kiện là: Với mọi i, j, $U_0 \circ (C_i \cup C_j) \neq \perp$, tức là không có hai bên không bị gián đoạn nào thấy các giao dịch xung đột được xác nhận.
- Tính sống động (Head): Khi có kẻ tấn công mạng lưới, điều kiện sau được áp dụng: Đối với bất kỳ đầu vào giao dịch tx nào thông qua (**new**, tx), điều sau đây cuối cùng được giữ nguyên: $\text{tx} \cap_{i \in [n]} \bar{C}_i \vee \forall i$: $U_0 \circ (C_i \cup \{\text{tx}\}) = \perp$, tức là mọi bên

sẽ quan sát giao dịch được xác nhận hoặc mọi bên sẽ quan sát giao dịch mâu thuẫn với giao dịch đã xác nhận của mình.[3]

- Tình hợp lý (Chuỗi): Khi có kẻ tấn công đang hoạt động, điều kiện sau được thỏa mãn: $\exists \tilde{S} \subseteq \cap_{i \in H}\ \hat{S}_i : U_{\text{final}} = U_0 \circ \tilde{S}$, tức là bộ UTxO cuối cùng là kết quả của một bộ giao dịch đã thấy.
- Tính hoàn chình (Chuỗi): Khi có kẻ tấn công đang hoạt động, điều kiện là: Đối với $\tilde{S}$ như trên, $\cup_{pi} \in H_{\text{cont}}\ \bar{C}_i \subseteq \tilde{S}$, tức là tất cả các giao dịch được coi là xác nhận bởi một bên trung thực ở cuối giao thức đều được xem xét.

Hãy lưu ý rằng giao thức đơn giản hóa của chúng tôi *với giải quyết xung đột* và giao thức đầy đủ của chúng tôi trong Phụ lục B và C đạt được tính sống động theo nghĩa trên, nhưng giao thức đơn giản hóa *không cần giải quyết xung đột* của chúng tôi trong Phần 6 chỉ đạt được khái niệm yếu hơn về tính sống động, cụ thể là

Thực thi không xung đột: Cho $N = \{\text{tx} \mid (\textbf{new}, \text{tx})\}$ tập hợp tất cả các giao dịch đầu vào cho một sự kiện mới trong quá trình thực thi giao thức Head. Thực thi giao thức Head không có *xung đột* nếu $U_0 \circ \text{N} \neq \perp$.

Tương ứng, khía cạnh tính sống động của giao thức đơn giản hóa không có giải pháp xung đột được ghi lại bởi sự kiện sau, thay vào đó:

- Tính sống động không xung đột (Head): Khi đối mặt với kẻ tấn công mạng lưới, người thực thi không có xung đột thỏa mãn điều kiện sau: Đối với bất kỳ đầu vào giao dịch tx nào thông qua (**new**, tx), $\text{tx} \in \cap_{i \in [n]} \bar{C}_i$ cuối cùng được giữ.

6.2. Máy giao thức

Máy giao thức **Prot** bao gồm một số chương trình con xử lý đầu vào từ môi trường (ví dụ: lệnh máy khách đưa ra một giao dịch mới để xác nhận hoặc sự xuất hiện của yêu cầu xác nhận của một bên khác). Giao thức được mô tả trong Hình 10. Tất cả các ký hiệu không rõ ràng có liên quan được giải thích trong các đoạn sau.

6.2.1. Đại diện trạng thái cục bộ

Mỗi bên duy trì các đối tượng cục bộ để đại diện cho các giao dịch, ảnh chụp nhanh và bộ Head-UTxO cục bộ của mình. Các đối tượng này tồn tại trong hai phiên bản, một

[3]Đặc biệt, *tính sống động* thể hiện giao thức tiến triển trong các điều kiện mạng hợp lý nếu không có bên Head nào bị hỏng, ngụ ý rằng với mọi giới hạn trên được đảm bảo về độ trễ gửi thông điệp, thời gian xác nhận giao dịch trong trường hợp xấu nhất được giới hạn trong *hàm δ*.

đối tượng *được nhìn thấy* đã được bên ký tên (bên đã nhìn thấy và chấp thuận sự kiện); và một đối tượng *được xác nhận* được liên kết với một chữ ký hợp lệ (bên đã nhận được một chữ ký hợp lệ trên đối tượng). Đối tượng nhìn thấy X được ký hiệu là $\hat{X}$ và đối tượng được xác nhận là $\bar{X}$.

Trạng thái giao thức cục bộ của một bên bao gồm các khóa xác minh đa chữ ký và khóa ký riêng của bên đó, các bộ đếm ảnh chụp nhanh $\hat{s}$ và $\bar{s}$, và của các biến

- $\hat{U}$ và $\bar{U}$, theo dõi các lần tương ứng được xem gần đây nhất, xác nhận, ảnh chụp nhanh,
- $\hat{L}$ và $\bar{L}$, theo dõi các lần tương ứng đã thấy gần đây, bộ UTxO đã xác nhận và
- $\hat{T}$ và $\bar{T}$, các tập hợp tương ứng đã thấy, đã xác nhận, các giao dịch chưa được xem xét bằng ảnh chụp nhanh.

Các biến $\hat{U}$ và $\bar{U}$ lưu trữ *đối tượng ảnh chụp nhanh*, là cấu trúc dữ liệu lưu giữ thông tin về ảnh chụp nhanh. Cụ thể, một đối tượng ảnh chụp nhanh U có cấu trúc sau:

$U.s$	Số ảnh chụp nhanh
$U.U$	Bộ UTxO tương ứng
$U.h$	Hash của U
$U.T$	Tập hợp các giao dịch liên quan đến ảnh chụp nhanh với người trước đó của nó
$U.S$	Bộ tích lũy chữ ký (mảng chữ ký)
$U.\tilde{\sigma}$	Đa chữ ký

Hàm **snObj**(s, U, T) khởi tạo một đối tượng ảnh chụp nhanh và được giải thích sau.

Tương tự, $\hat{T}$ và $\bar{T}$ lưu trữ các tập hợp các *đối tượng giao dịch*. Đối tượng giao dịch tx có cấu trúc sau:

tx.i	Chỉ số của bên phát hành giao dịch để chứng nhận
tx.tx	Giao dịch
tx.h	Hash của tx
tx.S	Bộ tích lũy chữ ký (mảng chữ ký)
tx.$\tilde{\sigma}$	Đa chữ ký

Hàm **txObj**(*i*, tx) khởi tạo một đối tượng giao dịch bằng cách đặt các trường thích hợp thành các giá trị đã thông qua (bao gồm tính toán Hash của tx) và các trường còn lại thành ∅ tương ứng ⊥.

6.2.2. Xác nhận thực thể ba vòng

Các giao dịch và ảnh chụp nhanh được xác nhận trong một quy trình 3 vòng không đồng bộ:[4]

- **req**: Người phát hành giao dịch hoặc ảnh chụp nhanh yêu cầu thực thể được ký bằng cách gửi mô tả thực thể cho mọi thành viên của Head.
- **ack**: Các thành viên của Head thừa nhận thực thể đang trả lời các chữ ký của họ trên thực thể cho tổ chức phát hành.
- **conf**: Người phát hành thu thập tất cả các chữ ký, kết hợp đa chữ ký và gửi đa chữ ký cho tất cả các thành viên của Head.

6.2.3. Ký hiệu mã Code

Mã Code được mô tả bằng cách xem của một bên Head chung p_i. Chúng tôi giả định rằng một bên chỉ chấp nhận các thông điệp được xác thực bởi người gửi đã xác nhận quyền sở hữu của mình (bằng cách sử dụng các kênh đã xác thực được thiết lập trong giai đoạn thiết lập) —các tin nhắn đã được xác thực chỉ đơn giản là người nhận không nhìn thấy. Vì đơn giản, bất cứ khi nào một bên p_i gửi một thông điệp đến tất cả các bên Head, nó cũng gửi thông điệp đó cho chính nó.

Đối với bộ giao dịch $\hat{T}$ (và tương tự $\bar{T}$), $\hat{T}[h]$ biểu thị tx $\in \hat{T}$ sao cho tx.$h = h$, tức là đối tượng giao dịch tương ứng với giao dịch có Hash $H(\text{tx}) = h$.

Toán tử ↓ biểu thị phép chiếu của một đối tượng lên một tập con các trường của nó. Ví dụ, $\hat{T}^{\downarrow(h)}$ biểu thị tập hợp các Hash tương ứng với các giao dịch trong $\hat{T}$.

Ký hiệu sau đây được sử dụng để mô tả việc áp dụng các giao dịch cho một bộ UTxO nhất định.

- $U' = U \circ \text{tx}$ gán cho U' bộ UTxO do áp dụng giao dịch tx cho bộ UTxO U. Trong trường hợp xác thực không thành công, nó trả về $U' = \perp$.

[4]Lưu ý rằng, là một biến thể, quy trình 3 vòng này (với giao tiếp tuyến tính trong n) có thể được cô đọng thành 2 vòng (với giao tiếp bậc hai ở n) bằng cách kết hợp hai vòng cuối cùng thành một thông báo chữ ký "tất cả cho tất cả". Biến thể này có thể thích hợp hơn cho *n nhỏ*.

- $U' = U \circ T$ gán cho U' bộ UTxO do áp dụng tất cả giao dịch trong tập giao dịch T thành bộ UTxO U. Trong trường hợp không phải tất cả các giao dịch đều có thể được áp dụng, nó trả về $U' = \perp$.

Trong các quy trình giao thức của Hình 10, theo **require**(P), chúng tôi biểu thị rằng vị từ P phải được thỏa mãn để thực hiện thêm một quy trình — trong khi kết thúc ngay lập tức trên $\neg P$. Bằng cách **wait**(P), chúng tôi thể hiện sự chờ đợi không chặn đối với vị từ P được thỏa mãn. Trên $\neg P$, việc thực hiện quy trình được dừng lại, xếp hàng đợi và kích hoạt lại ngay khi P được thỏa mãn. Cuối cùng, chúng tôi giả định các lần thực thi mã Code của mỗi quy trình là nguyên tử, loại trừ các Block mã Code có thể được đưa vào hàng đợi để thực thi sau này, trong trường hợp đó, chúng tôi giả định Block chờ là nguyên tử.

6.2.4. Luồng giao thức

Khởi tạo Head. Ban đầu, bằng cách kích hoạt thông qua sự kiện (**init**), các bên lưu trữ tài liệu khóa đa chữ ký của họ ở giai đoạn thiết lập, bộ quảng cáo $\bar{L} = \hat{L} = \bar{U} = \hat{U} = U_0$ trong đó U_0 là bộ UTxO ban đầu được trích xuất từ trạng thái η của giao dịch *collectCom* (xem Hình 5). Các bộ giao dịch ban đầu trống, $\bar{T} = \hat{T} = \emptyset$; và $\bar{s} = \hat{s} = 0$.

Xác nhận giao dịch mới.

(**new**). Bất cứ lúc nào, bằng cách gọi (**new**, tx), một bên Head có thể (không đồng bộ) đưa một giao dịch tx mới vào giao thức Head, bắt đầu quy trình xác nhận 3 vòng cho tx như được mô tả trong Phần 6.2.2. Đối với điều này, giao dịch phải được định dạng tốt (**valid**-tx) và áp dụng cho trạng thái UTxO cục bộ được xác nhận hiện tại: $L \circ tx \neq \perp$. Nếu các lần kiểm tra vượt qua, một yêu cầu (**reqTx**, tx) sẽ được gửi đến tất cả các bên.

(**reqTx**). Khi nhận được yêu cầu (**reqTx**, tx), chữ ký chỉ được cấp bởi một bên p_i nếu tx áp dụng cho trạng thái UTxO cục bộ *được xem* của họ: $\hat{L} \circ tx \neq \perp$. Nếu đúng như vậy, cả nhóm sẽ đợi cho đến khi trạng thái UTxO $\bar{L}$ *được xác nhận* đã "bắt kịp": chữ ký chỉ được chuyển giao ngay khi $\bar{L} \circ tx \neq \perp$, tức là một giao dịch chỉ được ký khi nó được áp dụng cho trạng thái được xác nhận cục bộ.

Giao thức Hydra Head được đơn giản hóa mà không có giải quyết xung đột

on $(\texttt{init}, i, \underline{K}_{\text{ver}}, K_{\text{sig}}, U_0)$ *from client*
- $\mathcal{V} \leftarrow \underline{K}_{\text{ver}}$
- $\text{avk} \leftarrow \text{MS-AVK}(\mathcal{V})$
- $\text{sk} \leftarrow K_{\text{sig}}$
- $\hat{s}, \overline{s} \leftarrow 0$
- $\hat{\mathcal{U}}, \overline{\mathcal{U}} \leftarrow \text{snObj}(0, U_0, \emptyset)$
- $\hat{\mathcal{L}}, \overline{\mathcal{L}} \leftarrow U_0$
- $\hat{\mathcal{T}}, \overline{\mathcal{T}} \leftarrow \emptyset$

on $(\texttt{new}, \text{tx})$ *from client*
- **require** $\text{valid-tx}(\text{tx})$ and $\overline{\mathcal{L}} \circ \text{tx} \neq \bot$
- **multicast** $(\texttt{reqTx}, \text{tx})$

on $(\texttt{newSn})$ *for* p_i
- **require** $\text{leader}(\overline{s}+1) = i$ and $\hat{\mathcal{U}} = \overline{\mathcal{U}}$
- $T \leftarrow (\text{maxTxos}(\overline{\mathcal{T}}))^{\downarrow(h)}$
- **multicast** $(\texttt{reqSn}, \overline{s}+1, T)$

on $(\texttt{close})$ *from client*
- **return** $(\overline{\mathcal{U}}.U, \overline{\mathcal{U}}.s, \overline{\mathcal{U}}.\tilde{\sigma}, \overline{\mathcal{T}}^{\downarrow(\text{tx},\tilde{\sigma})})$

on $(\texttt{cont}, \eta)$ *from client*
- $(U_\eta, s_\eta, T_\eta) \leftarrow \eta$
- **if** $\overline{s} \leq s$
  - $U \leftarrow U_\eta$
  - $s \leftarrow s_\eta$
  - $\tilde{\sigma} \leftarrow \varepsilon$
- **else**
  - $U \leftarrow \overline{\mathcal{U}}.U$
  - $s \leftarrow \overline{s}$
  - $\tilde{\sigma} \leftarrow \overline{\mathcal{U}}.\tilde{\sigma}$
- $T' \leftarrow \text{applicable}(U, \overline{\mathcal{T}}^{\downarrow(\text{tx})} \cup T_\eta) \setminus T_\eta$
- **if** $U = U_\eta$
  - $U \leftarrow \varepsilon$
- **return** $(U, s, \tilde{\sigma}, \{t \in \overline{\mathcal{T}}^{\downarrow(\text{tx},\tilde{\sigma})} \mid t.\text{tx} \in T'\})$

---

on $(\texttt{reqTx}, \text{tx})$ *from* p_j
- **require** $\text{valid-tx}(\text{tx}) \wedge \hat{\mathcal{L}} \circ \text{tx} \neq \bot$
- **wait** $\overline{\mathcal{L}} \circ \text{tx} \neq \bot$
  - $h \leftarrow H(\text{tx})$
  - $\hat{\mathcal{T}}[h] \leftarrow \text{txObj}(j, \text{tx})$
  - $\hat{\mathcal{L}} \leftarrow \hat{\mathcal{L}} \circ \text{tx}$
  - **output** $(\texttt{seen}, \text{tx})$
  - $\sigma_i \leftarrow \text{MS-Sign}(\text{sk}, h)$
  - **send** $(\texttt{ackTx}, h, \sigma_i)$ to p_j

on $(\texttt{ackTx}, h, \sigma_j)$ *from* p_j
- **require** $\hat{\mathcal{T}}[h].i = i$
- **require** $\hat{\mathcal{T}}[h].S[j] = \varepsilon$
- $\hat{\mathcal{T}}[h].S[j] \leftarrow \sigma_j$
- **if** $\forall k : \hat{\mathcal{T}}[h].S[k] \neq \varepsilon$
  - $\tilde{\sigma} \leftarrow \text{MS-ASig}(h, \mathcal{V}, \hat{\mathcal{T}}[h].S)$
  - **if** $\tilde{\sigma} \neq \bot$
    - **multicast** $(\texttt{confTx}, h, \tilde{\sigma})$

on $(\texttt{confTx}, h, \tilde{\sigma})$ *from* p_j
- **if** $\text{MS-Verify}(\text{avk}, h, \tilde{\sigma})$
  - $\text{tx} \leftarrow \hat{\mathcal{T}}[h].\text{tx}$
  - $\overline{\mathcal{L}} \leftarrow \overline{\mathcal{L}} \circ \text{tx}$
  - $\hat{\mathcal{T}}[h].\tilde{\sigma} \leftarrow \tilde{\sigma}$
  - $\overline{\mathcal{T}}[h] \leftarrow \hat{\mathcal{T}}[h]$
  - $\hat{\mathcal{T}} \leftarrow \hat{\mathcal{T}} \setminus \hat{\mathcal{T}}[h]$
  - **output** $(\texttt{conf}, \text{tx})$

on $(\texttt{reqSn}, s, T)$ *from* p_j
- **require** $s = \overline{s}+1$ and $\text{leader}(s) = j$
- **wait** $\overline{s} = \hat{s}$ *and* $T \subseteq \overline{\mathcal{T}}^{\downarrow(h)}$
  - $\hat{s} \leftarrow \hat{s}+1$
  - $\hat{\mathcal{U}} \leftarrow \text{snObj}(\hat{s}, \overline{\mathcal{U}}.U, T)$
  - $\sigma_i \leftarrow \text{MS-Sign}(\text{sk}, \hat{\mathcal{U}}.h \| \hat{s})$
  - **send** $(\texttt{ackSn}, \hat{s}, \sigma_i)$ to p_j

on $(\texttt{ackSn}, s, \sigma_j)$ *from* p_j
- **require** $s = \hat{s}$ and $\text{leader}(s) = i$
- **require** $\hat{\mathcal{U}}.S[j] = \varepsilon$
- $\hat{\mathcal{U}}.S[j] \leftarrow \sigma_j$
- **if** $\forall k : \hat{\mathcal{U}}.S[k] \neq \varepsilon$
  - $\tilde{\sigma} \leftarrow \text{MS-ASig}(\hat{\mathcal{U}}.h \| s, \mathcal{V}, \hat{\mathcal{U}}.S)$
  - **if** $\tilde{\sigma} \neq \bot$
    - **multicast** $(\texttt{confSn}, s, \tilde{\sigma})$

on $(\texttt{confSn}, s, \tilde{\sigma})$ *from* p_j
- **require** $s = \hat{s} \neq \overline{s}$
- **if** $\text{MS-Verify}(\text{avk}, \hat{\mathcal{U}}.h \| \hat{s}, \tilde{\sigma})$
  - $\overline{s} \leftarrow s$
  - $\hat{\mathcal{U}}.\tilde{\sigma} \leftarrow \tilde{\sigma}$
  - $\overline{\mathcal{U}} \leftarrow \hat{\mathcal{U}}$
  - $\overline{\mathcal{T}} \leftarrow \overline{\mathcal{T}} \setminus \text{Reach}^{\overline{\mathcal{T}}}(\overline{\mathcal{U}}.T)$

Hình 10: Máy giao thức Head cho *giao thức đơn giản không có giải quyết xung đột* theo quan điểm của bên p_i.

Trong trường hợp các điều kiện tiên quyết được thỏa mãn, một đối tượng giao dịch tương ứng sẽ được phân bổ, khởi tạo và thêm vào $\hat{T}$; $\hat{L}$ được cập nhật bằng cách áp dụng tx, và (**seen**, tx) là đầu ra; và cuối cùng, chữ ký trên Hash của tx, σ = **MS-Sign**(H(tx)), được gửi lại cho người phát hành giao dịch bằng cách trả lời bằng (**ackTx**, H(tx), σ).

(**ackTx**). Khi nhận được xác nhận (**ackTx**, h, σ_j), người phát hành giao dịch lưu trữ chữ ký nhận được trong đối tượng giao dịch tương ứng. Nếu chữ ký của mỗi bên đã được thu thập, p_i sẽ tính toán đa chữ ký $\tilde{\sigma}$ và nếu hợp lệ, sẽ gửi nó cho tất cả các bên trong một thông điệp (**confTx**, h, $\tilde{\sigma}$).

(**confTx**). Khi nhận được xác nhận (**confTx**, h, $\tilde{\sigma}$) từ người phát hành giao dịch, có đa chữ ký hợp lệ, đa chữ ký được lưu trữ trong đối tượng giao dịch tương ứng, $\bar{L}$ được cập nhật bằng cách áp dụng tx, và đối tượng giao dịch được chuyển từ $\hat{T}$ sang $\bar{T}$. Cuối cùng, (**conf**, tx) là đầu ra.

Tạo ảnh chụp nhanh. Song song với việc xác nhận giao dịch, các bên tạo ảnh chụp nhanh theo cách thức vòng lặp tuần tự nghiêm ngặt. Chúng tôi gọi bên chịu trách nhiệm phát hành ảnh chụp nhanh thứ i là *lãnh đạo* của ảnh chụp nhanh thứ i. Tần suất phát hành của ảnh chụp nhanh điều chỉnh sự cân bằng giữa không gian giao dịch mà các bên phải duy trì để lưu trữ các giao dịch đã được xác nhận nhưng được xử lý nhanh so với chi phí giao tiếp ảnh chụp nhanh trong giao thức Head. Vì thông tin trao đổi giữa các bên để xác nhận ảnh chụp nhanh là ít, về nguyên tắc, ảnh chụp nhanh như vậy có thể được phát hành một cách tham lam ngay khi lãnh đạo ảnh chụp nhanh tiếp theo nhìn thấy một giao dịch mới được xác nhận.

(**newSn**). Khi kích hoạt bởi (**newSn**), lãnh đạo ảnh chụp nhanh xác minh xem $\hat{U} = \bar{U}$ để đảm bảo rằng anh ta chưa ở trong quá trình tạo ảnh chụp nhanh. Người lãnh đạo p_i sau đó thông báo tập hợp giao dịch $\bar{T}$ các giao dịch đã xác nhận chưa được xử lý ảnh chụp nhanh sẽ được áp dụng để tính toán ảnh chụp nhanh mới. Tuy nhiên, để giảm chi phí giao tiếp, chỉ các Hash của các giao dịch *tối đa* của $\bar{T}$ được công bố là các giao dịch của $\bar{T}$ không được tham chiếu bởi một giao dịch khác trong $\bar{T}$. Tập tối đa này được tính bằng hàm T = **maxTxos** $(\bar{T})^{\downarrow(h)}$. Cuối cùng người lãnh đạo gửi (**reqSn**, $\bar{s}$ +1, T) cho tất cả các bên.

(**reqSn**). Khi nhận được yêu cầu (**reqSn**, s, T), bên p_i sẽ kiểm tra xem s có phải là số ảnh chụp tiếp theo hay không và p_j có trách nhiệm chỉ đạo việc tạo ra nó. Bên p_i sau đó

đợi cho đến khi ảnh chụp nhanh trước đó được xác nhận ($\bar{s} = \hat{s}$) và tất cả các giao dịch được đề cập trong T được xác nhận.

Sau đó p_i tăng bộ đếm ảnh chụp nhanh đã thấy của mình và cấp phát một đối tượng ảnh chụp nhanh mới gọi hàm **snObj** thực hiện các bước sau:

1. Nó cấu trúc lại bộ giao dịch được áp dụng cho ảnh chụp nhanh được xác nhận mới nhất bằng cách gọi hàm $\mathbf{Reach}^{\bar{T}}(T)$ tính toán tất cả các giao dịch trong $\bar{T}$ có thể truy cập được từ các giao dịch (với Hash) trong T bằng cách tuân theo các tham chiếu đầu ra (nghịch đảo của **maxTxos**); và
2. Tính toán bộ UTxO của ảnh chụp nhanh mới dưới dạng $\widehat{U}.U \leftarrow \bar{U}.\mathrm{U} \circ \mathbf{Reach}^{\bar{T}}(T)$, và
3. Tính toán Hash của $\widehat{U}.U$ và đặt các trường cho số ảnh chụp nhanh và các giao dịch tối đa được áp dụng.

Cuối cùng, p_i tính toán một chữ ký σ_i = **MS-Sign**(**sk**, $H(\widehat{U})||\hat{s}$), và trả lời cho p_j thông điệp (**ackSn**, $\bar{s}$, σ_i).[5]

(**ackSn**). Khi nhận được thông báo xác nhận (**ackSn**, s, σ_j), lãnh đạo ảnh chụp nhanh sẽ lưu trữ chữ ký nhận được trong đối tượng ảnh chụp nhanh tương ứng. Nếu chữ ký của mỗi bên đã được thu thập, p_i sẽ tính toán đa chữ ký $\tilde{\sigma}$ và nếu hợp lệ, sẽ gửi nó cho tất cả các bên trong một thông điệp (**confSn**, h, $\tilde{\sigma}$).

(**confSn**). Khi nhận được xác nhận (**confSn**, s, $\tilde{\sigma}$) từ lãnh đạo ảnh chụp nhanh có chứa đa chữ ký, p_i lưu trữ đa chữ kỹ và cập nhật $\bar{s} = s$ và $\bar{U} = \widehat{U}$. Cuối cùng, tập hợp các giao dịch đã xác nhận được giảm bớt bằng cách loại trừ các giao dịch đã được xử lý bởi $\bar{U}$: $\bar{T} \leftarrow \bar{T} \setminus \mathbf{Reach}^{\bar{T}}(\bar{U}.T)$.

Đóng Head.

(**close**). Để đóng Head, một bên tạo ra sự kiện (**close**) sẽ trả về ảnh chụp nhanh $\bar{U}.U$ đã xác nhận, số ảnh chụp nhanh $\bar{U}.s$ và đa chữ ký tương ứng $\bar{U}.\tilde{\sigma}$, cùng với các giao dịch được xác nhận còn lại $\bar{T}^{\downarrow(\mathrm{tx},\tilde{\sigma})}$ (đa chữ ký). Các mục này tạo thành chứng chỉ ξ sẽ được đăng trên chuỗi (xem Phần 6.3.2).

[5]Lưu ý rằng không có bộ UTxO nào phải được trao đổi trong quá trình này vì các bên có thể tính toán cục bộ một ảnh chụp nhanh mới bằng các Hash giao dịch đã cho.

(**cont**). Để tranh chấp trạng thái **closed** hiện tại trên chuỗi chính, một bên tạo ra sự kiện (**cont**, η) với đầu vào η là trạng thái Head được quan sát mới nhất đã được tổng hợp trên chuỗi cho Head cho đến nay (theo một chuỗi các giao dịch *đóng* và *tranh chấp*).

Sau đó, thuật toán tính toán dữ liệu "khác biệt" giữa trạng thái Head On-Chain hiện tại và chế độ xem đã xác nhận của người tranh chấp: ảnh chụp nhanh được xác nhận mới nhất (nếu mới hơn so với On-Chain) và tập hợp các giao dịch đã xác nhận (theo quan điểm của anh ta) chưa được xem xét bởi trạng thái hiện tại η. Các mục này tạo thành chứng chỉ ξ sẽ được đăng trên chuỗi (xem Phần 6.3.2).

Chúng tôi chỉ muốn chuyển các giao dịch (đã ký nhiều lần) trong $\bar{T}^{\downarrow(\mathrm{tx})} \setminus T_\eta$ chưa được xử lý bởi ảnh chụp nhanh mới nhất U. Điều này đạt được bằng cách áp dụng hàm **applicable** cho các thử nghiệm, đối với mỗi giao dịch trong $\mathrm{tx} \in \bar{T}^{\downarrow(\mathrm{tx})} \cup T_\eta$ theo thứ tự thích hợp, cho dù $U \circ \mathrm{tx} \neq \perp$ vẫn có thể áp dụng được. Hãy lưu ý rằng các giao dịch trong T_η phải được xem xét trong quá trình này vì một số giao dịch trong $\bar{T}$ có thể phụ thuộc trực tiếp vào chúng, và nếu không sẽ không được phát hiện là có thể áp dụng được. Vì chúng tôi chỉ muốn trích xuất dữ liệu "khác biệt", các giao dịch trong T_η cuối cùng bị xóa một lần nữa vì chúng đã được ghi lại ở trạng thái (tích lũy) η.

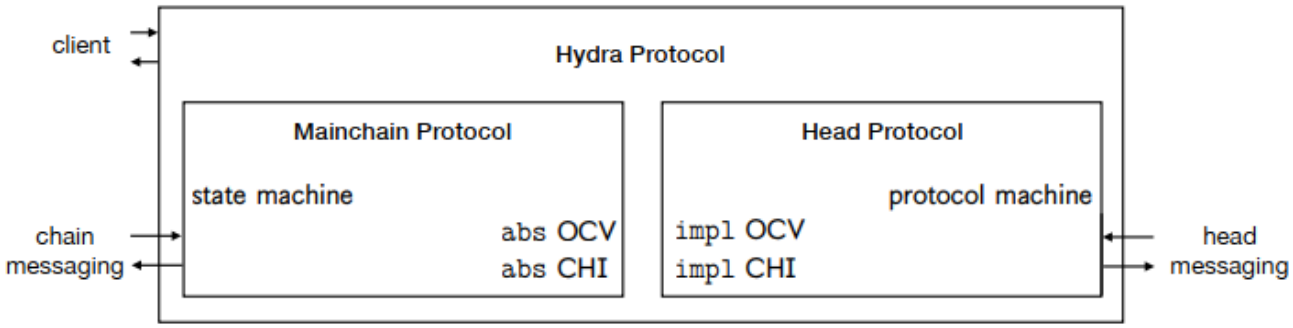
6.3. Chức năng chuỗi chính dành riêng cho từng Head

Ở cấp độ trừu tượng, như được mô tả trong Phần 5, chức năng chuỗi chính và chức năng Head được phân tách rõ ràng thành các sự kiện xảy ra trên chuỗi và các sự kiện xảy ra trong Head. Đặc biệt, những người tham gia mạng lưới không phải là thành viên của giao thức Head chỉ quan sát các sự kiện chuỗi chính.

Tuy nhiên, tùy thuộc vào việc triển khai cụ thể của quy trình chứng nhận Head (mà mô tả tóm tắt của chúng tôi về chức năng chuỗi chính là bất khả tri), một số chức năng chuỗi chính phải được điều chỉnh cho phù hợp với lựa chọn cụ thể cho giao thức Head. Điều này liên quan đến hai khía cạnh:

Xác minh On-Chain (OCV). Việc xác nhận các sự kiện của Head bằng sơ đồ đa chữ ký phải được xác minh trên chuỗi; và do đó, các hoạt động chính xác của chứng nhận Head phải được biết đến với giao thức chuỗi chính. Bây giờ, do phần On-Chain của giao thức chuỗi chính (tức là xác thực chuyển đổi máy trạng thái) được thực hiện bởi các tập lệnh trình xác thực EUTxO, các tập lệnh này sử dụng giao diện trừu tượng của OCV. Do đó, chúng tôi giải thích mã Code OCV của Head là triển khai đặc tả OCV chuỗi chính trừu tượng cho tất cả những người tham gia mạng lưới (Hình 11).

Tương tác chuỗi / Head (CHI). Khi quan sát các sự kiện On-Chain nhất định, chức năng chuỗi chính của thành viên Head phải tương tác với giao thức Head. Ví dụ: trường hợp này xảy ra khi một thành viên Head quan sát việc đóng Head trên chuỗi chính. Sau đó, chức năng chuỗi chính phải truy vấn giao thức Head để biết liệu giao dịch *tranh chấp* có phải được đăng hay không.



Hình 11: Các thành phần của giao thức Hydra.

6.3.1. Xác minh On-Chain (OCV)

Hãy nhớ lại rằng chức năng chuỗi chính được mô tả một cách chung chung là thuật ngữ η, trạng thái Head mới nhất được biết đến trên chuỗi và ξ, một chứng chỉ được đăng bởi một thành viên Head để cập nhật η bằng cách cung cấp thông tin được xác nhận bởi Head.

Chúng tôi tóm tắt ngắn gọn các hoạt động trừu tượng của OCV. Sau khi xử lý giao dịch *collectCom*, bộ UTxO ban đầu được lưu trữ dưới dạng η liên quan đến trạng thái **open**. Sau đó, một bên p_i có thể

- Tạo ra một chứng chỉ ξ được **Close**$(\cdot)$ chấp nhận để đóng Head và làm cho (quan điểm của anh ấy về) bộ UTxO hiện tại của Head có sẵn trên chuỗi chính và,
- Với trạng thái hiện tại η trên chuỗi chính, hãy xuất trình chứng chỉ ξ được **Contest**$(\eta,\cdot)$ để phản đối việc đóng và cung cấp chế độ xem cập nhật của bộ UTxO của Head cho chuỗi chính.

Cuối cùng, hàm **Final** kiểm tra bộ UTxO trong giao dịch để chuyển máy trạng thái sang trạng thái cuối cùng của nó so với thông tin được lưu trữ trong η.

Giờ đây, chúng tôi khởi tạo chức năng xác minh On-Chain (OCV) tương ứng cho giao thức Head được đưa ra trong phần này với cách xác nhận cụ thể của các trạng thái Head (xem Hình 12).

Các thuật toán để xác minh On-Chain

```
Initial (U_{p_1}, ..., U_{p_n})
    return (U_{p_1} ∪ ··· ∪ U_{p_n}, 0, ∅)

Close (K_agg, η, ξ)
    (U, s, σ̃, T) ← ξ
    if ∃(tx_i, σ̃_i) ∈ T :
      ¬MS-AVerify(K_agg, H(tx_i), σ̃_i)
        return ⊥
    if s = 0
        (U, ·, ·) ← η
    else if ¬MS-AVerify(K_agg, H(U)‖s, σ̃)
        return ⊥
    if U ∘ T^{↓(tx)} = ⊥
        return ⊥
    return (U, s, T^{↓(tx)})

Contest (K_agg, η, ξ)
    (U_η, s_η, T_η) ← η
    (U, s, σ̃, T) ← ξ
    if ∃(tx_i, σ̃_i) ∈ T : ¬MS-AVerify(K_agg, H(tx_i), σ̃_i
        return ⊥
    if s ≤ s_η
        U_N ← U_η
    else
        U_N ← U
        if ¬MS-AVerify(K_agg, H(U)‖s, σ̃) return ⊥
        T_η ← applicable(U_N, T_η)
    if U_N ∘ (T_η ∪ T^{↓(tx)}) = ⊥
        return ⊥
    return (U_N, s, T_η ∪ T^{↓(tx)})

Final (η, U)
    (U_η, s_η, T_η) ← η
    return (U = U_η ∘ T_η)
```

Hình 12: Các thuật toán được sử dụng bởi máy trạng thái để xác minh On-Chain.

Initial. Toàn bộ bộ UTxO ban đầu U_0 được tạo từ n bộ UTxO đã cam kết U_{p1}, ..., U_{pn} và được trả về dưới dạng $\eta = (U_0, 0, \emptyset)$.

Close. Máy trạng thái sử dụng thuật toán xác minh On-Chain (OCV) **Close** để xác minh thông tin do các bên gửi.

Hãy nhớ rằng, khi một p_i nhận được lệnh **close**, nó chỉ cần xuất ra dưới dạng chứng chỉ số ảnh chụp nhanh, bộ UTxO và đa chữ ký tự ứng với ảnh chụp nhanh $\overline{U}$ được xác nhận gần nhất, cũng như tất cả các giao dịch đã xác nhận trong $\overline{T}$ chưa được xem xét trong $\overline{U}$, cùng với đa chữ ký tương ứng.

Hàm OCV **Close** (xem Hình 12) xác minh tất cả đa chữ ký trong $\xi = (U, s, \tilde{\sigma}, T)$, tức là của $H(U) \parallel s$ và các giao dịch trong T, và đảm bảo rằng các giao dịch trong T có thể được áp dụng cho U (hoặc, trong trường hợp $s = 0$, cho U_0). Sau đó, thuật toán đưa ra trạng thái mới $\eta' = (U, s, T^{\downarrow(tx)})$.

Contest. Máy trạng thái sử dụng thuật toán OCV **Contest** để xác minh dữ liệu "khác biệt" do một bên tranh chấp gửi.

Hãy nhớ lại rằng, khi một p_i nhận được lệnh (**cont**, η) với $\eta = (U_\eta, s_\eta, T_\eta)$, anh ta cung cấp ảnh chụp nhanh mới nhất $\overline{U}.U$ nếu nó mới hơn U_η và những giao dịch đã xác nhận đó chưa được xem xét bởi ảnh chụp nhanh mới nhất. Trong trường hợp U_η mới hơn ảnh chụp nhanh của chính mình, bạn có thể tìm thấy các giao dịch chưa được phân phối bằng cách thử áp dụng chúng (cùng với T_η) cho U_η, vì những giao dịch đã được xem xét bởi U_η không thể áp dụng được nữa; tính toán này được thực hiện bởi hàm **applicable**.

Tương tự như trường hợp đóng, hàm OCV **Contest**, cho $\xi = (U, s, \tilde{\sigma}, T)$, đầu tiên sẽ kiểm tra tất cả các chữ ký.

Trong trường hợp ảnh chụp nhanh U được cung cấp mới hơn ảnh chụp nhanh U_η từ trạng thái On-Chain η, tập hợp T_η được giảm xuống cho các giao dịch vẫn áp dụng cho giao dịch mới hơn của cả hai ảnh chụp nhanh, U_N.

Cuối cùng, đảm bảo rằng $T_\eta \cup T^{\downarrow(\mathrm{tx})}$ có thể được áp dụng cho cả hai ảnh chụp nhanh mới nhất và trạng thái mới (tổng hợp) $\eta' = (U_N, s, T_\eta \cup T^{\downarrow(\mathrm{tx})})$ là đầu ra.

Final. Cho $\eta = (U_\eta, s_\eta, T_\eta)$ và U, **Final** kiểm tra rằng $U = U_\eta \circ T_\eta$.

Tương tác chuỗi / Head

on (clientTx, tx)
  head.(new, tx)

on (clientClose)
  $\xi \leftarrow$ head.(close)
  chain.postTx($close, \xi$)

on (chainInitial)
  require $K^{\mathsf{chain}}_{\mathsf{agg}} = K^{\mathsf{e}}_{\mathsf{agg}}$
  require $h_{\mathsf{MT}} = H_{\mathsf{Me\ kle}}(\underline{k}_{\mathsf{e}})$
  chain.postTx($commit, U$)

on (chainInitialTimeOut)
  if *(all members committed)*
    chain.postTx($collectCom$)
  else
    chain.postTx($abort$)

on (chainCollectCom)
  $(U_0, \cdot, \cdot) \leftarrow \mathsf{Initial}(U_{p_1}, \ldots, U_{p_n})$
  head.(init, $i, \underline{K}_{\mathsf{e}}, K_{\mathsf{ig},i}, U_0$)

on (chainClose)
  $\eta' = (U', s', T') \leftarrow$ chain.Close($K_{\mathsf{agg}}, \eta, \xi$)
  $\xi = (U, s, \tilde{\sigma}, T) \leftarrow$ head.(cont, η')
  if $s > s' \ \vee \ T \neq \emptyset$
    chain.postTx($contest, \xi$)

on (chainContest)
  $\eta' = (U', s', T') \leftarrow$ chain.Contest($K_{\mathsf{agg}}, \eta, \xi$)
  $\xi = (U, s, \tilde{\sigma}, T) \leftarrow$ head.(cont, η')
  if $s > s' \ \vee \ T \neq \emptyset$
    chain.postTx($contest, \xi$)

on (chainClosedTimeOut)
  chain.postTx($fanout$)

Hình 13: Tương tác chuỗi / Head: Các hành động chuỗi chính bổ sung cho các thành viên Head.

6.3.2. Tương tác chuỗi / Head (CHI)

Trong hình 13, chúng tôi tóm tắt một phần của chức năng chuỗi chính Hydra tương tác với thành viên Head (khách hàng - Client) và giao thức Head.

Quy trình **clientTx** xử lý yêu cầu của khách hàng để đưa ra giao dịch Head bằng cách ủy quyền yêu cầu cho giao thức Head. **ClientClose** thường xuyên xử lý yêu cầu của khách hàng để đóng Head. Nó tập hợp một chứng chỉ cho trạng thái cục bộ hiện tại từ giao thức Head và đăng chứng chỉ này trên chuỗi.

Quy trình **chainInitial** được kích hoạt khi nhìn thấy giao dịch *ban đầu* của Head trên chuỗi. Nó xác minh các tham số được ghi lại trong giao dịch *ban đầu* so với các tham số được thu thập trong giai đoạn thiết lập được mô tả trong Phần 4: cụ thể là, khóa đa chữ ký tổng hợp phải khớp và h_{MT} phải là Hash cây Merkle của các khóa xác minh được tập hợp $\underline{k}_{\mathrm{ver}}$. Nếu thành công, bộ UTxO của khách hàng sẽ được cam kết trên chuỗi.

Quy trình **chianInitialTimeOut** được kích hoạt khi thời hạn cam kết ban đầu đã hết. Sau đó, nó đăng một giao dịch *collectCom* chứa tất cả các bộ UTxO đã cam kết trong trường hợp tất cả các thành viên Head đã cam kết một bộ UTxO, hoặc trong trường hợp khác là một giao dịch *huỷ bỏ*.

Quy trình **chainCollectCom** được kích hoạt khi thấy giao dịch *collectCom* của Head trên chuỗi. Nó tính toán, thành U_0, bộ các UTxO đã cam kết và khởi tạo giao thức Head.

Quy trình **chainClose** và **chainContest** được kích hoạt bằng cách quan sát lần lượt các giao dịch *đóng* và *tranh chấp* của Head. Chúng so sánh trạng thái On-Chain mới nhất η với trạng thái của bên Head bằng cách gọi hàm **cont** của giao thức Head để lấy chứng chỉ ξ cho bản cập nhật On-Chain khác biệt để đại diện cho các phần của trạng thái cục bộ chưa được η xem xét. Nếu cần, một giao dịch *tranh chấp* tương ứng sẽ được đăng trên chuỗi.

Quy trình **chainClosedTimeOut** được kích hoạt sau khi hết thời gian tranh chấp. Sau đó, nó đăng một giao dịch *fanout* có chứa bộ UTxO cuối cùng.

6.4. Bằng chứng bảo mật

Phần này chứng minh rằng giao thức Head được trình bày trong Phần 6 đáp ứng tính nhất quán, tính sống động, tính hợp lý và tính hoàn chỉnh. Việc chứng minh tiến hành bằng cách thiết lập một số bất biến tạo điều kiện cho việc chứng minh các tính chất này. Trong suốt quá trình chứng minh, giả thiết được đưa ra rằng tối đa $n - 1$ thành viên Head bị hỏng. Hơn nữa, giả sử không có chữ ký nào bị giả mạo và không có Hash xung

đột; những sự kiện này chỉ xảy ra với xác suất không đáng kể. Hãy xem xét các biến ngẫu nhiên sau:

- SN_j : bộ UTxO tương ứng với ảnh chụp nhanh thứ j, tức là tập hợp lấy đa chữ ký thứ j trên ảnh chụp nhanh ($\mathrm{SN}_0 = U_0$);
- $\tilde{T}_j$: bộ giao dịch tương ứng với SN_j, được xác định chính thức thông qua $\tilde{T}_0 = \emptyset$ và $\tilde{T}_j := \tilde{T}_{j-1} \circ \mathbf{Reach}^{\bar{T}}(T)$ trong đó T là bộ được đề xuất trong (**reqSn**, j, T);
- C_{chain} : theo dõi "các giao dịch trên chuỗi" và được định nghĩa như sau: khi (thành công) đóng lại tương ứng tranh chấp với ξ cho η, giả sử $C_{chain} \leftarrow \tilde{T}_s \cup T$, trong đó $(\cdot, s, T)$ là đầu ra của **Close**(K_{agg}, η, ξ) tương ứng với **Contest**(K_{agg}, η, ξ);
- $\mathrm{SN}_{\mathrm{cur},i}$: ảnh chụp nhanh mới nhất được xác nhận bởi bên p_i.

Bổ đề 1 (Tính nhất quán). *Giao thức Head cơ bản đáp ứng thuộc tính nhất quán.*
Chứng minh. Tuân thủ $\bar{C}_i \cup \bar{C}_j \subseteq \hat{S}_i$ vì không có giao dịch nào có thể được xác nhận mà không có bất kỳ bên trung thực nào ký tên vào nó. Vì các bên không ký các giao dịch xung đột nên $U_0 \circ \hat{S}_i \neq \bot$. Vì vậy, $U_0 \circ (\bar{C}_i \cup \bar{C}_j) \neq \bot$.

Bất biến 1. *Xem xét việc thực thi giao thức Head cơ bản không có xung đột khi có kẻ tấn công mạng lưới. Sau đó, đối với bất kỳ đầu vào giao dịch* tx *nào vào giao thức thông qua* (**new**) *các điều khoản sau sẽ được lưu giữ đối với mọi bên p_i và p_j :*

$$\forall t_0 : \ \overline{\mathcal{L}}_i^{(t_0)} \circ \mathrm{tx} \neq \bot \Rightarrow \exists T \geq t_0 \forall t \geq T : \ \overline{\mathcal{L}}_j^{(t)} \circ \mathrm{tx} \neq \bot \ \vee \ \mathrm{tx} \in \overline{C}_j^{(t)}$$

trong đó chỉ số trên $\cdot^{(t)}$ cho biết thời gian mà biến tương ứng được đánh giá.
Chứng minh. Giả sử rằng bên p_i thấy tx tại thời điểm t_0 và $\overline{\mathcal{L}}_i^{(t_0)} \circ \mathrm{tx} \neq \bot$. Bằng cách không có xung đột và phân phối đầy đủ, chúng tôi nhận được rằng cuối cùng, mỗi bên p_j nắm giữ $\overline{C}_j^{(t)} \supseteq \overline{C}_i^{(t_0)}$. Tại thời điểm t, $\overline{\mathcal{L}}_j^{(t)} \circ \mathrm{tx} \neq \bot$ hoặc $\mathrm{tx} \in \overline{C}_j^{(t)}$ (vì chúng ta không có xung đột, và $\overline{C}_j^{(t)} \subseteq \mathcal{N}$).

Bổ đề 2 (Tính sống động không xung đột). *Giao thức head cơ bản đạt được tính sống động không xung đột.*
Chứng minh. Chúng tôi chứng minh rằng giao dịch tx do người chơi p_i phát hành cuối cùng sẽ được xác nhận bởi mỗi người chơi p_j. Theo tính không xung đột, trong (**new**, tx), chúng ta có $\bar{L}_i \circ \mathrm{tx} \neq \bot$.
Giả sử rằng $\mathrm{tx} \notin \bar{C}_j$, tức là p_j chưa thấy tx được xác nhận. Ngay sau khi p_j đi vào (hoặc được kích hoạt lại từ hàng đợi) (**reqTx**, tx) với điều kiện $\bar{L}_i \circ \mathrm{tx} \neq \bot$ (cuối cùng được

đảm bảo bởi Bất biến 1), bởi tính không xung đột, cũng như $\bar{L}_i \circ \text{tx} \neq \perp$, và p_j thừa nhận giao dịch. Do đó, mọi p_j cuối cùng đều thừa nhận giao dịch và $\text{tx} \in \cap_{i \in [n]} \bar{C}_i$.

Bất biến 2. *Xét một bên không bị gián đoạn tùy ý* p_i. *Gọi* $\tilde{T}$ *là tập hợp tương ứng với* $\text{SN}_{\text{cur},i}$. *Khi đó,* $\tilde{T} \cup \bar{T}_i = \bar{C}_i$, *trong đó* $\bar{T}_i$ *là tập hợp* $\bar{T}$ *của* p_i.
Chứng minh. Quan sát thấy rằng bất biến được thỏa mãn một cách đáng kể khi bắt đầu thực thi giao thức. Hơn nữa, mỗi khi một giao dịch mới được xác nhận qua **confTx**, cả $\bar{T}_i$ và $\bar{C}_i$ đều tăng theo giao dịch mới được xác nhận, trong khi $\tilde{T}$ không thay đổi.

Lần khác duy nhất một trong các bộ $\tilde{T}$, $\bar{T}_i$ hoặc $\bar{C}_i$ thay đổi là khi một ảnh chụp nhanh mới được xác nhận qua **confSn**. Trong trường hợp như vậy, hãy lưu ý rằng C_i giữ nguyên trong khi bất kỳ giao dịch nào bị xóa khỏi $\bar{T}_i$ đều được xem xét bởi ảnh chụp nhanh mới và do đó được thêm vào $\tilde{T}$. Do đó, bất biến vẫn thoả mãn.

Bất biến 3. $\tilde{T}_0 \subseteq \tilde{T}_1 \subseteq \tilde{T}_2 \subseteq \ldots$.
Chứng minh. Cho p_i là một bên trung thực. Có thể dễ dàng nhận thấy rằng tập hợp các giao dịch được xem xét bởi ảnh chụp nhanh mới luôn bao gồm tập hợp được xem xét bởi ảnh chụp nhanh trước đó kể từ tập hợp các giao dịch T trong một **reqSn** thỏa mãn rằng $\text{SN}_{\text{cur},i} \circ \mathbf{Reach}^{\bar{T}i}(T) \neq \perp$, (điều này được thể hiện bởi Bất biến 2).

Bất biến 4. C_{chain} *phát triển đơn điệu (w.r.t* $\subseteq$*).*
Chứng minh. Xét hoạt động **Contest**(K_{agg}, η, ξ) và cho $\eta = (U_\eta, s_\eta, T_\eta)$ và $\xi = (U, s, \tilde{\sigma}, T)$. Lưu ý rằng trước khi hoạt động $C_{\text{chain}} = \tilde{T}_{s\eta} \cup T_\eta$. Bây giờ hãy xem xét tập T^* trong đầu ra $(\cdot, \cdot, T^*)$ của **Contest**. Lưu ý rằng sau hoạt động $C_{\text{chain}} = \tilde{T}_s \cup T^*$. Quan sát thấy:

- Vì $s \geq s_\eta$, Bất biến 3 thể hiện rằng một giao dịch $\text{tx} \in \tilde{T}_{s\eta}$ cũng nằm trong $\tilde{T}_s$.
- Nếu một giao dịch $\text{tx} \in T_\eta$ không nằm trong T^*, thì $s > s_\eta$ và giao dịch được sử dụng bởi ảnh chụp nhanh với số s, tức là $\text{tx} \in \tilde{T}_s$.

Do đó, C_{chain} phát triển đơn điệu.

Bất biến 5. *Với mọi* $i \in H_{\text{cont}}$, $\bar{C}_i \subseteq C_{\text{chain}}$.
Chứng minh. Lấy bất kỳ bên trung thực p_i nào và đặt $\tilde{s}$ là số ảnh chụp nhanh hiện tại ở p_i, tức là $\text{SN}_{\text{cur},i} = \tilde{T}_{\tilde{s}}$. Hãy nhớ rằng, theo Bất biến 2, $\bar{C}_i = \tilde{T}_{\tilde{s}} \cup \bar{T}_i$. Hãy xem xét một hoạt động đóng hoặc tranh chấp của p_i cũng như đầu ra (U, s, T) của **Contest**, và quan sát thấy rằng sau hoạt động $C_{\text{chain}} = \tilde{T}_s \cup T^*$. Bởi Bất biến 3, $\tilde{T}_{\tilde{s}} \subseteq \tilde{T}_s$ và, bằng một lập luận tương tự như trong chứng minh Bất biến 4, nếu $\text{tx} \in \bar{T}_i$ không thuộc T, nó phải ở trong

$\tilde{T}_s$. Do đó, $\bar{C}_i \subseteq C_{\text{chain}}$. Hơn nữa, vì C_{chain} phát triển đơn điệu (Bất biến 4), bất biến vẫn được thỏa mãn.

Bất biến 6. *Với mọi bên không bị gián đoạn* p_i, $\cup_{j\in[n]} \bar{C}_j \subseteq \hat{S}_i$.
Chứng minh. Các bên trung thực sẽ chỉ xuất ra (**conf**, tx) nếu tồn tại đa chữ ký hợp lệ cho tx, điều này thể hiện rằng mỗi bên trung thực sẽ xuất ra (**saw**, tx) ngay trước khi họ ký tx.

Bất biến 7. *Với j bất kỳ,* $\tilde{T}_j \subseteq \cap_{i\in H} \bar{C}_i$.
Chứng minh. Chỉ những giao dịch đã được xác nhận bởi tất cả các bên trung thực mới có thể được đưa vào ảnh chụp nhanh đã xác nhận.

Bất biến 8. $C_{\text{chain}} \subseteq \cap_{i\in H} \hat{S}_i$.
Chứng minh. Cho $\eta = (U, s, T)$. Chúng ta thấy $C_{\text{chain}} = \tilde{T}_s \cup T$. Hãy xem xét một giao dịch tx $\in C_{\text{chain}}$.

- Nếu tx $\in \tilde{T}_s$ thì tx $\in \cap_{i\in H} \bar{C}_i \subseteq \cap_{i\in H} \hat{S}_i$ bởi Bất biến 7 và 6.
- Nếu tx $\in T$ thì tx $\in \cap_{i\in H} \hat{S}_i$ vì không có giao dịch nào có thể được xác nhận mà không bị tất cả các bên trung thực nhìn thấy.

Bổ đề 3 (Tính hợp lý). *Giao thức Head cơ bản đáp ứng tính hợp lý.*
Chứng minh. Gọi $\eta = (U, s, T)$ là giá trị của η ngay trước khi áp dụng **Final**(η, U_{final}). Rõ ràng, tập U_{final} duy nhất sẽ được **Final** chấp nhận là $U_0 \circ (\tilde{T}_s \cup T)$. Theo định nghĩa $\tilde{T}_s \cup T = C_{\text{chain}}$. Tính hợp lý bây giờ theo sau từ bất biến 8.

Bổ đề 4 (Tính hoàn chỉnh). *Giao thức Head cơ bản thỏa mãn tính hoàn chỉnh.*
Chứng minh. Theo dõi từ Bất biến 5 và một đối số tương tự như trong chứng minh Bổ đề 3.

7. Đánh giá thử nghiệm

Bây giờ chúng ta sẽ nghiên cứu hiệu suất của giao thức Hydra về cả độ trễ (thời gian giải quyết giao dịch) và thông lượng (tốc độ xử lý giao dịch, TPS), sử dụng mô phỏng chính xác về thời gian. Các mô phỏng sẽ chứng minh rằng Hydra là tối ưu trong việc giải quyết giao dịch nhanh chóng và chúng tôi sử dụng các *đường cơ sở* để có được cái nhìn sâu sắc về các đặc điểm hiệu suất tốc độ giao dịch của giao thức một cách có hệ thống.

Để xác định giao dịch thanh toán nhanh như thế nào trong Hydra và tốc độ xử lý chúng, chúng ta phải xem xét các yếu tố sau:

Mở và đóng Head. Điều này bao gồm việc tạo và gửi các giao dịch cam kết / ngừng cam kết và đợi cho đến khi chúng được xác nhận là có trên chuỗi.

Hiệu suất của giao thức Head. Với sự phân bố địa lý và dung lượng CPU/mạng lưới của các Node Head, mất bao lâu để trao đổi các thông điệp dẫn đến các giao dịch và ảnh chụp nhanh được xác nhận?

Hạn chế đối với giao dịch đang diễn ra. Khi người chơi muốn gửi hai giao dịch, trong đó một giao dịch sử dụng sự thay đổi từ giao dịch kia, họ phải hoãn gửi giao dịch thứ hai cho đến khi họ có xác nhận cho giao dịch đầu tiên. Hơn nữa, người chơi có thể muốn ngăn chặn quá nhiều giao dịch đã xác nhận, nhưng không phải giao dịch ảnh chụp nhanh để giữ cho ngừng cam kết nhỏ hơn. Đồng thời, điều này giới hạn số lượng giao dịch *đang diễn ra* (đã gửi nhưng chưa được xác nhận) mà bất kỳ Node nào có thể có.

Giá trị có rủi ro. Để giảm thiểu điều này, người chơi có thể đợi một số giao dịch được xác nhận trước khi gửi thêm giao dịch, hạn chế hơn nữa số lượng giao dịch đang diễn ra.

Vì thời gian mở và đóng Head phụ thuộc phần lớn vào giao thức Layer 1 cơ bản và có thể được khấu hao trong suốt thời gian tồn tại của Head, chúng tôi không đề cập đến khía cạnh này trong các mô phỏng của mình. Hơn nữa, để đơn giản hóa các mô phỏng, chúng tôi lập mô hình ảnh hưởng của UTxO hữu hạn bằng cách giới hạn trực tiếp số lượng giao dịch đang diễn ra trên mỗi Node. Do đó, chúng tôi tập trung các mô phỏng vào việc thực thi giao thức Head, như được chỉ rõ trong Hình 10.

7.1. Phương pháp luận

Thiết lập thử nghiệm bao gồm một tập hợp các Node cố định, với băng thông mạng cụ thể cho mỗi Node và vị trí địa lý của mỗi Node sẽ xác định độ trễ mạng giữa mỗi cặp Node. Mỗi Node gửi các giao dịch với một *giao dịch đồng thời* (Transaction Concurrency) cụ thể *c*: nó gửi giao dịch *c* nhanh như tài nguyên của nó cho phép và sau đó gửi một giao dịch khác bất cứ khi nào một trong các giao dịch mà nó đã gửi trước đó được xác nhận. Điều này kiểm soát số lượng giao dịch đang diễn ra là *c* trên mỗi Node. *Ảnh chụp nhanh* được thực hiện thường xuyên: các Node thay phiên nhau tạo

ảnh chụp nhanh, và bất cứ khi nào lãnh đạo hiện tại cũng có ít nhất một giao dịch được xác nhận, nó sẽ tạo ảnh chụp nhanh với tất cả các giao dịch đã xác nhận mà nó biết.

Để đánh giá đúng các kết quả mô phỏng, chúng tôi so sánh nó với các kịch bản cơ sở đủ đơn giản để tạo điều kiện cho các giới hạn hiệu suất lạc quan một cách chính xác. Chúng tôi suy ra các giới hạn đó bằng cách xem xét từng chuỗi sự kiện phải xảy ra để một số giao dịch được xác nhận và tổng hợp thời gian cho mỗi sự kiện trong các chuỗi đó. Đặc biệt, chúng tôi có ba tài nguyên có khả năng giới hạn tỷ lệ giao dịch:

1. *Dung lượng CPU* tại mỗi Node xác định cách thức các giao dịch có thể được xác thực nhanh chóng và các chữ ký được tạo hoặc xác minh;
2. *Băng thông mạng* đến và đi giới hạn số Byte thông điệp có thể được nhận và gửi bởi mỗi Node trong một thời gian nhất định;
3. Mỗi thông điệp giữa hai Node bị trễ bởi *độ trễ mạng* giữa các Node đó.

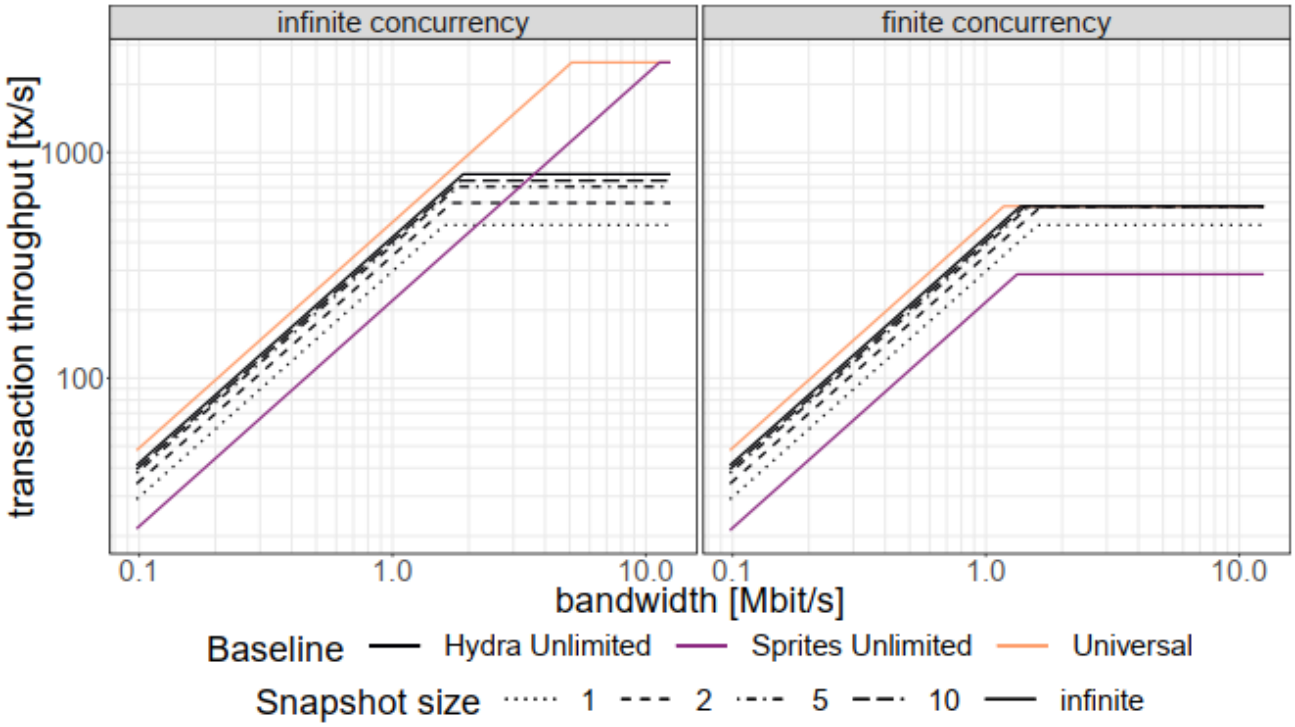
Tùy thuộc vào cấu hình của hệ thống, việc sử dụng tối đa các tài nguyên này sẽ giới hạn tỷ lệ giao dịch. Đây là một sự lý tưởng hóa: trong một quá trình thực thi thực sự của một giao thức, các hiệu ứng tranh chấp sẽ khiến ngay cả tài nguyên khan hiếm nhất cũng bị chặn và thỉnh thoảng không hoạt động. Do đó, chúng tôi mong đợi các kết quả thử nghiệm bị ràng buộc bởi các đường cơ sở và giải thích sự khác biệt là tác động của các hiệu ứng tranh chấp. Chúng tôi xem xét các đường cơ sở sau:

Đường cơ sở chung: Tin cậy hoàn toàn. Để định lượng mức giá mà chúng tôi phải trả cho sự đồng thuận trong Hydra, chúng tôi so sánh các mô phỏng với một kịch bản mà chúng tôi cho rằng sự tin tưởng hoàn hảo giữa tất cả những người tham gia; tức là, chúng tôi chỉ phân phối kiến thức về các giao dịch mà không cố gắng đạt được sự đồng thuận. Trong trường hợp này, các Node gửi giao dịch (sau khi kiểm tra rằng chúng hợp lệ). Các Node khác chỉ xác nhận rằng họ đã nhìn thấy chúng (mà không cần xác thực hoặc ký tên). Chúng tôi vẫn xem xét ảnh hưởng của việc có một giao dịch đồng thời hữu hạn.

Đường cơ sở này đặt giới hạn trên cho thông lượng giao dịch của *bất kỳ* giao thức nào phân phối và xác thực các giao dịch trong một hệ thống phân tán. Hơn nữa, đối với bất kỳ giao thức *đồng thuận* nào, chúng ta nên mong đợi một số chi phí bổ sung (có thể hoặc không thể làm giảm thông lượng thực tế trong các vùng khác nhau của các tham số).

Hydra không giới hạn. Kịch bản này tương tự như giao thức Head, nhưng được thực thi trong các trường hợp lý tưởng, bỏ qua các hiệu ứng tranh chấp như được mô tả ở trên. Trái ngược với việc thực thi giao thức thực, trong đó kích thước ảnh chụp nhanh là một thuộc tính nổi lên tùy thuộc vào tốc độ giao dịch được xác nhận, trong đường cơ sở, chúng tôi có thể kiểm soát trực tiếp số lượng giao dịch được chứa trong ảnh chụp nhanh.

Sprites không giới hạn. Để so sánh với công việc trước đây, chúng tôi cũng bao gồm một đường cơ sở theo cách thực thi tối ưu của giao thức ngoài chuỗi từ [31]. Một sự khác biệt quyết định đối với giao thức Head là trong Sprites, tất cả các Node đều gửi đầu vào của họ cho một lãnh đạo, điều này sẽ đối chiếu chúng và thu thập chữ ký cho toàn bộ các lô giao dịch. So với Hydra, lô này làm giảm nhu cầu về thời gian CPU và số lượng thông điệp, vì ít chữ ký hơn phải được thực hiện và chia sẻ, với chi phí của các vòng lặp mạng bổ sung và sử dụng băng thông mạng cao hơn ở Node lãnh đạo hiện tại, Node này phải gửi theo lô tất cả các giao dịch đến mọi Node khác.



Hình 14: Ví dụ về kịch bản đường cơ sở cho các giao dịch đồng thời hữu hạn và vô hạn.

Chúng tôi đưa ra các ví dụ về đường cơ sở trong Hình 14. Chúng tôi vẽ các kịch bản đường cơ sở khác nhau bằng cách sử dụng các màu khác nhau. Đối với trường hợp Hydra không giới hạn, chúng tôi có nhiều đường, tùy thuộc vào số lượng giao dịch trong mỗi ảnh chụp nhanh; càng nhiều giao dịch được nhóm trong bất kỳ một ảnh chụp nhanh nào, thì chi phí trên *mỗi giao dịch* càng thấp.

Bảng điều khiển bên trái hiển thị giới hạn của giao dịch đồng thời vô hạn. Trong trường hợp đó, thời gian quay vòng của mạng có thể được phân bổ hoàn toàn và không phải là một yếu tố giới hạn. Tỷ lệ giao dịch kết quả có hình dạng đầu gối: nó là tuyến tính trong băng thông mạng miễn là đó là yếu tố giới hạn và sẽ không đổi khi giới hạn từ thời gian CPU chiếm ưu thế. So sánh Hydra không giới hạn và Đường cơ sở chung, chúng ta thấy rằng có một số khác biệt trong vùng băng thông thấp, đó là do đa chữ ký được gửi trong Hydra. Trong khu vực có liên quan đến thời gian CPU, sự khác biệt rõ ràng hơn, do chi phí tính toán của đa chữ kỹ. Nhìn vào đường cơ sở của Sprites không giới hạn, chúng ta thấy sự cân bằng trong các giao dịch theo lô: công việc tính toán được giảm thiểu đáng kể, bằng cách chỉ ký một lô giao dịch lớn duy nhất[6]. Điều này dẫn đến cái giá phải trả là tăng lưu lượng mạng tại Node lãnh đạo, Node này phải gửi mọi giao dịch đến mọi Node khác. Lưu ý rằng trong hình này, chúng tôi chỉ sử dụng một cụm gồm 3 Node; đối với các cụm lớn hơn, nhu cầu về băng thông mạng của Node lãnh đạo thậm chí sẽ còn cao hơn.

Để có hình ảnh chân thực hơn, chúng ta hãy chuyển sang bảng điều khiển bên phải. Ở đây, chúng ta có một giao dịch đồng thời hữu hạn và thời gian vòng quanh mạng đủ lớn để trở thành yếu tố giới hạn (thay vì thời gian CPU) khi chúng ta có đủ băng thông. So sánh Hydra không giới hạn với Đường cơ sở chung, chúng ta thấy rằng cả hai đều đi ngang ở khoảng 580 TPS. Giới hạn về độ trễ mạng là như nhau đối với cả hai đường cơ sở, vì số vòng quay để xác nhận một giao dịch là như nhau (thông điệp lớn hơn đối với Hydra không giới hạn, nhưng điều này chỉ đặt ra yêu cầu cao hơn về băng thông). Điều thú vị là nếu chúng tôi tạo ảnh chụp nhanh cho mỗi giao dịch đơn lẻ, chúng tôi vẫn bị giới hạn bởi sức mạnh của CPU, nhưng ngay khi chúng tôi chỉ tạo ảnh chụp nhanh cho mỗi giao dịch khác, chi phí tạo ra ảnh chụp nhanh đủ nhỏ để không còn quan trọng nữa, so với giới hạn từ độ trễ mạng. Trong hình này, đường cơ sở của Sprites không giới thấp hơn nhiều so với những đường khác. Các yêu cầu về băng thông, đặc biệt là băng thông mạng của Node lãnh đạo lớn hơn nhiều so với các giao thức khác và

[6]Trong giới hạn của giao dịch đồng thời vô hạn, chúng tôi cũng coi kích thước lô trong Sprites là không giới hạn.

việc tập trung các giao dịch trước khi gửi chúng đến mỗi Node cần phải có thêm một vòng lặp.

Hãy lưu ý rằng việc tạo ra một đường cơ sở không giới hạn cho một giao thức nhất định và so sánh nó với một đường cơ sở chung hoặc các giao thức khác, không chỉ có giá trị để đánh giá việc triển khai mà còn là một công cụ để dự đoán hiệu suất có thể có trong giai đoạn thiết kế giao thức.

7.2. Thực hiện

Trong phần sau, chúng tôi sẽ mô tả cách chúng tôi triển khai các mô phỏng cho giao thức Head. Việc triển khai có sẵn tại https://github.com/input-output-hk/hydra-sim.

Chúng tôi lập mô hình các Node Head bằng cách sử dụng các luồng đồng thời trao đổi các thông điệp giao thức từ Hình 10 qua các kênh. Chúng tôi sử dụng thư viện io-sim [2] cho phép viết mã Code đồng thời, sau đó thực thi nó trực tiếp dưới dạng các luồng trong hệ thống thời gian chạy Haskell *hoặc* chạy cùng một mã trong mô phỏng hệ thống thời gian chạy. Cái thứ 2 mang lại một dấu vết thực thi của mã rất nhanh, vì nó trì hoãn một luồng bằng cách chỉ tăng một số đại diện cho thời gian của luồng, thay vì *thực sự* tạm dừng luồng. Như chúng tôi mô tả bên dưới, các mô phỏng sử dụng nhiều độ trễ của luồng, vì vậy điều này cho phép thực hiện mô phỏng nhanh hơn nhiều. Chúng tôi cũng có thể chèn thủ công các điểm theo dõi tại các điểm có liên quan trong giao thức (ví dụ như khi giao dịch được xác nhận). Ví dụ: đo thời gian xác nhận cho một thông điệp, sau đó có thể được thực hiện bằng cách chỉ cần trừ dấu thời gian của các sự kiện "giao dịch được gửi" (**new**) và "giao dịch được xác nhận" (**confTx**).

Hoạt động mật mã. Thay vì sử dụng các hàm mật mã thực cho đa chữ ký, chúng tôi sử dụng các hàm giả không thực hiện bất kỳ phép tính nào, nhưng thay vào đó cho phép độ trễ có thể điều chỉnh được của chuỗi đang thực hiện hoạt động.

Truyền thông điệp. Trước khi được gửi qua mạng, mỗi thông điệp phải được tuần tự hóa và vượt qua giao diện mạng, điều này cần thời gian tuyến tính trong kích thước thông điệp. Vì vậy, sự kiện một thông điệp được gửi bởi một Node không tương ứng với một thời điểm duy nhất, mà là một khoảng thời gian. Chúng tôi tính đến điều đó bằng cách lập mô hình từng thông điệp theo cạnh *đầu* và *cuối* của nó. Khoảng cách thời gian giữa cạnh đầu và cạnh cuối là *độ trễ tuần tự hóa* của một thông điệp được xác định bởi kích thước của nó và băng thông của giao diện mạng của Node. Chúng tôi nắm bắt điều này với một tham số S cho độ trễ trên mỗi Byte. Hơn nữa, chúng tôi tính đến rằng

giao diện mạng chỉ có thể bắt đầu gửi thông điệp tiếp theo *sau khi* cạnh cuối của thông điệp trước đó đã được gửi. Khi mạng đủ bận, đây có thể là một điểm gây tranh cãi.

Chúng tôi mô hình hóa mạng bằng độ trễ *G* giữa mỗi cạnh thông điệp rời khỏi Node gửi và đến Node đích. Tham số *G* được xác định bởi khoảng cách giữa hai Node và không phụ thuộc vào kích thước thông điệp.[7] Chúng tôi sử dụng dữ liệu thực được đo giữa các trung tâm dữ liệu của Amazon Web Services.

Khi cạnh đầu tiên của một thông điệp đến Node nhận, chúng tôi đặt giao diện mạng đến của nó vào trạng thái bận, trong một thời gian cho trước bởi kích thước của thông điệp và băng thông của Node này. Cuối cùng, khi nhận được cạnh cuối, nội dung thông điệp được đặt vào hộp thư đến cục bộ để Node có thể bắt đầu hoạt động trên thông điệp.

Nếu chúng ta chỉ xem xét một thông điệp duy nhất, mô hình này sẽ chỉ dẫn đến độ trễ của toàn bộ thông báo được xác định bởi *G*, kích thước thông điệp và *S* của Node chậm hơn. Nhưng một khi chúng ta có nhiều thông điệp trong hệ thống, nó cũng giải thích chính xác sự tranh cãi tại các điểm kết nối gửi đi và đến. Sự tranh cãi đưa ra phương sai, vì thông điệp có thể phải đợi hoặc không phải đợi ở một trong hai đầu của mạng.

Tối ưu hóa mô phỏng. Chúng tôi đã áp dụng hai cải tiến để tối ưu hóa hiệu suất mà không làm thay đổi tính bảo mật của giao thức. Đầu tiên, khi gửi một giao dịch mới qua Node mới, một Node sẽ xác thực giao dịch và sau đó gửi **reqTx** cho mọi bên, bao gồm cả chính nó. Khi các bên nhận được **reqTx** thì sẽ xác thực lại giao dịch. Đối với Node gửi, điều này là không cần thiết (nó chỉ xác thực cùng một giao dịch), vì vậy chúng tôi bỏ qua xác thực thứ hai trên cùng một Node. Thứ hai, đặc tả của giao thức nói rằng các trình xử lý được thực thi nghiêm ngặt lần lượt. Việc tránh đồng thời theo cách này sẽ đơn giản hóa việc phân tích giao thức. Nhưng có một trường hợp mà chúng ta có thể thực hiện song song các hành động một cách an toàn: khi nhận được **reqTx** (và tương tự là **reqSn**), một Node sẽ xác thực giao dịch hoặc ảnh chụp nhanh so với trạng thái cục bộ của nó và nếu thích hợp, hãy ký tên và trả lời. Hành động ký không truy cập trạng thái của Node, vì vậy chúng ta có thể thực hiện đồng thời với việc xử lý các sự kiện tiếp theo một cách an toàn.

[7]Các thông điệp trong giao thức Hydra đủ nhỏ để bỏ qua các hiệu ứng cửa sổ TCP sẽ gây ra sự phụ thuộc vào kích thước thông điệp.

Đây là những thay đổi khá nhỏ, mà bất kỳ triển khai cụ thể nào cũng sẽ áp dụng, vì vậy chúng tôi cảm thấy cần phản ánh chúng trong mô phỏng cũng như trong đường cơ sở.

7.3. Kết quả thử nghiệm

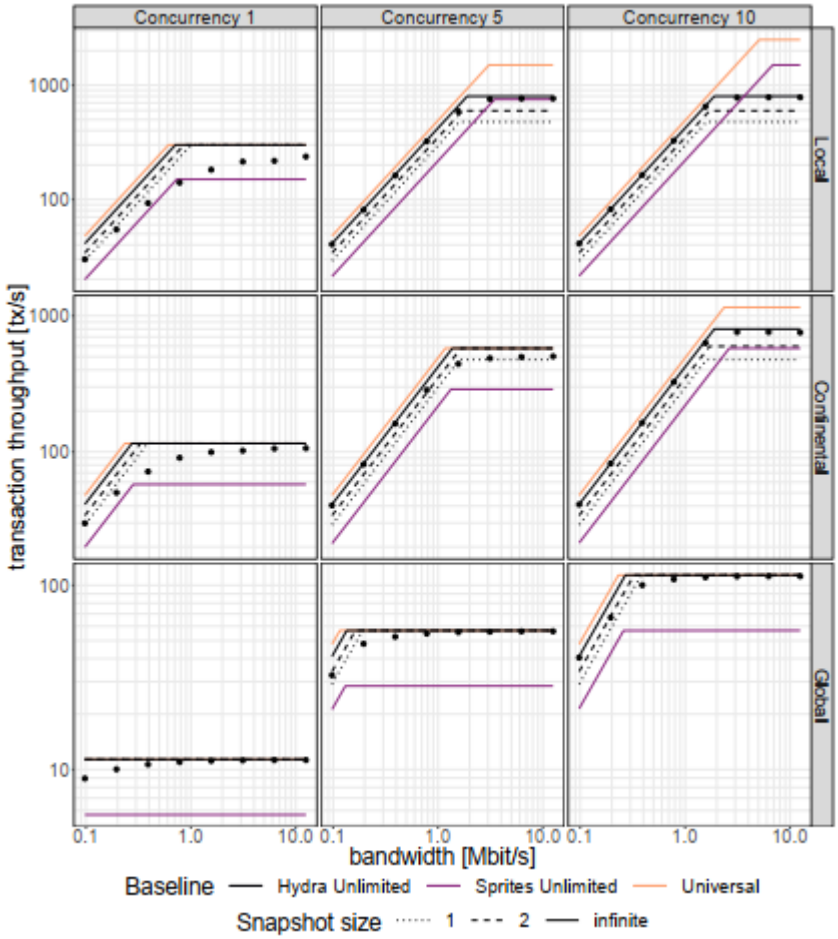
Chúng tôi đã thực hiện thử nghiệm cho ba cụm với sự phân bố địa lý khác nhau của các Node: triển khai *cục bộ* ba Node trong cùng một khu vực AWS, triển khai *lục địa* trên nhiều vùng AWS trên cùng một lục địa (Ireland, London và Frankfurt) và triển khai *toàn cầu* (Oregon, Frankfurt và Tokyo). Đối với mỗi cụm đó, chúng tôi đo lường sự phụ thuộc của thời gian xác nhận và thông lượng giao dịch vào băng thông và giao dịch đồng thời, và so sánh với các đường cơ sở được mô tả ở trên. Kết quả số phụ thuộc vào một số tham số mà chúng tôi đặt, thể hiện thời gian mà các hoạt động cơ bản trong giao thức thực hiện. Chúng tôi sử dụng các cài đặt được mô tả bên dưới.

Quy mô giao dịch. Chúng tôi sử dụng hai loại giao dịch đại diện: (1) giao dịch UTxO *đơn giản* với 2 đầu vào và 2 đầu ra, có kích thước là 265 Byte và (2) giao dịch tập lệnh chứa tập lệnh lớn hơn 10 Kbyte. Chúng tôi sử dụng các tham chiếu giao dịch có kích thước 32 Byte. Đối với mỗi thông điệp, chúng tôi cho phép chi phí cấp giao thức là 2 Byte.

Thời gian xác nhận giao dịch. Đây là thời gian CPU mà một Node sẽ sử dụng để kiểm tra tính hợp lệ của một giao dịch. Chúng tôi sử dụng các giá trị cố định ở đây: 0,4 ms cho các giao dịch đơn giản và 3 ms cho các giao dịch tập lệnh.

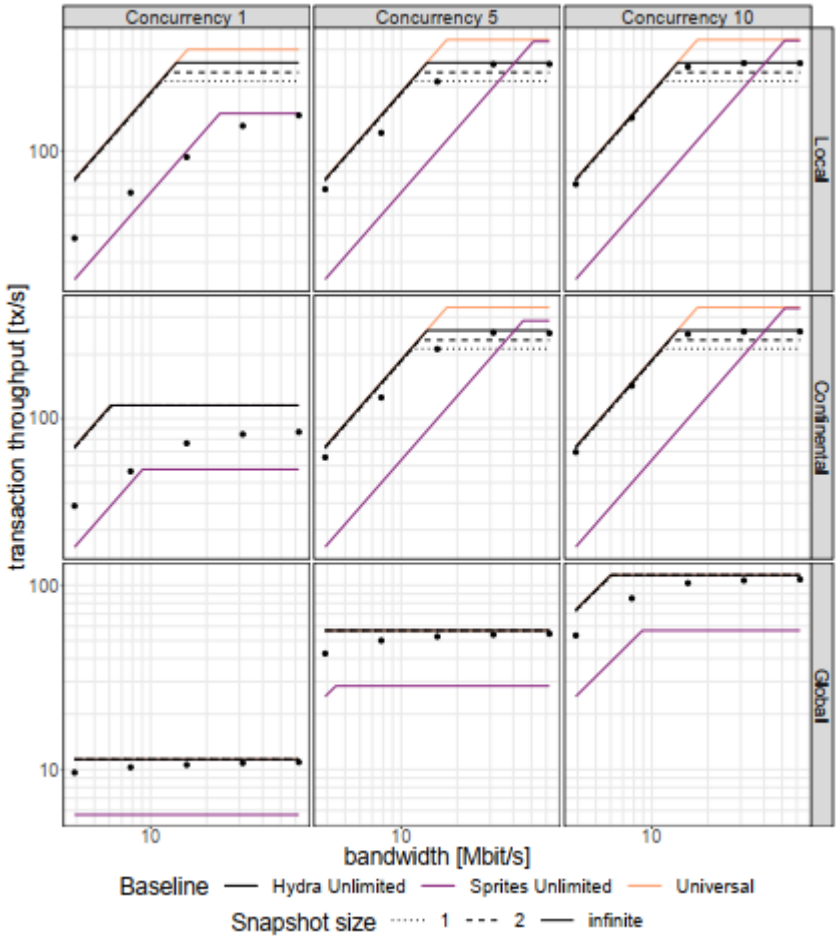
Thời gian cho các hoạt động đa chữ ký. Chúng tôi đã thực hiện các điểm chuẩn cho lược đồ đa chữ ký [11] dẫn đến các ước tính sau: 0,15 ms cho **MS-Sign**, 0,01 ms cho **MS-ASig** và 0,85 ms đối với **MS-AVerify**.

Thông lượng giao dịch. Hình 15 và 16 hiển thị kết quả tương ứng cho các giao dịch UTxO và tập lệnh thông thường. Các hàng khác nhau tương ứng với các thiết lập địa lý khác nhau của các cụm, trong khi các cột khác nhau về giao dịch đồng thời.



Hình 15: Tỷ lệ giao dịch cho giao thức Hydra Head, so với các kịch bản cơ sở. Giao dịch UTxO đơn giản với 2 đầu vào và 2 đầu ra.

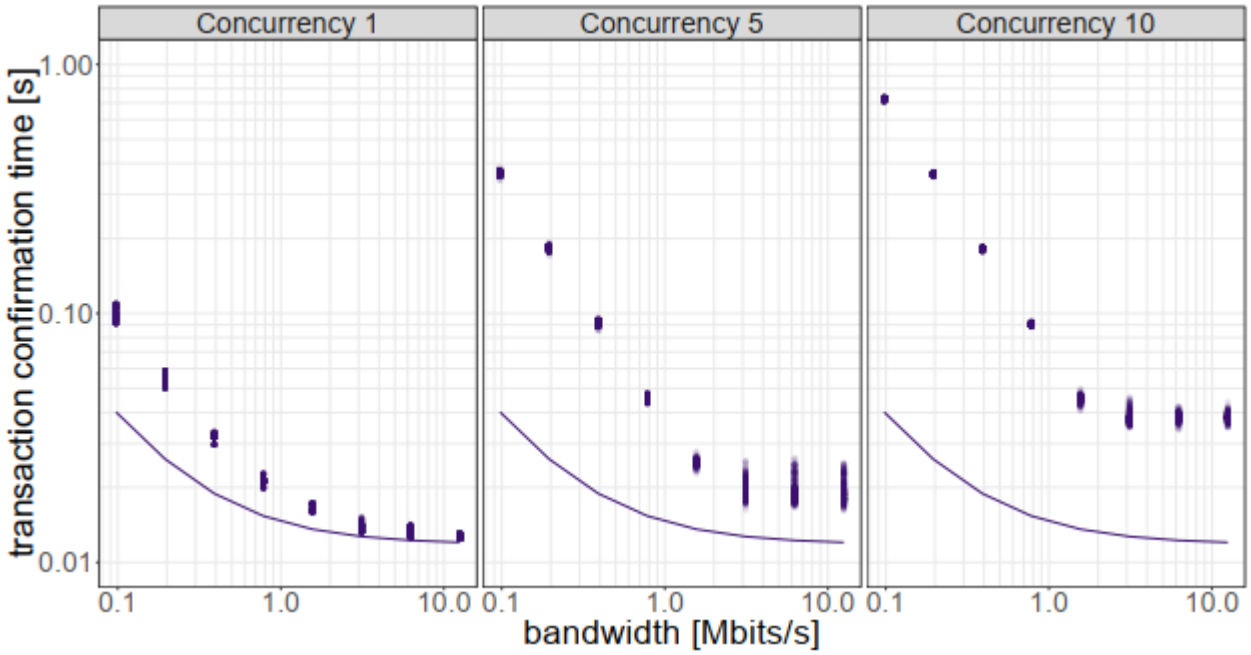
Đúng như dự đoán, Đường cơ sở chung luôn đưa ra tỷ lệ giao dịch cao nhất. Đối với Hydra không giới hạn, chúng tôi thấy 3 đường cơ sở cho các kích thước ảnh chụp nhanh khác nhau (được mô tả bằng các đường chấm, đứt nét và liền nét). Trong một số trường hợp, chúng trùng hợp. Đó là những thay đổi cấu hình mà chúng tôi bị giới hạn bởi độ trễ mạng: thực hiện ảnh chụp nhanh làm tăng nhu cầu về thời gian và băng thông CPU (đối với các chữ ký và thông điệp bổ sung), nhưng nó không làm tăng số lượng các vòng lặp mạng tuần tự phải được thực hiện để xác nhận các giao dịch (các thông điệp cho ảnh chụp nhanh và cho các giao dịch được truyền qua mạng đồng thời).



Hình 16: Tỷ lệ giao dịch cho giao thức Hydra Head, so với các kịch bản cơ sở.
Giao dịch tập lệnh.

So sánh các Đường cơ sở chung và Hydra không giới hạn, chúng tôi thấy rằng chúng giống hệt nhau bất cứ khi nào tỷ lệ giao dịch bị giới hạn bởi độ trễ của mạng. Điều đó có thể được giải thích vì sự khác biệt giữa hai đường cơ sở chỉ khác nhau về nhu cầu của chúng về thời gian CPU (để tạo và xác thực chữ ký) và băng thông (để gửi chữ ký). Lưu ý rằng đối với các giao dịch tập lệnh (Hình 16), nhu cầu về CPU vẫn cao hơn, do đó chi phí bổ sung cho các cấu trúc đa chữ ký thường có tác động thấp hơn nhiều đến tốc độ giao dịch.
Nhìn vào đường cơ sở của Sprites không giới hạn, chúng tôi quan sát thấy hiệu quả của việc phân lô thông qua một lãnh đạo trung tâm: lãnh đạo cần gửi tất cả các giao dịch đến

mọi Node khác. Do đó, giao diện mạng của nó thường là một nút cổ chai. Ngoài ra, chúng tôi thấy vòng tròn bổ sung giữa lãnh đạo và mọi Node khác làm giảm TPS bất cứ khi nào độ trễ mạng là tài nguyên giới hạn. Nhưng khi chúng ta có đủ đồng thời để tạo thành các lô lớn và đến khu vực mà chúng ta bị giới hạn bởi thời gian CPU, thì việc tiết kiệm bằng cách ký các lô thay vì các giao dịch riêng lẻ trở nên rõ ràng, và Đường cơ sở của Sprite gần đạt đến đường cơ sở chung.



Hình 17: Thời gian xác nhận cho các giao dịch UTxO đơn giản, trong một cụm nằm trong một vùng AWS. Từ bảng này sang bảng khác, chúng tôi tăng tính đồng thời của giao dịch. Thời gian xác nhận tối thiểu về mặt lý thuyết được thể hiện bằng một đường đứt nét.

So sánh kết quả thực nghiệm với đường cơ sở Hydra không giới hạn, chúng ta thấy rằng trong hầu hết các trường hợp, việc mô phỏng giao thức xấp xỉ đường cong tối ưu khá tốt. Chúng ta chỉ nhận được sự khác biệt đáng kể cho đồng thời thấp và không đủ băng thông.

Về ảnh chụp nhanh, các số liệu tiết lộ rằng việc thực hiện ảnh chụp nhanh có tác động không đáng kể đến tỷ lệ giao dịch: ngoài các khu vực có băng thông là giới hạn, các

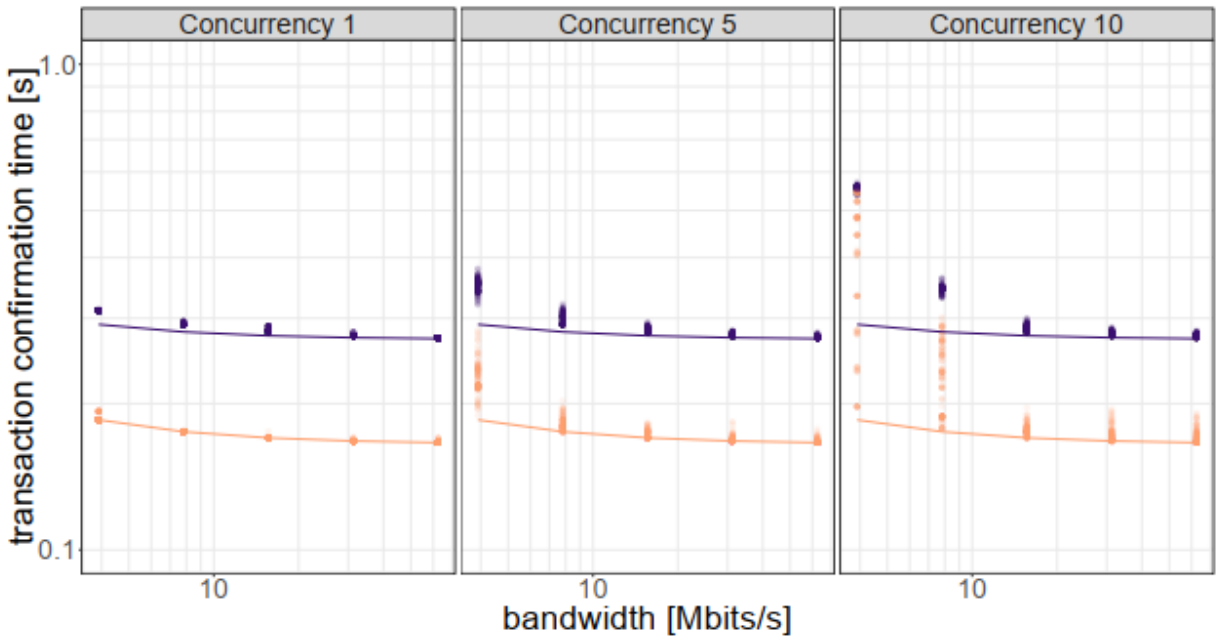
đường cơ sở cho các kích thước ảnh chụp nhanh khác nhau chỉ khác nhau khi chúng ta bị ràng buộc bởi CPU. Điều này yêu cầu đủ giao dịch đồng thời để khấu hao độ trễ mạng. Nhưng đối với đồng thời lớn, chúng ta cũng nhận được ảnh chụp nhanh lớn, vì vậy chi phí từ ảnh chụp nhanh *cho mỗi giao dịch* là nhỏ.

Thời gian xác nhận giao dịch. Một khía cạnh mà Hydra thực sự tỏa sáng là giải quyết nhanh chóng: ngay sau khi tất cả các bên đã ký kết giao dịch và Node gửi đã tổng hợp đa chữ ký hợp lệ, đa chữ ký này cung cấp đảm bảo rằng giao dịch có thể được đưa vào sổ cái của hệ thống Layer 1. Chúng ta có thể có được thời gian xác nhận tối thiểu bằng cách cộng thời gian xác thực giao dịch hai lần (một lần tại Node phát hành, một lần tại mọi Node khác), gửi thông điệp **reqTx** và **ackTx** qua đường dẫn dài nhất trong mạng, đồng thời tạo và xác thực chữ ký tổng hợp.

Hình 17 minh họa các điều kiện mà chúng tôi đạt được thời gian xác nhận tối thiểu. Trong bảng điều khiển đầu tiên, chúng tôi có một giao dịch đồng thời của nó. Chúng tôi thấy rằng, với đủ băng thông, chúng tôi tiến rất gần đến thời gian xác thực tối thiểu, được chỉ ra bằng đường. Trong các bảng khác, chúng tôi tăng tính đồng thời. Trong khi điều này làm tăng tổng thông lượng giao dịch bằng cách gửi các giao dịch song song, các giao dịch riêng lẻ có nhiều khả năng bị chậm lại do tắc nghẽn trong các giao diện mạng. Do đó, thời gian xác nhận và sự lây lan của nó tăng lên.

Trong các cụm trên các khu vực khác nhau, thời gian xác nhận thường phụ thuộc vào Node nào đã gửi giao dịch. Ví dụ, trong Hình 18, chúng ta thấy rằng các giao dịch từ Oregon có xu hướng được xác nhận nhanh hơn so với các giao dịch từ Frankfurt hoặc Tokyo. Điều này là do việc xác nhận yêu cầu một vòng đến Node ngang hàng xa nhất, và Frankfurt và Tokyo cách xa nhau hơn một trong hai đến từ Oregon.

Chúng tôi thấy rằng ngay cả đối với các giao dịch tập lệnh và mạng phân tán toàn cầu, chúng tôi luôn đạt được khả năng thanh toán tốt dưới nửa giây nếu chúng tôi cung cấp đủ băng thông.



Hình 18: Như Hình 17, nhưng đối với các giao dịch tập lệnh trong một cụm trải dài các khu vực AWS Oregon, Frankfurt và Tokyo. Ở đây, thời gian xác nhận tối thiểu phụ thuộc vào Node nào đang gửi giao dịch, vì vậy chúng ta có hai dòng tối ưu.

Các cụm lớn hơn. Ngoài 3 cụm Node, chúng tôi cũng đã đánh giá kết quả phụ thuộc như thế nào vào kích thước cụm bằng cách chạy mô phỏng với các cụm lên đến 100 Node (nằm trong cùng một vùng AWS):

- Tỷ lệ giao dịch của một cụm lớn hơn gần bằng tỷ lệ của một cụm 3 Node. Điều này là do thực tế là số lượng tính toán trên mỗi Node trên mỗi giao dịch không phụ thuộc vào số lượng người tham gia[8].

[8]Lưu ý rằng việc tổng hợp chữ ký và xác minh chữ ký tổng hợp phụ thuộc vào số lượng người tham gia. Tuy nhiên, điều này không ảnh hưởng đến tỷ lệ giao dịch trong mô phỏng của chúng tôi, vì 3 lý do: i) chúng tôi giả định rằng chúng tôi tổng hợp các khóa xác minh một lần vào đầu giao thức Head và chỉ thực hiện xác minh đối với khóa xác minh tổng hợp đã được tính toán trong giao thức, ii) ngay cả đối với 100 người tham gia, việc kết hợp các chữ ký sẽ nhanh hơn so với việc tạo ra một chữ ký duy nhất, iii) việc kết hợp các chữ ký được thực hiện đồng thời với phần còn lại của giao thức (xem Phần 7.2).

- Băng thông cần thiết tại mỗi Node để đạt được tốc độ giao dịch tối đa phụ thuộc vào kích thước cụm. Điều này không có gì đáng ngạc nhiên, vì mỗi Node cần giao tiếp với nhiều Node ngang hàng hơn.
- Vì lý do tương tự, thời gian xác nhận giao dịch tăng lên theo quy mô cụm.

Hãy lưu ý rằng các mô phỏng này vẫn sử dụng kiểu giao tiếp mà mọi người đều gửi thông điệp cho nhau, điều này không tối ưu cho các cụm lớn. Thay vào đó, chúng ta nên xây dựng một biểu đồ để phát các thông điệp, giữ cho số lượng người tham gia giao tiếp trực tiếp là nhỏ đối với mỗi người tham gia. Một lợi thế của cách tiếp cận Hydra là chúng ta có thể dễ dàng có các phiên bản khác nhau của giao thức Head, hoặc các triển khai khác nhau của cùng một giao thức Head, được tối ưu hóa cho các kích thước cụm khác nhau.

7.4. Thảo luận

Do cách thức đạt được sự đồng thuận bằng cách nhận được xác nhận từ mọi người tham gia, chúng tôi luôn đạt được giải quyết theo giây, ngay cả đối với những Head được phân phối trên toàn cầu. Khi chúng tôi phân bổ đủ tài nguyên mạng và chọn mức đồng thời thấp, chúng ta sẽ có được thời gian xác nhận tối ưu.

Về thông lượng giao dịch, quan trọng hơn những con số thô là sự so sánh với các giới hạn lý thuyết từ các kịch bản đường cơ sở:

- Chúng tôi thấy rằng chúng tôi không phải trả một chi phí đáng kể cho việc tạo ảnh chụp nhanh, cả về thông lượng giao dịch, cũng như về thời gian xác nhận. Đây là một điểm quan trọng: so với các giao thức kênh trạng thái khác, Hydra sử dụng song song UTxO để tránh phải thực hiện tuần tự các giao dịch. Ảnh chụp nhanh là cần thiết cho cách tiếp cận đó, vì nếu không, các giao dịch ngừng cam kết sẽ trở nên khó sử dụng. Chúng tôi thấy rằng ảnh chụp nhanh không làm chậm giao thức theo bất kỳ cách nào đáng kể, do đó xác nhận thiết kế của Hydra.
- So sánh Đường cơ sở chung, Hydra không giới hạn và các kết quả thực nghiệm, chúng tôi đã đạt đến giới hạn lý thuyết ở các khu vực mà chúng tôi có thể mong đợi. Khi chi phí đạt được sự đồng thuận thông qua đa chữ ký bị chi phối bởi thời gian vòng quanh mạng và xác thực giao dịch, chúng tôi tiến gần đến kịch bản chung. Chúng tôi chỉ thấy những sai lệch đáng kể so với Hydra không giới hạn khi có băng thông và giao dịch đồng thời thấp.

Bên cạnh việc chứng minh khả năng hoạt động hiệu quả của Hydra, các mô phỏng cũng cho phép các nhà khai thác xử lý băng thông mạng mà họ cần cung cấp để không ảnh hưởng đến hiệu suất của Head. Chúng cũng cho thấy rằng có sự cân bằng giữa tổng thông lượng giao dịch và thời gian giải quyết giao dịch riêng lẻ khi tăng tính đồng thời.

7.5. Lưu ý về thông lượng giao dịch

Chúng ta có thể thấy rằng tốc độ thông lượng giao dịch tối đa đạt được trong các thử nghiệm (đối với các giao dịch đơn giản) là khoảng 800 giao dịch mỗi giây. Giới hạn này là kết quả của thời gian xác thực giao dịch giả định là 0,4 ms và xác minh đa chữ ký mà chúng tôi cho phép 0, 85 ms. Do đó, mỗi giao dịch yêu cầu 1,25 ms thời gian CPU tại mỗi Node[9], vì vậy chúng tôi bị giới hạn ở 800 giao dịch mỗi giây.
Có nhiều cách đơn giản để tăng thông lượng trong một hệ thống trực tiếp:

- Cách hiệu quả nhất để mở rộng quy mô hệ thống với Hydra là chạy nhiều Head song song. Bằng cách chạy *n* Head, chúng tôi đạt được thông lượng gấp *n* lần của một Head đơn lẻ.
  Lưu ý rằng đối với nhiều trường hợp sử dụng, một Head đơn lẻ sẽ chỉ được sử dụng bởi những người tham gia trong khu vực địa lý hạn chế, cho phép thiết lập *cục bộ* hoặc *lục địa* hiệu quả.
- Để tăng thông lượng của một Head đơn lẻ, người ta có thể đầu tư vào phần cứng có khả năng hơn để tăng tốc độ xác thực giao dịch và xác minh chữ ký.
- Trong các thử nghiệm, mọi Node đều xử lý tuần tự các giao dịch. Nhưng có thể thực hiện song song các phần lớn của xác thực giao dịch và tất cả xác minh chữ ký cho nhiều giao dịch, sử dụng nhiều phần lõi trên mỗi Node. Sự tối ưu hóa này cũng có thể cải thiện thông lượng của một Head đơn lẻ.

8. Lời cảm ơn

Aggelos Kiayias được hỗ trợ một phần bởi Dự án số 780477 của EU, PRIVILEDGE. Chúng tôi muốn cảm ơn Duncan Coutts và Neil Davies đã tư vấn về các khía cạnh kỹ thuật của mô phỏng và Neil Davies đã cung cấp các phép đo thời gian khứ hồi giữa các vùng AWS khác nhau.

[9]Lưu ý rằng như được mô tả trong Phần 7.2, chúng tôi tạo đa chữ ký trong một chuỗi chuyên dụng.

Tài liệu tham khảo

[1] Mô hình eUTXO-2. https://github.com/hydra-supplementary-material/eutxo-spec/blob/master/expand-utxo-specification.pdf.

[2] Thư viện io-sym. https://github.com/input-output-hk/ouroboros-network/tree/master/io-sim, https://github.com/input-output-hk/ouroboros-network/tree/master/io-sim-classes.

[3] Mạng lưới kết nối. https://docs.connext.network/en/latest/background/architecture.html.

[4] John Adler. Lý do của Rollup lạc quan. https://medium.com/@adlerjohn/the-why-s-of-precision-rollup-7c6a22cbb61a, tháng 11 năm 2019.

[5] Ian Allison. Vitalik Buterin của Ethereum giải thích cách các kênh trạng thái giải quyết quyền riêng tư và khả năng mở rộng. *Thời báo Kinh doanh Quốc tế*, 2017.

[6] Nicola Atzei, Massimo Bartoletti, Stefano Lande và Roberto Zunino. Một mô hình chính thức của các giao dịch Bitcoin. Trong *Mật mã tài chính và Bảo mật dữ liệu - Hội nghị quốc tế lần thứ 22, FC 2018, Nieuwpoort, Cura¸cao, ngày 26 tháng 2 - ngày 2 tháng 3 năm 2018, Các bài báo đã được sửa đổi*, trang 541–560, 2018.

[7] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón và Pieter Wuille. Kích hoạt các đổi mới Blockchain với các Sidechain được chốt, 2014.

[8] Mihir Bellare và Gregory Neven. Đa chữ ký trong mô hình khóa công khai đơn giản và bổ đề chia đôi tổng quát. Trong Ari Juels, Rebecca N. Wright, và Sabrina De Capitani di Vimercati, người biên tập, *ACM CCS 2006* , trang 390–399. ACM Press, tháng 10 / tháng 11 năm 2006.

[9] Wiki Bitcoin. Các kênh thanh toán. Bài viết trên Wiki, truy cập ngày 11 tháng 11 năm 2019.

[10] Guy E. Blelloch. Lập trình các thuật toán song song. *Thông tin liên lạc của ACM*, 39: 85–97, 1996.

[11] Alexandra Boldyreva. Chữ ký ngưỡng, đa chữ ký và chữ ký mù dựa trên lược đồ chữ ký nhóm gap-Diffie-Hellman. In Yvo Desmedt, chủ biên, *PKC 2003*, tập 2567 của *LNCS*, trang 31–46. Springer, Heidelberg, tháng 1 năm 2003.

[12] Dan Boneh, Manu Drijvers và Gregory Neven. Đa chữ ký nhỏ gọn cho các Blockchain nhỏ hơn. Trong Thomas Peyrin và Steven Galbraith, người biên tập, *ASIACRYPT 2018, Phần II*, tập 11273 của *LNCS*, trang 435–464. Springer, Heidelberg, tháng 12 năm 2018.

[13] Manuel MT Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Michael Peyton Jones và Philip Wadler. Mô hình UTxO mở rộng. Trong *Hội thảo lần thứ 4 về Hợp đồng thông minh đáng tin cậy*, năm 2020. http://fc20.ifca.ai/wtsc/WTSC2020/WTSC20_paper_25.pdf.

[14] Manuel MT Chakravarty, Roman Kireev, Kenneth MacKenzie, Vanessa McHale, Jann Mu¨ller, Alexander Nemish, Chad Nester, Michael Peyton Jones, Simon Thompson, Rebecca Valentine và Philip Wadler. Các hợp đồng Blockchain chức năng. https://iohk.io/ en / research / library / paper / functions-Blockchain-Contract/, tháng 5 năm 2019.

[15] Jeff Coleman, Liam Horne và Li Xuanji. Ngược lại: Các kênh trạng thái tổng quát, 2018.

[16] Christian Decker và Roger Wattenhofer. Mạng thanh toán nhanh và có thể mở rộng với các kênh thanh toán vi mô song công Bitcoin. Trong *Hội thảo chuyên đề về Hệ thống tự ổn định*, trang 3–18. Springer, 2015.

[17] Nhà phát triển Ergo. Ergo: Một nền tảng linh hoạt để kiếm tiền theo hợp đồng. https://ergoplatform.org/docs/whitepaper.pdf, tháng 5 năm 2019.

[18] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, Julia Hesse và Kristina Hostáková. Kênh trạng thái ảo đa bên. Trong *Hội nghị Quốc tế Thường niên về Lý thuyết và Ứng dụng của Kỹ thuật Mật mã*, trang 625–656. Springer, 2019.

[19] Stefan Dziembowski, Lisa Eckey, Sebastian Faust và Daniel Malinowski. Perun: Trung tâm thanh toán ảo trên Crypto. Trong *Hội nghị chuyên đề IEEE về Bảo mật và Quyền riêng tư (SP) năm 2019*, các trang 106–123. IEEE, 2019.

[20] Stefan Dziembowski, Grzegorz Fabianski, Sebastian Faust và Siavash Riahi. Giới hạn thấp hơn cho các giao thức ngoài chuỗi: Khám phá các giới hạn của Plasma. Cryptology ePrint Archive, Report 2020/175, 2020. https://eprint.iacr.org/2020/175.

[21] Stefan Dziembowski, Sebastian Faust và Kristina Hostáková. Các mạng lưới kênh trạng thái chung. Trong *Kỷ yếu của Hội nghị ACM SIGSAC 2018 về Bảo mật Máy tính và Truyền thông*, trang 949–966. ACM, 2018.

[22] Ethereum. Cây Patricia, 2019. Kho lưu trữ Github.

[23] Matthias Fitzi, Daniel Gottesman, Martin Hirt, Thomas Holenstein và Adam Smith. Thỏa thuận Byzantine có thể phát hiện an toàn chống lại phần lớn bị lỗi. Trong Aleta Ricciardi, chủ biên, *ACM PODC thứ 21*, trang 118–126. ACM, tháng 7 năm 2002.

[24] P. Gazi, A. Kiayias và D. Zindros. Các Sidechain bằng chứng cổ phần. Năm *2019 Hội nghị chuyên đề IEEE về Bảo mật và Quyền riêng tư (SP)*, trang 677–694, Los Alamitos, CA, Hoa Kỳ, tháng 5 năm 2019. Hiệp hội Máy tính IEEE.

[25] Kazuharu Itakura và Katsuhiro Nakamura. Một hệ thống mật mã khóa công khai thích hợp cho các ký tự đa dạng kỹ thuật số. *Nghiên cứu & Phát triển NEC*, (71): 1–8, 1983.

[26] Aggelos Kiayias và Dionlysis Zindros. Các Sidechain bằng chứng công việc. *IACR Cryptology ePrint Archive*, 2018: 1048, 2018.

[27] Georgios Konstantopoulos. Tiền mặt Plasma: Hướng tới các công trình Plasma hiệu quả hơn, năm 2019.

[28] Jeremy Longley và Oliver Hopton. Thảo luận và lộ trình công nghệ Funfair, 2017.

[29] ScaleSphere Foundation Ltd. Mạng Celer: Mang quy mô Internet đến mọi Blockchain, năm 2018.

[30] Silvio Micali, Kazuo Ohta và Leonid Reyzin. Đa chữ ký nhóm phụ có thể giải trình: Bản tóm tắt mở rộng. Trong Michael K. Reiter và Pierangela Samarati, người biên tập, *ACM CCS 2001*, trang 245–254. ACM Press, tháng 11 năm 2001.

[31] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan và Patrick McCorry. Sprites và kênh trạng thái: Mạng thanh toán nhanh hơn Lighting. Trong *Hội nghị Quốc tế về Mật mã Tài chính và Bảo mật Dữ liệu*, trang 508–526. Springer, 2019.

[32] J. Poon và V. Buterin. Plasma: Hợp đồng thông minh tự trị có thể mở rộng. http://plasma.io/plasma.pdf .

[33] Joseph Poon và Thaddeus Dryja. Mạng lưới Bitcoin Lightning: Thanh toán tức thì ngoài chuỗi có thể mở rộng, 2016.

[34] Joachim Zahnentferner. Một mô hình trừu tượng của Crypto dựa trên UTxO với các tập lệnh. *IACR Cryptology ePrint Archive*, 2018: 469, 2018.

A. Bảo mật của các lược đồ đa chữ ký

Để thuận tiện, định nghĩa của chúng tôi về lược đồ đa chữ ký được đưa ra trong Phần 2 khác với định nghĩa tiêu chuẩn trong ví dụ [8] bằng cách giả định sự tồn tại của các thuật toán riêng biệt **MS-Sign** và **MS-ASig** và giả định rằng cách tạo ra một đa chữ ký là mỗi bên tạo ra một chữ ký cục bộ thông qua **MS-Sign** và sau đó chúng được trao đổi và kết hợp bằng **MS-ASig**. Đây không phải là sai lệch quan trọng, vì các lược đồ đa chữ ký hiện đại điển hình (bao gồm cả lược đồ chúng tôi sử dụng trong mô phỏng của mình [11]) đáp ứng mô hình này. Dưới đây, chúng tôi trình bày một định nghĩa tiêu chuẩn về bảo mật đa chữ ký từ [8, 12], phù hợp với trường hợp đặc biệt này.
Một lược đồ đa chữ ký bảo mật cần phải thỏa mãn hai thuộc tính: tính hoàn chỉnh và tính không thể làm giả được (Unforgeability).

Tính hoàn chỉnh. Đối với bất kỳ n, $\Pi \leftarrow$ **MS-Setup**(1^k) và (**vk**$_i$, **sk**$_i$) $\leftarrow$ **MS-KG** (Π) cho $i = 1, ..., n$, cho bất kỳ thông điệp m nếu chúng ta có $\sigma_i \leftarrow$ **MS-Sign**$(\Pi,$ **sk**$_i$, $m)$, $\tilde{\sigma} \leftarrow$ **MS-ASig** $(\Pi, m, \{\mathsf{vk}_i\}_{i=1}^n, \{\sigma_i\}_{i=1}^n)$ và **avk** $\leftarrow$ **MS-AVK** $(\Pi, \{\mathsf{vk}_i\}_{i=1}^n)$, sau đó **MS-Verify** $(\Pi, \tilde{\sigma}, m,$ **avk**$)$ = True.

Tính không thể làm giả được. Thuộc tính này được xác định bởi một trò chơi ba giai đoạn:

- *Thiết lập*. Người thách thức chạy $\Pi \leftarrow$ **MS-Setup** (1^k), tạo cặp khóa thách thức (**vk***, **sk***) $\leftarrow$ **MS-KG**(Π) và chạy đối nghịch $A(\Pi,$ **vk***$)$.
- *Truy vấn ký tên. A* được phép thực hiện các truy vấn ký trên bất kỳ thông điệp m nào, tức là A có quyền truy cập vào chữ ký của Oracle **MS-Sign** $(\Pi,$ **sk***,$\cdot)$.
- *Đầu ra*. Cuối cùng, A đưa ra giả mạo đa chữ ký $\tilde{\sigma}^*$, một thông điệp m^* và một bộ khóa xác minh V^*, A thắng nếu **vk*** $\in V^*$, A không thực hiện truy vấn ký trên m^*, và

  MS-Verify$(\Pi,$ **MS-AVK**$(\Pi, V^*), m^*, \tilde{\sigma}^*)$ = True.

Chúng ta nói rằng A là (τ, q, ε)-Forger để cho lược đồ đa chữ ký **MS** = (**MS-Setup**, **MS-KG**, **MS-AVK**, **MS-Sign**, **MS-Asig**, **MS-Verify**) nếu nó chạy trong thời gian τ, thực hiện truy vấn ký q, và thắng trò chơi trên với xác suất ít nhất là ε. **MS** là (τ, q, ε)-Unforgeability nếu không tồn tại (τ, q, ε)-Forger.

B. Giao thức Head đơn giản với giải quyết xung đột

Phần này giải thích sự khác biệt giữa giao thức Head cơ bản từ Phần 6 và giao thức Head có giải quyết xung đột; nó cũng chứa một bằng chứng bảo mật tương ứng.

B.1. Mô tả giao thức

Giao thức Head *với giải quyết xung đột* (CR - Conflict Resolution) hoạt động giống như giao thức Head cơ bản, ngoại trừ việc ảnh chụp nhanh cũng được sử dụng để giải quyết xung đột giữa các giao dịch. Trong giao thức cơ bản, vì các bên không ký các giao dịch xung đột, nên những xung đột như vậy sẽ phải được giải quyết trên chuỗi chính ngay cả khi không có thành viên Head nào bị hỏng.

Trong phiên bản CR của giao thức Head (xem Hình 19), mỗi thành viên Head duy trì thêm một tập hợp $\hat{R}$ các giao dịch đã biết xung đột với một tập hợp $T_R \subseteq \hat{T}$ các giao dịch đã được ký bởi p_i. Trong trường hợp (có khả năng) mà các giao dịch bằng $\hat{R}$ cũng đã được ký bởi ít nhất một bên, thì không có giao dịch nào trong $T_R \cup \hat{R}$ có thể được xác nhận với quy trình xác nhận cơ bản dựa trên đa chữ ký trên các giao dịch. Để tránh điều này, bất cứ khi nào p_i là lãnh đạo ảnh chụp nhanh, p_i cũng bao gồm các bộ (Hash của) các giao dịch trong T_R và $R \leftarrow \hat{R}$ trong một thông điệp **reqSn** (ngoài ra đối với tập hợp $\bar{T}$ của các giao dịch đã được xác nhận không được bao gồm trong bất kỳ ảnh chụp nhanh nào cho đến nay).

Phần còn lại của quá trình ảnh chụp nhanh được thay đổi (so với giao thức cơ bản) để đảm bảo rằng các giao dịch trong T_R được xác nhận và các giao dịch trong R bị loại bỏ. Cụ thể, một bên p_i nhận được yêu cầu chụp nhanh **reqSn** trước tiên sẽ đợi cho đến khi anh ta biết được tất cả các giao dịch được tham chiếu bởi các bộ T_R và R nhận được từ lãnh đạo ảnh chụp nhanh p_j. (Quan sát thấy rằng các bên có thể có các tập cục bộ khác nhau $\hat{T}$ và $\hat{R}$). Sau đó, nó đảm bảo rằng các tập hợp này là hợp pháp khi các giao dịch trong T_R và R thực sự xung đột. Hơn nữa, nó kiểm tra rằng các giao dịch trong T_R không xung đột với các giao dịch đã được xác nhận. Nếu các lần kiểm tra này vượt qua, p_i cập nhật các bộ $\hat{T}$ và $\hat{R}$ của mình để khớp với các bộ của lãnh đạo ảnh chụp nhanh p_j. Hàm **snObj** — ngoài số ảnh chụp nhanh, bộ UTxO của ảnh chụp nhanh trước đó và tập hợp T các giao dịch được xác nhận tối đa

Giao thức Hydra Head đơn giản hoá với giải quyết xung đột

Hình 19: Máy giao thức Head với giải quyết xung đột theo quan điểm của bên p_i.

giờ đây cũng lấy đầu vào là bộ T_R và tính toán bộ UTxO cho ảnh chụp nhanh mới $\widehat{U}$ như[10]

$$\widehat{U}.U \leftarrow \mathrm{U.U} \circ (\mathbf{Reach}^{\bar{T}}(\mathrm{T}) \cup T_R).$$

Bên p_i cuối cùng ký $H(\widehat{U}.U) \| \hat{s}$ và gửi chữ ký cho lãnh đạo ảnh chụp nhanh p_j.

Phần còn lại của quy trình ảnh chụp nhanh rất giống với quy trình trong giao thức cơ bản, ngoại trừ trong **confSn**, (**conf**, tx) là đầu ra cho các giao dịch được tham chiếu bởi T_R; các giao dịch này cũng bị loại bỏ dưới dạng tập hợp $\widehat{T}$ và việc thực thi **reqTx** bị dừng và bị loại bỏ đối với tất cả các giao dịch xung đột với T_R (các thực thi như vậy sẽ bị kẹt trong lệnh **wait** mãi mãi).

Quan sát thấy rằng nếu ít nhất một bên bị hỏng, phiên bản CR của giao thức Head hiện cho phép kẻ tấn công tạo các giao dịch đa chữ ký xung đột với các giao dịch được xác nhận thông qua quy trình ảnh chụp nhanh. Điều này xảy ra nếu một bên trung thực p_i ký một ảnh chụp nhanh bao gồm trong tập T_R (được gửi bởi lãnh đạo ảnh chụp nhanh) một giao dịch tx' trong tập hợp $\hat{R}$ của p_i : tx' đang ở trong $\hat{R}$ có nghĩa là p_i đã ký tắt trên một giao dịch tx xung đột với tx'. Điều này dẫn đến điều kiện chạy đua giữa hai sự kiện sau:

- p_i nhận được đa chữ ký của tx qua **confTx**;
- p_i ký tắt trên ảnh chụp nhanh bao gồm tx'.

Điều quan trọng, chỉ một trong những sự kiện này phải xảy ra. Cuối cùng, trong **reqSn**, một bên kiểm tra xem tập hợp T_R không xung đột với các giao dịch đã được xác nhận (trong câu lệnh **require** thứ tư) và (**conf**, tx) chỉ được xuất trước khi p_i ký vào ảnh chụp nhanh mới (kiểm tra $\widehat{U} \circ \mathrm{tx} \neq \perp$), tức là nếu đa chữ ký cho tx đến sau khi ảnh chụp nhanh được ký, p_i sẽ chỉ cần bỏ nó đi. Lưu ý rằng, trong trường hợp lệnh **require** trên không thành công đối với một bên trung thực, quá trình sản xuất ảnh chụp nhanh sẽ bị đình trệ vì ảnh chụp nhanh này sẽ không bao giờ được xác nhận. Tuy nhiên, trong trường hợp này, Head có thể được đóng lại một cách an toàn vì lãnh đạo ảnh chụp nhanh bị hỏng.

B.2. Bằng chứng bảo mật

Xem xét các biến ngẫu nhiên được xác định ở đầu Phần 6.4. Bằng chứng tiến hành theo các dòng tương tự như của giao thức cơ bản với việc xem xét thêm thực tế là

[10]Hãy nhớ lại rằng $\mathbf{Reach}^{\bar{T}}(\mathrm{T})$ trả về các giao dịch trong $\bar{T}$ có thể truy cập được (bằng cách theo dõi các tham chiếu đầu ra) từ các giao dịch (với Hash) trong T.

các bên (trung thực) khác nhau có thể ký các giao dịch xung đột với nhau do điều kiện cuộc đua.

Bổ đề 5 (Tính nhất quán). *Giao thức head với CR thỏa mãn thuộc tính nhất quán.*
Chứng minh. Hãy xem xét một bên p_i không bị gián đoạn tùy ý và một giao dịch tx mà p_i xuất ra (**conf**, tx). Giả sử tồn tại một bên p_j sao cho tx xung đột với tx' $\in \bar{C}_j$. Hãy xem xét các trường hợp sau:

- Cả tx và tx' đều được xác nhận thông qua **confTx**. Điều này không thể xảy ra vì nó có nghĩa là các bên trung thực đã ký thông điệp **reqTx** cho các giao dịch xung đột.
- Giao dịch tx được xác nhận qua **confTx** và tx' qua **confSn**. Điều này không thể xảy ra vì nó có nghĩa là p_i đã ký một ảnh chụp nhanh xung đột với tx' trước khi xuất ra (**conf**, tx).
- Cả tx và tx' đều được xác nhận thông qua **confSn**. Điều này không thể xảy ra vì các giao dịch được xác nhận thông qua ảnh chụp nhanh được một bên kiểm tra xung đột trước khi bên đó ký vào ảnh chụp nhanh.

Bất biến 9 (Phát hiện lỗi của lãnh đạo ảnh chụp nhanh). *Nếu đối với bất kỳ bên trung thực nào, trong* (**confTx**)*, bên đó cho rằng* $\bar{L} \circ \text{tx} = \perp$ *hoặc* $\widehat{U} \circ \text{tx} = \perp$ *thì có một bên bị lỗi.*
Chứng minh. Nếu $\bar{L} \circ \text{tx} = \perp$ thì chắc chắn phải có một lãnh đạo ảnh chụp nhanh đã ký tx nhưng giải quyết xung đột của tx có lợi cho một giao dịch khác — chống lại tx trái với quy tắc để thích giao dịch đã ký hơn. Hành vi này bị lỗi. Trường hợp $\widehat{U} \circ \text{tx} = \perp$ thể hiện hành vi bị lỗi của lãnh đạo ảnh chụp nhanh hiện tại theo cách tương tự.

Bất biến 10 (Bao gồm $\bar{C}$ lẫn nhau cuối cùng). *Xem xét sự hiện diện của một kẻ tấn công mạng. Khi đó, cho trước* C_i *của bên* p_i *tại bất kỳ thời điểm nào, bất kỳ bên* p_j *nào cuối cùng sẽ có* $C_j \supseteq C_i$.
Chứng minh. Bất kỳ giao dịch nào được thêm vào C_i trong (**confSn**) cuối cùng sẽ được (hoặc đã được) thêm vào C_j vì người bảo vệ duy nhất trong (**confSn**) liên quan đến xác minh chữ ký.

Bất kỳ giao dịch tx nào được thêm vào C_i trong (**confTx**) cuối cùng sẽ kích hoạt (hoặc đã kích hoạt) một (**confTx**) tương ứng cho bên p_j và bằng cách giả định rằng tất cả các bên đều trung thực và với Bất biến 9, tx $\in C_j$ cuối cùng giữ nguyên.

Trong bổ đề sau chúng tôi thiết lập rằng, dưới sự kiểm soát của kẻ tấn công mạng lưới, các ảnh chụp nhanh mới tiếp tục được tạo và xác nhận. Thuộc tính này không chỉ được yêu cầu cho bằng chứng về tính hợp lý mà còn chứng minh rằng các giao dịch cũ cuối cùng có thể bị xóa.

Bổ đề 6 (Tính sống động của ảnh chụp nhanh). *Dưới sự hiện diện của kẻ tấn công mạng lưới, đối với bất kỳ* $s > 0$*, một ảnh chụp nhanh với số ảnh chụp nhanh cuối cùng sẽ được xác nhận.*

Chứng minh. Chúng ta phải chứng minh rằng đối với bất kỳ số ảnh chụp nhanh s và bên p_i nào, các điều kiện sau đây cuối cùng được giữ nguyên khi p_i vào phiên bản xử lý sự kiện tương ứng (reqSn, s, T, T_R, R):

1. $T \subseteq \bar{T}^{\downarrow(h)} \wedge T_R \cup R \subseteq (\hat{T} \cup \hat{R})^{\downarrow(h)}$
2. $\forall tx \in \tilde{T}_R$: **conflict**$(\tilde{R} \cup \{tx\}) \wedge \forall tx \in \tilde{R}$: **conflict**$(\tilde{T}_R \cup \{tx\})$
3. ¬**conflict**$(\bar{T} \cup \tilde{T}_R)$

Đối với (1), phần $T \subseteq \bar{T}^{\downarrow(h)}$. Khi lãnh đạo ảnh chụp nhanh chọn $T \subseteq \bar{T}^{\downarrow(h)}$ trong (**newSn**) (và $\bar{T} \subseteq \bar{C}$), với Bất biến 10, mọi bên p_i cuối cùng sẽ nhận thấy $T \subseteq \bar{T}_i^{\downarrow(h)} \subseteq \bar{C}_i$.

Đối với (1), phần $T_R \cup R \subseteq (\hat{T} \cup \hat{R})^{\downarrow(h)}$. Sau khi nhập (**newSn**), tất cả các giao dịch trong $tx \in T_R \cup R$ cuối cùng sẽ được kích hoạt (**reqTx**, tx) thể hiện rằng $T_R \cup R \subseteq (\hat{T} \cup \hat{R})^{\downarrow(h)}$.

Đối với (2). Như phần trên, chúng ta có $T_R \cup R \subseteq (\hat{T} \cup \hat{R})^{\downarrow(h)}$. Khi điều kiện đó được thỏa mãn thì cũng với $\forall tx \in \tilde{T}_R$: **conflict**$(\tilde{R} \cup \{tx\}) \wedge \forall tx \in \tilde{R}$: **conflict**$(\tilde{T}_R \cup \{tx\})$ bởi sự trung thực của lãnh đạo ảnh chụp nhanh và cách anh ta phải chọn T_R và R trong (**newSn**).

Đối với (3). Nếu **conflict**$(\bar{T} \cup \tilde{T}_R)$ thì lãnh đạo ảnh chụp nhanh giải quyết xung đột có lợi cho giao dịch $tx \in \tilde{T}_R$ sao cho **conflict**$(\bar{T} \cup \{tx\})$ thể hiện rằng anh ta cũng đã ký một giao dịch xung đột với tx, mâu thuẫn với giả định rằng anh ta trung thực.

Bất biến 11 (Tính sống động của giao dịch cục bộ). *Dưới sự hiện diện của một kẻ tấn công mạng, hãy xem xét bất kỳ giao dịch* tx *nào do một bên* p_i *phát hành thông qua* (**new**)*. Sau đó, cuối cùng, hoặc là* $tx \in \bar{C}_i$*, hoặc* **conflict**$(tx, \bar{C}_i)$.

Chứng minh. Sau (**new**, tx), p_i cuối cùng sẽ xử lý tx bằng (**reqTx**, tx), và $tx \in \hat{T} \cup \hat{R}$ (chênh lệch đối xứng). Giả sử rằng không bao giờ $tx \in \bar{C}_i$, giả sử rằng tất cả các bên đều trung thực và theo Bất biến 9, thể hiện rằng $tx \in \hat{R}_j$ của ít nhất một bên p_j. Hãy xem xét ảnh chụp nhanh tiếp theo được tạo ra bởi p_j. Vì $tx \in \hat{R}_j$, có một giao dịch

tx' $\in \hat{T}_j$ với **conflict**({tx, tx'}) mà anh ta thêm vào T_R, và theo Bổ đề 6, cuối cùng tx' $\in \bar{C}_k$ với mọi bên p_k.

Bổ đề 7 (Tính sống động). *Giao thức Head đáp ứng tình sống động.*
Chứng minh. Theo tính sống động của giao dịch cục bộ và Bất biến 10.

Bất biến 12. *Gọi $\tilde{T}$ là tập tương ứng với* $SN_{cur,i}$. *Khi đó, $\tilde{T} \cup \bar{T}_i = \bar{C}_i$, trong đó $\bar{T}_i$ là (biến ngẫu nhiên tương ứng với) tập hợp $\bar{T}$ của* p_i.
Chứng minh. Cố định một số bên p_i. Quan sát thấy rằng bất biến được thỏa mãn một cách đáng kể khi bắt đầu thực thi của giao thức. Hơn nữa, mỗi khi một giao dịch mới được xác nhận qua **confTx**, cả $\bar{T}_i$ và $\bar{C}_i$ tăng lên nhờ giao dịch mới được xác nhận, trong khi $\tilde{T}$ không đổi.

Lần khác duy nhất một trong các bộ $\tilde{T}$, $\bar{T}_i$ và $\bar{C}_i$ thay đổi là khi một ảnh chụp nhanh mới được xác nhận qua **confSn**. Trong trường hợp như vậy, hãy lưu ý rằng:

- Bất kỳ giao dịch nào bị xóa khỏi $\bar{T}_i$ đều được xem xét bởi ảnh chụp nhanh mới và do đó được thêm vào $\tilde{T}$, và
- Bất kỳ giao dịch nào được thêm vào $\bar{C}_i$ cũng được xem xét bởi ảnh chụp nhanh mới và do đó được thêm vào $\tilde{T}$.

Do đó, bất biến vẫn thoả mãn.

Bất biến 13. $\tilde{T}_0 \subseteq \tilde{T}_1 \subseteq \tilde{T}_2 \subseteq$ ....
Chứng minh. Cho p_i là một bên trung thực. Dễ dàng nhận thấy rằng tập hợp các giao dịch được xem xét bởi ảnh chụp nhanh mới luôn bao gồm tập hợp được xem xét bởi ảnh chụp nhanh trước đó vì tập hợp các giao dịch T và T_R trong một **reqSn** thỏa mãn rằng $SN_{cur,i} \circ (\mathbf{Reach}^{\overline{T_i}}(T) \cup T_R) \neq \perp$, (điều này được thể hiện bởi Bất biến 12).

Bất biến 14. *Nếu* tx $\in \bar{C}_i \cap C_{chain}$, *nó sẽ vẫn ở đó.*
Chứng minh. Xét phép toán **Contest**(K_{agg}, η, ξ) và cho $\eta = (U_\eta, s_\eta, T_\eta)$ và $\xi = (U, s, \tilde{\sigma}, T)$. Lưu ý rằng $C_{chain} = \tilde{T}_{s\eta} \cup T_\eta$ giữ trước khi hoạt động. Bây giờ hãy xem xét tập T^* trong đầu ra $(\cdot, \cdot, T^*)$ của **Contest**. Lưu ý rằng sau hoạt động $C_{chain} = \tilde{T}_s \cup T^*$. Quan sát thấy rằng:

- Vì $s \geq s_\eta$, Bất biến 13 thể hiện rằng một giao dịch tx $\in \tilde{T}_{s\eta}$ cũng nằm trong $\tilde{T}_s$.
- Nếu một giao dịch tx $\in T_\eta$ không nằm trong T^* mà trong $\bar{C}_i$, thì $s > s_\eta$ và giao dịch được sử dụng bởi ảnh chụp nhanh với số s, tức là tx $\in \tilde{T}_s$. Điều này là do thực tế là các bên trung thực không ký các ảnh chụp nhanh mâu thuẫn với các giao dịch đã xác nhận.

Bất biến 15. *Với mọi* $i \in H_{\text{cont}}$, $\bar{C}_i \subseteq C_{\text{chain}}$.
Chứng minh. Lấy bất kỳ bên trung thực p_i và đặt $\tilde{s}$ là số ảnh chụp nhanh hiện tại ở p_i, tức là $SN_{cur,i} = U_0 \circ \tilde{T}_{\tilde{s}}$. Hãy nhớ rằng, theo Bất biến 12, $\bar{C}_i = \tilde{T}_{\tilde{s}} \cup \bar{T}_i$. Xem xét một hoạt động đóng hoặc tranh chấp theo p_i cũng như đầu ra (U, s, T^*) của **Contest** và quan sát thấy rằng $C_{\text{chain}} = \tilde{T}_s \cup T^*$ giữ sau hoạt động. Qua Bất biến 13, $\tilde{T}_{\tilde{s}} \subseteq \tilde{T}_s$ và, bằng một đối số tương tự như trong chứng minh Bất biến 14, nếu tx $\in \bar{T}_i$ thì không ở trong T^*, nó phải ở $\tilde{T}_s$. Do đó, $\bar{C}_i \subseteq C_{\text{chain}}$. Hơn nữa, bởi Bất biến 14, bất biến vẫn được thỏa mãn.

Bất biến 16. $\bar{C}_i \subseteq \hat{S}_i$.
Chứng minh. Các bên trung thực sẽ chỉ xuất ra (**conf**, tx) nếu tồn tại một đa chữ ký hợp lệ cho tx, điều này thể hiện rằng mỗi bên trung thực sẽ xuất ra (**saw**, tx) ngay trước khi họ ký tx.

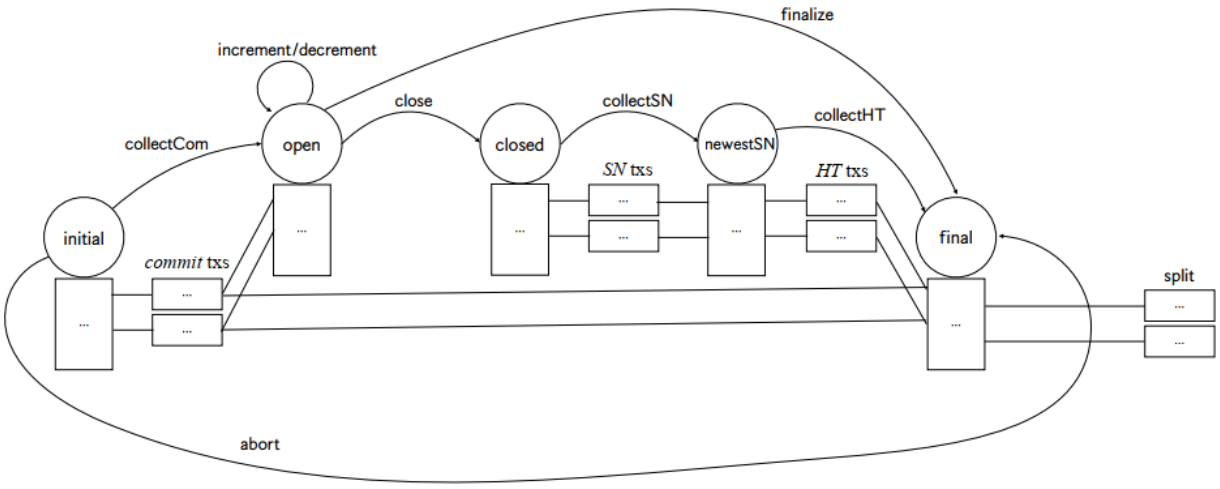
Bất biến 17. $\tilde{T}_j \subseteq \cap_{i \in H} \bar{C}_i$.
Chứng minh. Chỉ những giao dịch đã được xác nhận bởi tất cả các bên trung thực mới có thể được đưa vào ảnh chụp nhanh.

Bất biến 18. $C_{\text{chain}} \subseteq \cap_{i \in H} \hat{S}_i$.
Chứng minh. Cho $\eta = (U, s, T)$. Quan sát thấy rằng $C_{\text{chain}} = \tilde{T}_s \cup T$. *Hãy xem xét một giao dịch* tx $\in C_{\text{chain}}$.

- Nếu tx $\in \tilde{T}_s$ thì tx $\in \cap_{i \in H} \bar{C}_i \subseteq \cap_{i \in H} \hat{S}_i$ bởi Bất biến 17 và 16.
- Nếu tx $\in T$ thì tx $\in \cap_{i \in H} \hat{S}_i$ vì không có giao dịch nào có thể được xác nhận mà không bị tất cả các bên trung thực nhìn thấy.

Bổ đề 8. *Giao thức head với CR thỏa mãn thuộc tính tính hợp lý.*
Chứng minh. Gọi $\eta = (U, s, T)$ là giá trị của η ngay trước khi áp dụng **Final**(η, U_{final}). Rõ ràng, tập U_{final} duy nhất sẽ được **Final** chấp nhận là $U_0 \circ (\tilde{T}_s \cup T)$. Theo định nghĩa $\tilde{T}_s \cup T = C_{\text{chain}}$. Tính hợp lý bây giờ thể hiện theo Bất biến 18.

Bổ đề 9. *Giao thức Head với CR thỏa mãn thuộc tính tính hoàn chỉnh.*
Chứng minh. Bổ đề thể hiện từ Bất biến 15 và một đối số tương tự như trong chứng minh Bổ đề 8.



Hình 20: Biểu đồ trạng thái chuỗi chính với (a) cam kết và ngừng cam kết tăng thêm, (b) kết thúc lạc quan và (c) giai đoạn tranh chấp song song.

C. Máy trạng thái chuỗi chính đầy đủ

Phần này phác thảo những bổ sung sau cho giao thức cơ bản:

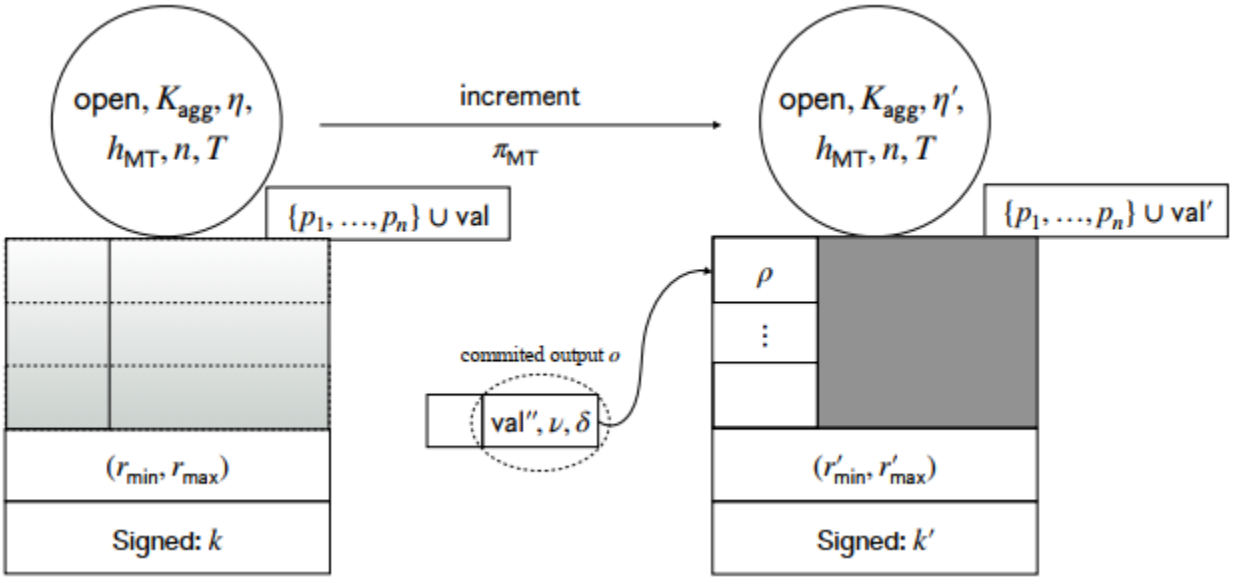
- *Cam kết và ngừng cam kết tăng thêm:* Những điều này cho phép các thành viên Head, trong khi Head đang mở, (1) cam kết các UTxO mới cho Head và (2) loại bỏ các UTxO khỏi Head.
- *Một quy trình hoàn thiện lạc quan:* Nếu tất cả các thành viên Head đồng ý đóng một Head, quy trình này cho phép quyết toán Head với một giao dịch.
- *Một cách hiệu quả hơn để đóng Head:* Đối với các trường hợp Head bị đóng do mạng chậm và / hoặc các thành viên Head bị hỏng, quy trình này cho phép đóng Head với thời gian tranh chấp ngắn. Hơn nữa, quy trình đóng mới được thiết kế theo cách giữ cho kích thước của các giao dịch chuỗi chính nhỏ.

Biểu đồ trạng thái tương ứng với máy trạng thái (SM) chuỗi chính đầy đủ được mô tả trong Hình 20. Các giao dịch được sử dụng để triển khai các tính năng trên được giải thích trong Phần C.1 và sử dụng các thuật toán xác minh On-Chain (OCV) bổ sung **Increment**, **Decrement**, **Finalize**, **Snapshot**, **ValidSN**, **ValidHT** và **Fanout** cũng như **Close** và **Final** đã được sửa đổi. Các thuật toán OCV bổ sung cũng như các thay đổi đối với giao thức Head được giải thích trong Phần C.3, sau khi xác định một biến thể của cây Merkle-Patricia trong Phần C.2.

C.1. Các giao dịch chuỗi chính mới

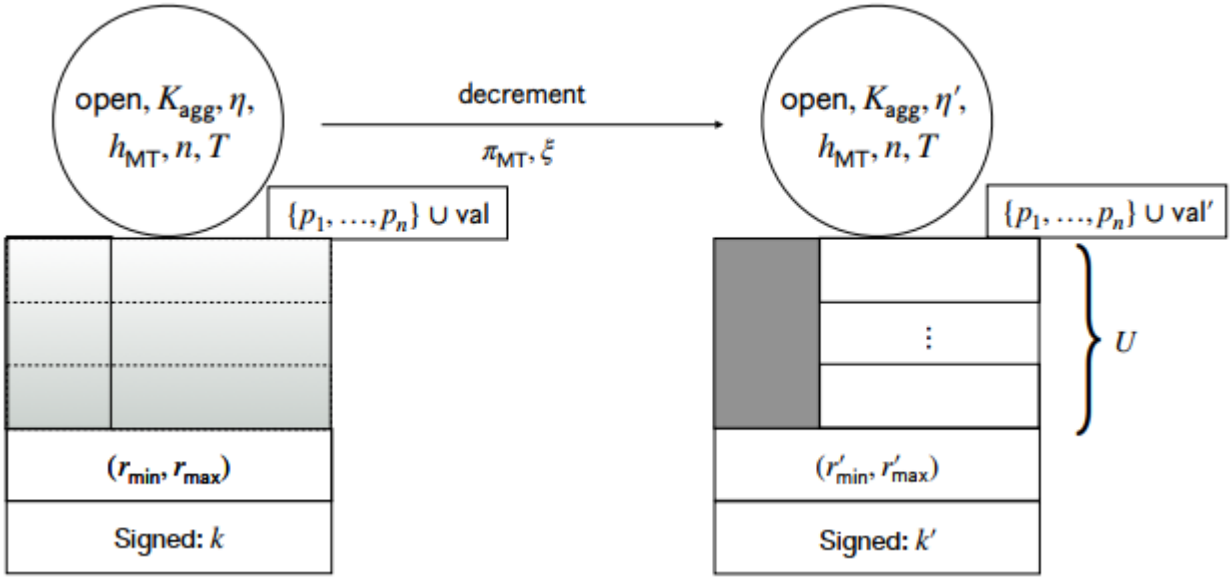
C.1.1. Cam kết và ngừng cam kết tăng thêm

Trong giao thức cơ bản, các UTxO chỉ có thể được cam kết với một Head trước khi nó đạt đến trạng thái mở. Khi Head đang chạy, không thể thêm UTxO nào vào nó. Tương tự, cách duy nhất để giải phóng UTxO trong Head và làm cho chúng có sẵn để chi tiêu trên chuỗi chính là đóng Head. Các cam kết và ngừng cam kết tăng thêm cho phép các UTxO tùy ý được thêm vào và loại bỏ tương ứng khỏi Head, trong khi Head đang mở.



Hình 21: *Giao dịch collectCom* / *gia tăng* / *giảm đi* (bên trái) với giao dịch *gia tăng* (bên phải).

Cam kết tăng thêm. Để thêm UTxO vào Head, thành viên Head có thể đăng một giao dịch *gia tăng* (xem Hình 21), gây ra sự chuyển đổi trạng thái từ mở sang chính nó. Giao dịch *gia tăng* có thể có bất kỳ số lượng đầu vào nào (nhưng ít nhất một) tiêu thụ các đầu ra mới được cam kết. Gọi *U* là tập các đầu ra *o* như vậy; hàm OCV **Increment** xử lý thông tin này và xuất ra trạng thái Head cập nhật $\eta' \leftarrow$ **Increment**(η, *U*).



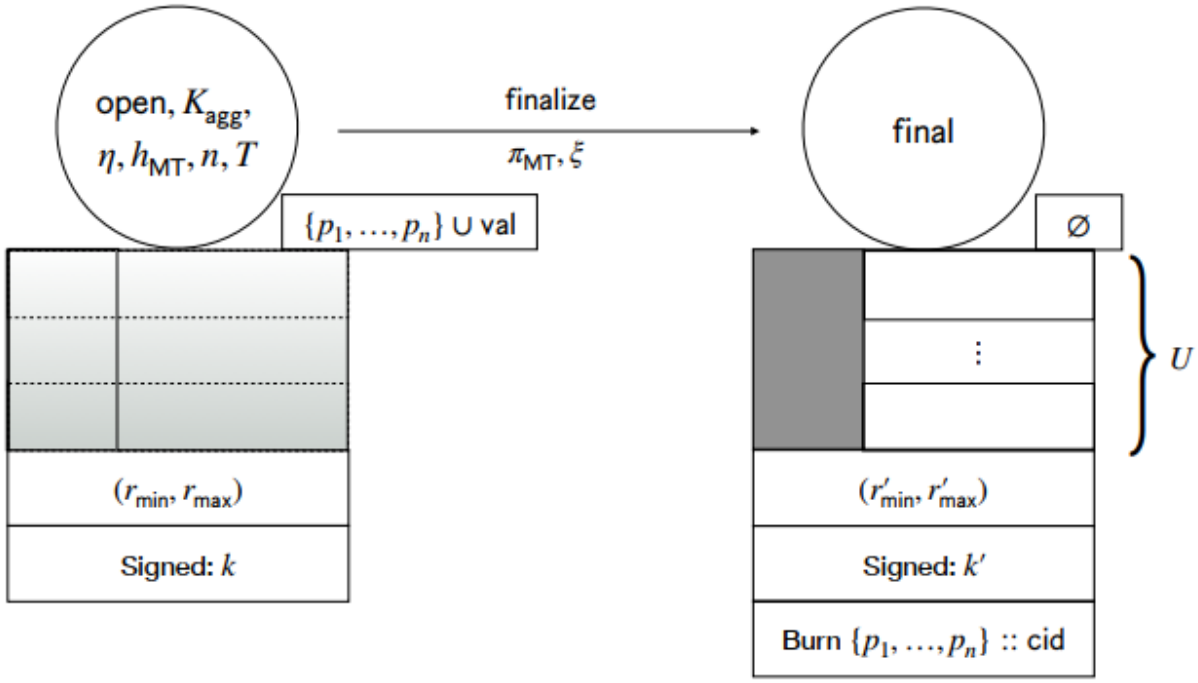
Hình 22: *Giao dịch collectCom* / *gia tăng* / *giảm đi* (bên trái) với giao dịch *giảm dần* (bên phải).

Ngừng cam kết tăng thêm. Một thành viên Head muốn làm cho UTxO bên trong Head có sẵn trên chuỗi chính sẽ đăng một giao dịch *giảm đi* (xem Hình 22), một lần nữa gây ra quá trình chuyển đổi từ mở sang chính nó. Giao dịch *giảm dần* có thể có bất kỳ số lượng đầu ra nào (nhưng ít nhất một) làm cho các đầu ra mới được loại bỏ có sẵn trên chuỗi chính. Gọi U là tập các đầu ra như vậy; hàm OCV **Decrement** xử lý thông tin này cùng với chứng chỉ ξ do các thành viên Head tạo ra để cho phép hoạt động ngừng cam kết. **Decrement** xuất ra trạng thái Head được cập nhật $\eta' \leftarrow$ **Decrement**(K_{agg}, η, ξ, U); nó cũng có thể xuất ra $\perp$, nhưng để một giao dịch đóng có hiệu lực thì cần phải có $\eta' \neq \perp$.

C.1.2. Đóng Head lạc quan

Nếu tất cả các thành viên Head đồng ý rằng một Head nên được đóng lại, giai đoạn đóng / tranh chấp có thể được bỏ qua và, bằng cách đăng một giao dịch *hoàn tất* (xem Hình 23), Head SM có thể được thực hiện từ trạng thái **open** đến trạng thái **final** ngay lập tức. Giao dịch cuối cùng phải có đầu ra tương ứng với trạng thái Head cuối cùng theo thỏa thuận của các bên. Để đạt được điều đó, vị từ OCV **Finalize**(η, K_{agg}, U, ξ) kiểm tra đầu ra của giao dịch tập hợp U so với chứng chỉ đặc biệt ξ do Redeemer cung cấp và thông tin được ghi trong η. Chứng chỉ ξ bao gồm một giá trị

đa chữ ký h||"final", trong đó h là Hash của bộ UTxO cuối cùng. Giao dịch *hoàn tất* chỉ hợp lệ nếu **Final** cho kết quả là True. Hơn nữa, tất cả các Token tham gia phải được đốt.



Hình 23: Giao dịch *thu thập* / *gia tăng* / *giảm đi* (bên trái) với giao dịch *hoàn tất* (bên phải).

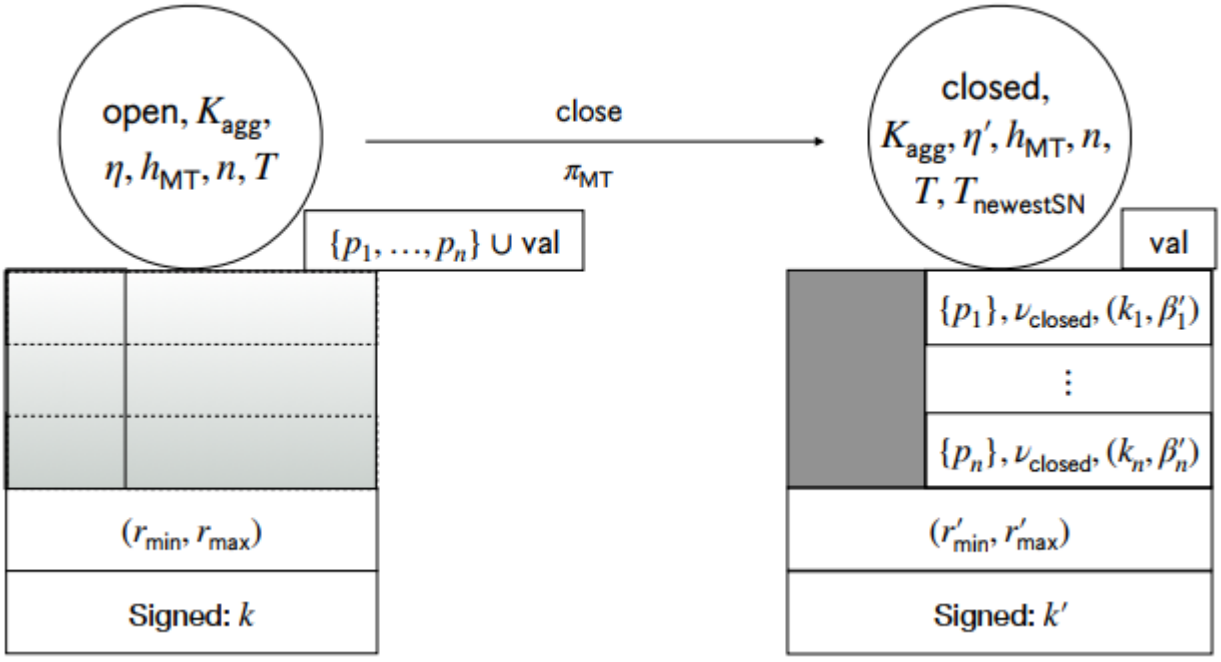
C.1.3. Giai đoạn tranh chấp hiệu quả

Hãy nhớ rằng trong giao thức đơn giản, để kết thúc một Head, một số bên p trước tiên đăng một giao dịch *đóng*, giao dịch này cũng chứa thông tin về trạng thái Head hiện tại. Trong giai đoạn tranh chấp tuần tự tiếp theo, mỗi bên có cơ hội cung cấp thêm thông tin mới nhất trong trường hợp thông tin của p đã lỗi thời hoặc p bị hỏng. Trong trường hợp xấu nhất, quá trình này yêu cầu một chuỗi n giao dịch chuỗi chính. Để tránh vấn đề này, một giai đoạn tranh chấp song song có liên quan nhiều hơn có thể được sử dụng để đóng một Head.

Giai đoạn tranh chấp song song này được điều chỉnh cho phù hợp với giao thức Head thực tế đang được sử dụng (xem Phần 6) trong đó trước tiên nó thu thập song song các đề xuất cho *ảnh chụp nhanh* gần nhất và sau đó cũng song song với *giao dịch treo* (Hanging Transaction), là các giao dịch *Head* đã xác nhận chưa được xem

xét bằng ảnh chụp nhanh. Lý do thực hiện việc đóng Head trong hai bước là việc thu thập ảnh chụp nhanh trước tiên ngăn thành viên Head bị hỏng đăng ảnh chụp nhanh đã cũ cùng với một số lượng lớn các giao dịch treo.

Do đó, để đóng một Head, một giao dịch *đóng* được đăng bởi một số thành viên Head. Giao dịch *đóng* có *n* đầu ra — một đầu ra cho mỗi thành viên Head — mà giao dịch *SN* có thể được gắn vào. Giao dịch *SN* cho phép một thành viên Head đăng (thông tin về) ảnh chụp nhanh mới nhất lên chuỗi chính. Giao dịch tiếp theo, giao dịch *collectSN*, thu thập tất cả các giao dịch *SN* và chọn ảnh chụp nhanh gần nhất. Nó cũng có *n* đầu ra, mà mỗi bên có thể đính kèm một giao dịch *HT*. Mỗi giao dịch *HT* cho phép một thành viên Head đăng (thông tin về) các giao dịch treo. Cuối cùng, giao dịch *collectHT* thu thập tất cả các giao dịch *HT* và xác định bộ UTxO cuối cùng. Giao dịch *collectHT* cũng xác định cách phân tách bộ UTxO cuối cùng và cung cấp đủ nhiều đầu ra cho các giao dịch *phân tách* được đính kèm, trong đó mỗi giao dịch *phân tách* có đầu ra là một tập con của bộ UTxO cuối cùng.



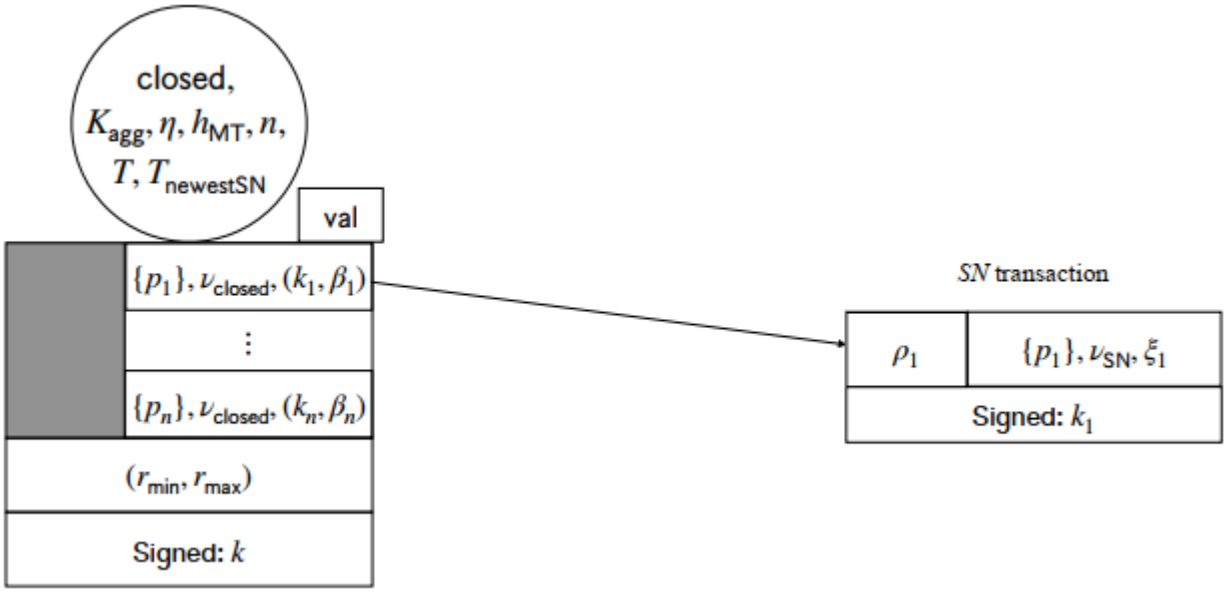
Hình 24: Giao dịch *collectCom / gia tăng / giảm đi* (bên trái) với giao dịch *đóng* (bên phải).

Bắt đầu đóng Head. Để đóng một Head, một thành viên Head có thể đăng giao dịch *đóng* (xem Hình 24), dẫn đến chuyển trạng thái từ **open** sang **closed**. So với giao thức cơ bản, người đăng không bắt buộc phải cung cấp bất kỳ thông tin nào về trạng thái của Head tại thời điểm này. Thay vào đó, giao dịch *đóng* có *n* đầu ra bị khóa bởi

trình xác thực v_{closed}. Đầu ra thứ *i* có (k_i, β'_i) trong trường dữ liệu của nó, trong đó k_1, ..., k_n là các khóa công khai được Hash trong h_{MT} và β'_i chứa thông tin được yêu cầu bởi thuật toán OCV **ValidSN** để xác minh giao dịch *SN* được đăng (xem bên dưới). Cụ thể, chúng được tạo bởi thuật toán OCV $(\eta', \beta'_1, \ldots, \beta'_n \leftarrow$ **Close**(K_{agg}, η), cũng được phép cập nhật trạng thái Head. Quan sát thấy rằng giao dịch *đóng* cũng đặt *n* Token tham gia vào đầu ra.

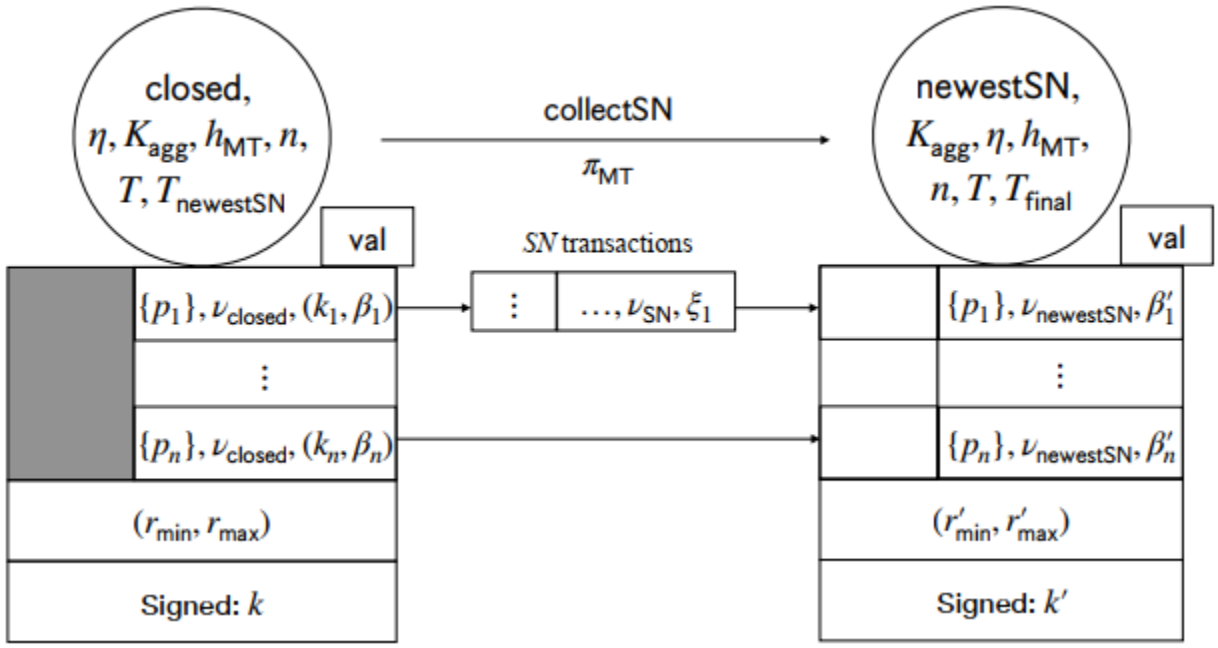
Trình xác thực v_{closed} đảm bảo những điều sau: hoặc đầu ra được tiêu thụ bởi

1. Một giao dịch SM *collectSN* (xem bên dưới) hoặc
2. Một giao dịch *SN* (được xác định bằng cách có trình xác thực v_{SN} trong đầu ra duy nhất của nó) và
    a) Giao dịch được ký bởi k_i,
    b) Thuật toán OCV **ValidSN**$(\beta'_i, \xi_i, \rho_i)$ trả về True, trong đó ξ_i và ρ_i trong trường dữ liệu đầu ra của giao dịch *SN* tương ứng. Redeemer chứa thông tin ảnh chụp nhanh (xem bên dưới và Hình 25).



Hình 25: Giao dịch *đóng* (bên trái) với giao dịch *SN* (bên phải).

Sau khi một giao dịch *đóng* đã được đăng, một *khoảng thời gian đăng trên SN* sẽ bắt đầu kéo dài ít nhất *T* slot. Do đó, Slot cuối cùng $T_{newestSN}$ của khoảng thời gian nói trên được ghi lại trong trạng thái và nó được đảm bảo rằng $T_{newestSN} \geq r'_{max} + T$.



Hình 26: Giao dịch *collectSN* (bên trái) với giao dịch *collectHT* (bên phải) và giao dịch *HT* (ở giữa).

Cung cấp thông tin ảnh chụp nhanh. Trong giao dịch *SN* (xem Hình 25), một bên chỉ cần cung cấp thông tin về ảnh chụp nhanh gần nhất của họ trong Redemeer ρ và trường dữ liệu đầu ra ξ; ρ chỉ chứa dữ liệu có liên quan để xác minh chính giao dịch *SN*, trong khi ξ chứa thông tin liên quan đến SM. Cụ thể, ξ chứa $h||s$, trong đó h là Hash và s là số lượng ảnh chụp nhanh mới nhất, và ρ chứa đa chữ ký trên $h||s$. Tất cả các giao dịch *SN* được thu thập bởi một giao dịch SM *collectSN*.

Thu thập thông tin ảnh chụp nhanh. Giao dịch *collectSN* (xem Hình 26) khiến SM chuyển từ trạng thái **closed** sang **newestSN**. Hàm OCV **Snapshot** chịu trách nhiệm thu thập các giá trị mà ξ_i đã cung cấp trong các giao dịch *SN* và tính toán trạng thái Head η' mới là $(\eta', \beta'_1, \ldots, \beta'_n) \leftarrow$ **Snapshot**$(K_{agg}, \eta, \xi_1, \ldots, \xi_n)$, trong đó β'_i có mục đích tương tự như mục đích của β'_i trong giao dịch *đóng* .

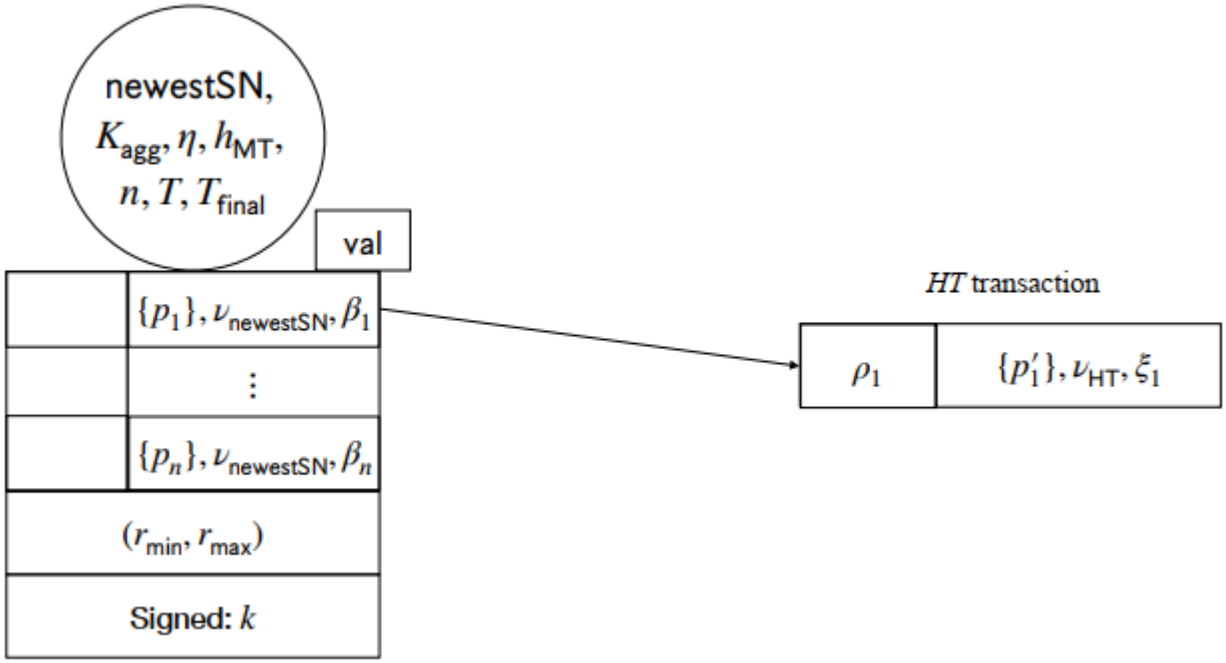
Giao dịch *collectSN* có n đầu ra, mỗi đầu ra được khóa bởi trình xác thực $v_{newestSN}$, đảm bảo điều sau: hoặc đầu ra được sử dụng bởi

1. Giao dịch SM *collectHT* (xem bên dưới) hoặc
2. Một giao dịch hangingTx (được xác định bằng cách có trình xác thực v_{HT} trong đầu ra duy nhất của nó) và
    a) Giao dịch được ký bởi k_i,

b) Thuật toán OCV **ValidHT** (β'_i, ξ_i, ρ_i) trả về True, trong đó ξ_i và ρ_i chứa thông tin về các giao dịch treo (xem Hình 27).

Khi một giao dịch *collectSN* đã được đăng, một *khoảng thời gian đăng hangingTx* sẽ bắt đầu kéo dài ít nhất T Slot. Do đó, Slot cuối cùng T_{final} của khoảng thời gian nói trên được ghi lại ở trạng thái và được đảm bảo rằng $T_{\text{final}} \geq r'_{\max} + T$.

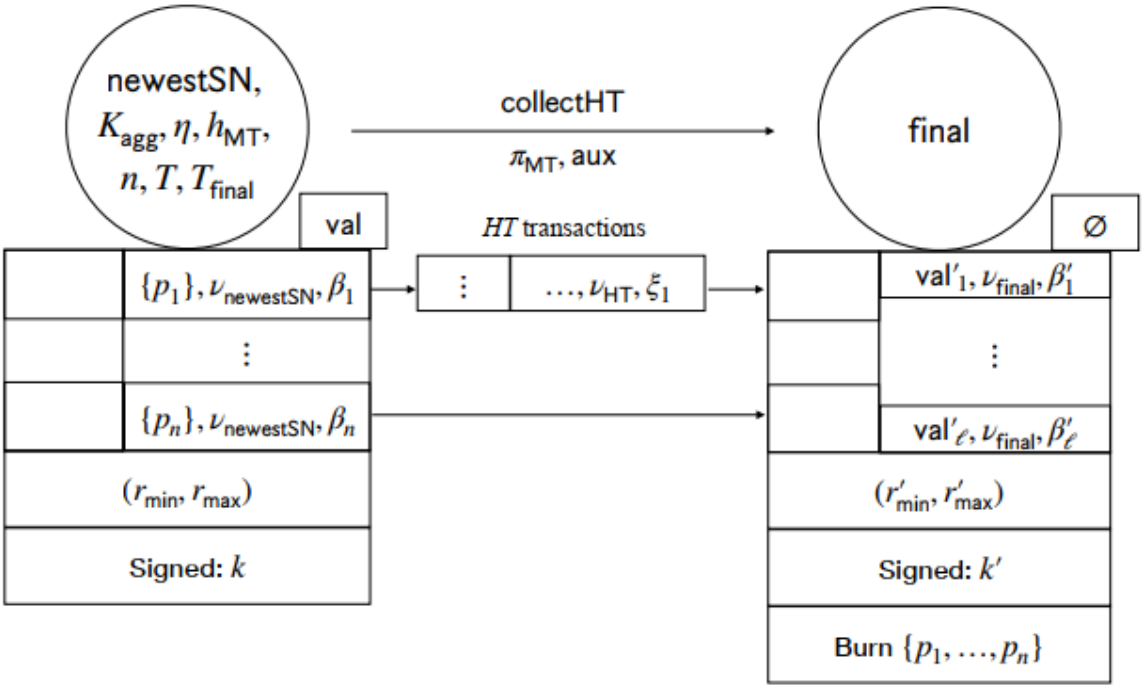
Cung cấp các giao dịch treo. Trong giao dịch *HT* (xem Hình 27), một bên chỉ cần cung cấp thông tin về các giao dịch treo không có trong ảnh chụp nhanh gần nhất được cung cấp trong giai đoạn ảnh chụp nhanh. Như với các giao dịch *SN*, thông tin này được phân tách giữa Redeemer và trường dữ liệu đầu ra. Tất cả các giao dịch *HT* được thu thập bởi một giao dịch *collectHT* của *SM* .



Hình 27: Giao dịch *collectSN* (bên trái) với giao dịch *HT* (bên phải).

Thu thập giao dịch treo. Giao dịch *collectHT* (xem Hình 28) thu thập tất cả thông tin mà ξ_i cung cấp bởi các giao dịch *HT* để tính toán bộ UTxO cuối cùng (tức là bộ UTxO là kết quả từ việc áp dụng tất cả các giao dịch treo vào ảnh chụp nhanh mới nhất) và để xác định cách thực hiện phân vùng bộ UTxO thành ℓ thành phần (để tránh đăng các giao dịch lớn trên chuỗi chính). Cụ thể, hàm OCV **Fanout** lấy làm khóa đầu vào K_{agg}, η, thông tin bổ trợ **aux** trong Redeemer, và các giá trị ξ_i và tính toán (**val**'$_i$, β'_i) trong đó **val**'$_i$ là giá trị trong phân vùng thứ i và β'_i được sử dụng để xác thực giao dịch *phân tách* tương ứng. Giao dịch cuối cùng cũng phải ghi các

Token tham gia p_1, ..., p_n và chỉ có thể được đăng sau khi giai đoạn *HT* hoàn thành, tức là chỉ khi nào $r'_{\text{min}} \geq T_{\text{final}}$.



Hình 28: Giao dịch *đóng* (bên trái) với giao dịch *collectSN* (bên phải) và giao dịch *SN* (ở giữa).

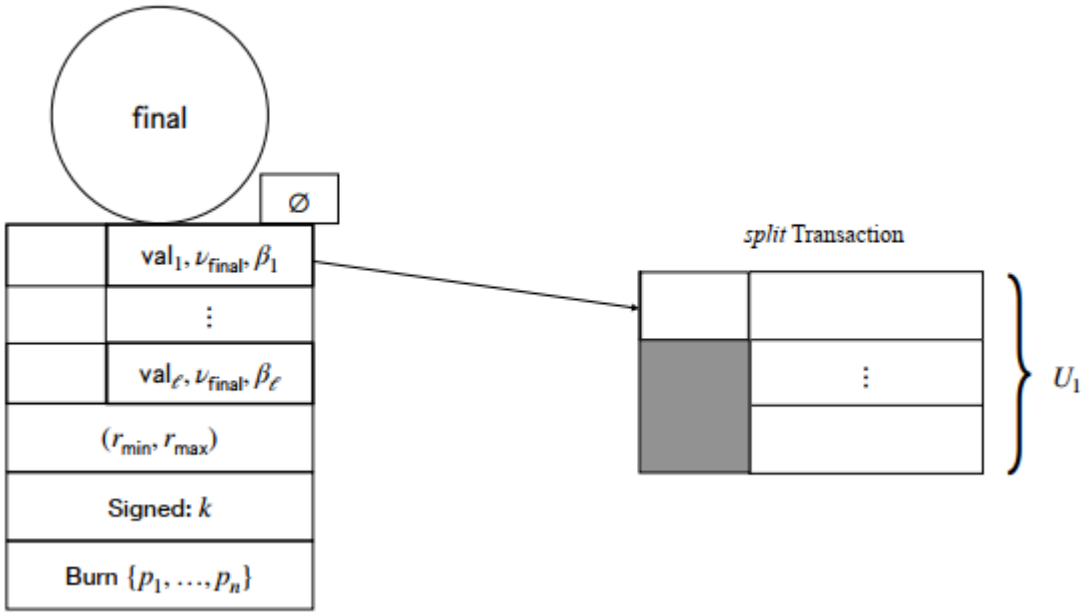
Phân tách bộ UTxO cuối cùng. Nhiệm vụ của các giao dịch *phân tách* (xem Hình 29) là làm cho các UTxO trong một phân vùng cụ thể (do **Fanout** xác định) có sẵn để sử dụng trên chuỗi chính. Để xác minh điều này được thực hiện đúng, trình xác thực v_{final} chạy vị từ OCV **Final**(β_i, U_i), trong đó U_i là tập hợp các đầu ra của giao dịch *phân tách* .

C.2. Bộ UTxO và Cây Merkle-Patricia

Giao thức Head và các thuật toán OCV cho các giao thức Hydra đầy đủ sử dụng một biến thể của *Cây Merkle-Patricia (MPTs)* [22]. MPTs của Hydra lưu trữ một tập hợp các cặp *D* outref / output (**out-ref**, *o*) theo cách mà

- *(Thuyết xác định)* tập hợp *D* xác định cây (tức là thứ tự chèn và loại bỏ không ảnh hưởng đến hình dạng của cây),

- *(Hashing) M*ột cây có thể được Hash, tức là *Hash gốc* (Root Hash) có thể được tính toán theo cách tính toán khó có thể tìm được hai cây (hoặc tương đương, hai tập *D* và *D*') có cùng một Hash,



Hình 29: Giao dịch *collectHT* với giao dịch *phân tách* .

- *(Bằng chứng thành viên)* Thành viên của một cặp (**out-ref**, *o*) trong cây có thể được xác minh bằng cách sử dụng Hash gốc và thông tin bổ trợ **aux** có kích thước $O(\log|D|)$,
- *(Xóa và thêm)* cho một Hash gốc h_{root} tương ứng với tập D và đặt $R \subseteq D$ và A với $A \cap D = \emptyset$, Hash gốc tương ứng với $D \setminus R \cup A$ có thể được tính toán từ h_{root} và thông tin bổ trợ **aux** có kích thước $O(\log|D|)$, và
- *(Phân tách)* cho một số B bất kỳ, một cây tương ứng với D có thể được phân tách thành các cây con tương ứng với các tập rời rạc $D_1, \ldots, D_\ell$ (đối với một số ℓ) với $D_1 \cup \ldots \cup D_\ell = D$ sao cho
  - Các phần tử của mỗi D_i có kích thước tối đa là B Bit, và
  - Các giá trị Hash gốc và tổng giá trị tương ứng với mỗi D_i có thể được tính bằng cách sử dụng Hash gốc h_{root} của D và thông tin bổ trợ **aux** có kích thước B.[11]

[11]Quan sát thấy rằng điều này chỉ hoạt động nếu cây không vượt quá một kích thước tối đa nhất định.

Xác định MPTs. MPTs được Hydra sử dụng có kích thước bảng chữ cái $A = 16$. MPT tương ứng với tập D được xác định thông qua thuật toán **MPT-Build** trong Hình 30. Lưu ý rằng outrefs đảm nhận vai trò của "khóa" và đầu ra o của "giá trị". MPT được xác định một cách đệ quy, trong đó Node gốc **node** = (**pre**, H, S, V) của cây tương ứng với D với $D > 1$ có các trường sau:

- *Tiền tố (Prefix):* Trường **pre** lưu trữ tiền tố chung của tất cả các khóa được tìm thấy trong D.
- *Chidren:* Mảng H lưu trữ Hash của tất cả các Node con A, trong đó $H[i] = \perp$ nếu Node con tương ứng không có mặt. Con thứ i của **node** là gốc của MPT chứa tất cả các phần tử tương ứng với tập D'' được tính như sau:
    1. Gọi tập D' là tập nhận được bằng cách loại bỏ tiền tố **pre** từ mọi khóa **out-ref** trong mọi cặp (**out-ref**, o) $\in D$; điều này được biểu thị là (**pre**, D') $\leftarrow$ **CP** (D) trong Hình 30.
    2. Gọi D'' là tập nhận được bằng cách loại bỏ thêm ký tự i khỏi mỗi **out-ref**; điều này được biểu thị là **Proj**(D', i) trong Hình 30.
- *Kích thước (Size):* Với mỗi $i \in [A]$, $S[i]$ ghi kích thước tổng hợp của các lá của cây con ở con thứ i. Kích thước tổng hợp của tất cả các lá trong cây con của **node** là **Sum**(S), trong đó $\text{Sum}(S) = \sum_{i \in [A]} S[i]$.
- *Giá trị (Value):* Với mỗi $i \in [A]$, $V[i]$ ghi lại giá trị của cây con ở con thứ i. Giá trị của cây con của **node** là Sum(V), trong đó $\text{Sum}(V) = \sum_{i \in [A]} V[i]$.

Các Node lá **leaf** = (**pre**, o) tương ứng với một phần tử đơn lẻ $D = \{(\textbf{pre}, o)\}$. Kích thước của chúng được cho bởi **Size**(**leaf**) và giá trị của chúng là **val**, trong đó o = (**val**, v, δ).

Hashing. Hash của MPT chỉ đơn giản là Hash của Node gốc của nó.

Bằng chứng thành viên. Để cung cấp bằng chứng rằng một số **out-ref** xuất hiện trong một MPT với Hash gốc h_{root}, chỉ cần cung cấp thông tin bổ trợ **aux** cho các Node trên đường dẫn từ gốc đến lá chứa o là đủ. Hàm xác minh tương ứng được ký hiệu là **MPT-VfyMemb**(h, **out-ref**, **aux**).

Loại bỏ và thêm vào. Tương tự như bằng chứng thành viên,

- Để xóa một cặp (**out-ref**, o) khỏi MPT với Hash gốc h_{root}, Hash gốc mới có thể được tính toán nếu được cung cấp dưới dạng thông tin bổ trợ **aux** cho các Node trên đường dẫn từ gốc đến Node đã bị xóa, trong đó, trong các trường

hợp mà Node đó chỉ có một anh chị em, anh chị em đó cũng phải được cung cấp;

- Để thêm một cặp (**out-ref**, o) vào MPT với Hash gốc h_{root}, Hash gốc mới có thể được tính nếu được cung cấp dưới dạng thông tin bổ trợ **aux** cho các Node trên đường dẫn từ gốc đến Node nơi **out-ref** phân kỳ từ tiền tố đi qua.

```
MPT-Build (D)
    if |D| > 1
        (pre, D') ← CP(D)
        H[·], S[·], V[·] ← ε
        for i ∈ [A]
            (H[i], S[i], V[i]) ← MPT-Build(Proj(D', i))
        node ← (pre, H, S, V)
        h ← H(node)
        N[h] ← node
        return (h, Sum(S), Sum(V))
    else if |D| = 1
        {(pre, o)} ← D
        leaf ← (pre, o)
        h ← H(leaf)
        N[h] ← leaf
        (val, ·, ·) ← o
        return (h, Size(leaf), val)
    else
        return (⊥, 0, ∅)
```

Hình 30: Quy trình đệ quy để xây dựng một MPT từ tập D các cặp outref/output (**out-ref**, *o*). Thuật toán lưu trữ các Node trong mảng N được lập chỉ mục theo giá trị Hash của chúng và trả về giá trị Hash của Node gốc cũng như tổng kích thước và giá trị của toàn bộ cây.

Để loại bỏ toàn bộ tập $R \subseteq D$ của các cặp outref / output và sau đó thêm một tập A với $A \cap D = \emptyset$ (điều xảy ra khi một giao dịch được áp dụng cho một bộ UTxO), các thao tác trên có thể được nối đơn giản, tạo ra một chuỗi phụ kết hợp **aux**. Hàm tính

toán Hash gốc mới h'_{root} từ Hash gốc cũ h_{root}, các bộ R và A, và A được ký hiệu bằng $h'_{\text{root}} \leftarrow$ **MPT-CompRA** (h_{root}, R, A, **aux**).

Phân tách. Để phân tách, như đã mô tả ở trên, một cây có các Node N, đầu tiên, mỗi Node **node** = (**pre**, H, S, V) mà cây con có các lá với kích thước kết hợp $\sum_i S[i] > B$ được thêm vào một danh sách phân tách (được lập chỉ mục bởi các Hash của Node), được gọi là *biên giới phân tách* (Split Frontier). Khi đó, mọi Node **node** ≠ **split** có bố mẹ trong biên giới phân tách là gốc của cây con tương ứng với một trong các tập con D_i. Bằng cách này, kích thước tổng hợp của tất cả các phần tử trong mỗi D_i tối đa là B (nếu không, **node** sẽ nằm trong tập phân tách). Ký hiệu các Node gốc này là $\mathbf{node}_1$, ..., $\mathbf{node}_\ell$ và gọi chúng là *các Node phân tách* .

Để tính toán các giá trị Hash h_1, ..., h_ℓ và các giá trị $\mathbf{val}_1$, ..., $\mathbf{val}_\ell$ của các Node phân tách từ Hash gốc h_{root} của Node gốc của toàn bộ cây, **aux** chỉ bao gồm các biên giới phân tách **split** (bao gồm các giá trị và Hash đã nói).

Cuối cùng, đối với mỗi Node phân tách $\mathbf{node}_i$, hãy xác định *tiền tố phân tách* $\mathbf{pre}_i$ là tiền tố chung của tất cả các outrefs trong D_i. Tiền tố phân tách sẽ cần thiết để tính Hash h_i cho các tập D_i : h_i thu được bằng cách tính MPT tương ứng với D_i, nhưng bằng cách loại bỏ $\mathbf{pre}_i$ khỏi tiền tố **pre** trong Node gốc kết quả trước khi Hash nó.

Hàm tính toán các giá trị trên được ký hiệu là

(h_1, ..., h_ℓ, $\mathbf{val}_1$, ..., $\mathbf{val}_\ell$, $\mathbf{pre}_1$, ..., $\mathbf{pre}_\ell$) ← **MPT-CompSpl** (h_{root}, B, **aux**).

C.3. Giao thức Head và xác minh trên chuỗi

Để được sử dụng với SM cải tiến (xem Hình 20), một số thay đổi nhỏ phải được thực hiện trong giao thức Head. Phần này tóm tắt những thay đổi này và mô tả ở mức độ cao cách các hàm OCV có thể được triển khai để hoạt động với SM được cải tiến.

Cây Merkle-Patricia, bộ UTxO và giao dịch. Giao thức Head và các thuật toán OCV có thể được triển khai theo cách mà ngoài các giao dịch phân tách, chỉ cần đăng thông tin Hash về các ảnh chụp nhanh và các giao dịch treo. Bằng cách đó, các giao dịch trên chuỗi chính của Hydra vẫn ở mức nhỏ ngay cả khi bộ UTxO của Head trở nên lớn hoặc có nhiều giao dịch treo.

Hãy nhớ rằng UTxO chỉ đơn giản là một cặp u = (**out-ref**, o) của outref và output. Giao thức Head đầy đủ duy trì bộ UTxO hiện tại bằng cách lưu trữ tất cả các UTxO trong một MPT như được trình bày trong Phần C.2. Khi tạo ảnh chụp nhanh mới, các bên ký vào Hash gốc của MPT (thay vì Hash đơn giản của bộ UTxO).

Hãy lưu ý rằng việc áp dụng giao dịch cho một bộ UTxO luôn bao gồm việc xóa một số UTxO và thêm một số UTxO mới. Do đó, việc phát triển Hash tương ứng với bộ UTxO để bao gồm một giao dịch mới bao gồm các thao tác xóa và thêm đơn giản trên MPT.

Để giữ cho các giao dịch *HT* nhỏ (xem bên dưới), khi xác nhận một giao dịch tx = (I, O, $\mathbf{val}_{\text{Forge}}$, r, K), tx được Hash bằng việc tính toán

$$H(\mathbf{ID}(\text{tx}), \mathbf{out\text{-}ref}_1, ..., \mathbf{out\text{-}ref}_w, o_1, ..., o_{w'}, h_{\text{rest}}),$$

trong đó

- **ID**(tx) là ID của tx theo quy tắc sổ cái,
- $I = \{i_1, ..., i_w\}$ và $i_j = (\mathbf{out\text{-}ref}_j, \rho_j)$,
- $O = (o_1, ..., o_{w'})$ và
- $h_{\text{rest}} = H(\rho_1, ..., \rho_w, \mathbf{val}_{\text{Forge}}, r, K)$ là Hash của phần còn lại của giao dịch.

Đối với các bằng chứng thành viên /chèn /xóa MPT, cách của Hash này chỉ cho phép cung cấp ID **ID**(tx), các tham chiếu đầu ra trong I, đầu ra O và Hash h_{rest}, thường ngắn hơn nhiều so với toàn bộ giao dịch.

Hàm xác minh On-Chain. Sử dụng MPTs, các hàm OCV cho quá trình ngừng cam kết hiệu quả có thể được thực hiện như sau:

- $\eta' \leftarrow \mathbf{Initial}(U_1, ..., U_n)$ tính toán MPT tương ứng với sự kết hợp của bộ UTxO U_i và lưu trữ Hash trong đầu ra η'.
- $(\eta', \beta'_1, \beta'_n) \leftarrow \mathbf{Close}(\eta)$ để lại $\eta' = \eta$ không đổi và đặt K_{agg} vào mỗi β'_i.
- **ValidSN** (β, ρ, ξ) sử dụng K_{agg} (được lưu trữ trong β) để xác minh đa chữ ký (được lưu trữ trong ρ) trên Hash MPT và số ảnh chụp nhanh (được lưu trữ trong ξ). Thuật toán trả về True khi và chỉ khi chữ ký xác minh và số ảnh chụp nhanh lớn hơn 0.
- $(\eta', \beta'_1, \beta'_n) \leftarrow \mathbf{Snapshot}(\eta, \xi_1, ..., \xi_n)$ chỉ cần chọn ξ_i có số ảnh chụp nhanh cao nhất và lưu trữ Hash gốc MPT tương ứng trong η'. Mỗi đầu ra β'_i bao gồm K_{agg} cũng như Hash đã nói. Hãy lưu ý rằng nếu tất cả ξ_i đều trống (vì không có bên nào đăng giao dịch *SN* hợp lệ), thì Hash ban đầu (vẫn ở η) được tính trong **Initial** sẽ được sử dụng.
- **ValidHT**(β, ρ, ξ) có liên quan nhiều hơn. Hãy nhớ rằng trình xác thực này kiểm tra một giao dịch *HT*, qua đó một số bên đăng các giao dịch bị treo trong

ξ cùng với các đa chứ kỹ và bằng chứng tương ứng trong ρ cho thấy rằng các giao dịch này đã được xác nhận trong Head và có thể được áp dụng cho ảnh chụp nhanh gần đây nhất, mà Hash h_{SN} được lưu trữ trong β.

Giao dịch treo tx = (I, O, $\mathbf{val}_{\text{Forge}}$, r, K) được cung cấp thông qua các giá trị

$$\tilde{tx} = (\mathbf{ID}(\text{tx}), \mathbf{out\text{-}ref}_1, ..., \text{out-ref}_w, o_1, ..., o_{w'}, h_{\text{rest}})$$

như đã xác định ở trên. Đối với mỗi giao dịch như vậy, **ValidHT** tính $h \leftarrow H(\tilde{tx})$ và kiểm tra, sử dụng K_{agg} (được lưu trữ trong β), ρ có chứa đa chữ ký hợp lệ trên h.

Khi tất cả các giao dịch đã được xác thực, **ValidHT** hiện xử lý chúng theo thứ tự cấu trúc liên kết và kiểm tra từng giao dịch tx

- Hoặc có một $\mathbf{out\text{-}ref}_i$ =(*txID, txIdx*) trong tx để *txID* đề cập đến một giao dịch đã xử lý trước đó, hoặc
- Có một bằng chứng thành viên MPT **aux** trong ρ sao cho **MPT-VfyMemb** (h_{SN}, $\mathbf{out\text{-}ref}_i$, **aux**) = True cho một số $\mathbf{out\text{-}ref}_i$ trong tx.

Hãy lưu ý rằng vì các giao dịch được ký kết nhiều lần, nên chỉ cần kiểm tra một outref duy nhất để đảm bảo rằng tx không bị cũ (tức là chưa phải ảnh chụp nhanh mới nhất được sử dụng).

- (β_1, ..., β_ℓ, $\mathbf{val}_1$, ..., $\mathbf{val}_\ell$) ← **Fanout**(η, **aux**, ξ_1, ..., ξ_n) có hai nhiệm vụ: Đầu tiên, nó phải thu thập các giao dịch treo (được lưu trữ trong các biến ξ_i) và tính toán, sử dụng Hash h_{SN} (được lưu trữ trong η) của ảnh chụp nhanh gần nhất và thông tin bổ trợ $\mathbf{aux}_1$ (được lưu trữ trong **aux**), Hash h_{final} của bộ UTxO cuối cùng; điều này có thể được thực hiện bằng hàm **MPT-CompRA**. Thứ hai, nó phải lấy thông tin bổ trợ $\mathbf{aux}_2$ (được lưu trữ trong **aux**) và tính toán sự phân tách (h_1, ..., h_ℓ, $\mathbf{val}_1$, ..., $\mathbf{val}_\ell$, $\mathbf{pre}_1$, ..., $\mathbf{pre}_\ell$) ← **MPT-CompSpl**(h_{final}, B, $\mathbf{aux}_2$). Mỗi β_i được đặt thành (h_i, $\mathbf{pre}_i$).
- **Final**(β_j, U_j) chỉ cần Hash U_j và kiểm tra xem nó có khớp với η_j hay không.

D. Các khía cạnh khác của giao thức

D.1. Tài trợ cho quá trình máy trạng thái

Để máy trạng thái (SM) chuỗi chính của giao thức Hydra tiến triển, các thành viên Head cần đăng các giao dịch tương ứng; điều này đúng cho cả giao thức đơn giản và đầy đủ. Tuy nhiên, các thành viên Head có thể quyết định chờ các thành viên Head khác đăng các giao dịch này để tiết kiệm phí. Do đó, có thể cần thiết phải cung cấp phần thưởng cho việc đăng một số giao dịch SM hoặc ít nhất, phân bổ tiền để trang trải các khoản phí phát sinh.

Các ví dụ sau đây phác thảo cách có thể trao phần thưởng cho một số giao dịch SM. Tuy nhiên, nói chung, các thành viên Head quyết định chính sách khen thưởng chính xác khi Head được thành lập:

- *Ban đầu*: Nói chung, không cần phải đưa ra phần thưởng khi đăng giao dịch *ban đầu*, chỉ cần ý định của người khởi tạo để mở Head là đủ.
- *Cam kết*: Một bên sẵn sàng tham gia vào Head sẽ đăng giao dịch *cam kết* của mình. Vì vậy, không cần phải phân bổ bất kỳ phần thưởng nào cho nó.
- *collectCom*: Vì chỉ có một giao dịch *collectCom* được đăng, một bên có thể đợi xem các thành viên Head khác có đăng giao dịch trước hay không, điều này làm chậm tiến độ của Head SM. Do đó, người ta nên khuyến khích việc đăng các giao dịch *collectCom*.
- *Hủy bỏ*: Nếu Head không được thành lập do thiếu các giao dịch *cam kết*, các thành viên Head còn lại có động cơ để huỷ bỏ Head. Tuy nhiên, khuyến khích có thể mạnh hơn đối với các bên yêu cầu số tiền bị khóa ngay lập tức. Do đó, việc thưởng cho bên đăng giao dịch *hủy bỏ* là có ý nghĩa.

Các lập luận tương tự có thể được đưa ra đối với các giao dịch *đóng*, *tranh chấp* và *fanout* cũng như đối với các giao dịch của Hydra SM đầy đủ.

Tất cả các khoản tiền cho phần thưởng được phân bổ trước trong các giao dịch cam kết, có thể được thực thi bởi giao dịch *ban đầu*. Để làm như vậy, phần thưởng yêu cầu phải được ước tính trước. Điều này đặc biệt yêu cầu giới hạn trên về kích thước của bộ UTxO của Head, số lượng giao dịch treo tối đa, v.v. Những số lượng này có thể được thực hiện rõ ràng dưới dạng tham số của Head.

Các giao dịch đưa Head SM về trạng thái cuối cùng (ví dụ: *hủy bỏ* hoặc *fanout*) sẽ đảm bảo rằng các khoản tiền thặng dư sẽ được phân phối lại một cách thích hợp.

D.2. Xử lý thời gian trong giao thức Head

Hãy nhớ rằng mỗi giao dịch chứa một Tuple $r = (r_{\min}, r_{\max})$ chỉ định phạm vi Slot $[r_{\min}, r_{\max}]$ trong đó giao dịch phải được đưa vào sổ cái để hợp lệ. Các khái niệm quan trọng về thời gian như cam kết về thời gian dựa trên cấu trúc này.

Trong chuỗi chính, có thể dễ dàng đồng ý về việc liệu một giao dịch có được đưa vào sổ cái trong một phạm vi Slot nhất định hay không bằng cách xác minh xem Slot của Block chứa nó có nằm trong phạm vi Slot đó hay không; và thực tế này sẽ được xác nhận một cách không thể thay đổi một khi Block nằm đủ sâu trong Blockchain.

Ngược lại, quá trình xác nhận trong Head là không đồng bộ, và chúng tôi yêu cầu một cơ chế khác để đưa ra quyết định này. Do đó, chúng tôi thiết lập quy tắc rằng một bên chỉ có thể ký một giao dịch nếu họ đã nhìn thấy giao dịch đó trong phạm vi Slot bao gồm r; về tổng thể, điều này thể thiện rằng tất cả các bên trung thực đã nhìn thấy một giao dịch được xác nhận "đúng thời hạn".[12]

Bây giờ, một bên có thể không tìm hiểu về xác nhận cho đến khi phạm vi Slot hết hạn, nhưng điều này về cơ bản không khác so với chuỗi chính.

Giải quyết xung đột. Một vấn đề nảy sinh với giải quyết xung đột (CR - Conflict Resolution) khi nhiều giao dịch có phạm vi Slot hữu hạn ($r_{\max} < \infty$) cạnh tranh để đổi lấy cùng một UTxO. Xung đột như vậy không thể luôn được giải quyết bằng cơ chế CR tiêu chuẩn của chúng tôi, vì một lãnh đạo ảnh chụp nhanh bao gồm cả giao dịch ưa thích của anh ta, nói chung, không thể biết liệu tx đã được (hoặc sẽ được) chấp nhận đúng thời hạn hay không; nếu không, ảnh chụp nhanh sẽ phải bị từ chối, và Head sẽ phải đóng lại.

Để tránh đóng Head do từ chối ảnh chụp nhanh trong trường hợp của sự kiện này (hiếm khi xảy ra), chúng tôi sửa đổi CR theo hai cách.

Thứ nhất, đối với CR, lãnh đạo ảnh chụp nhanh chỉ bao gồm tập hợp con của các giao dịch T_R (xem Phần B.1) còn lại một khoảng thời gian an toàn trước khi hết thời hạn. Sau đó, một Head chỉ có những người tham gia trung thực sẽ bị đóng lại trong điều kiện mạng cực kỳ tồi tệ.

[12]Lưu ý rằng, mặc dù giao thức Head của chúng tôi không đồng bộ, chúng tôi vẫn có thể dựa vào thời gian được đồng bộ hóa gần như giữa những người tham gia Head (ví dụ: bằng cách quan sát chuỗi chính) vì chúng tôi đã cần đưa ra giả định này để xử lý các giai đoạn tranh chấp một cách an toàn trong khi kết thúc Head.

Thứ hai, chúng ta cần một cơ chế để xử lý các giao dịch hết hạn đã được nhìn thấy, nhưng có thể chưa được xác nhận. Giả sử một bên p đã ký một giao dịch txA nhưng không được xác nhận cho đến sau thời hạn r_{max} của giao dịch; do đó, tại thời điểm này, p vẫn chưa biết liệu giao dịch cuối cùng có được xác nhận hay không vì một số người tham gia có thể đã nhận được txA quá muộn. Nếu tại thời điểm này một giây, giao dịch xung đột txB được xuất bản và p là lãnh đạo ảnh chụp nhanh, nó phải đưa ra lựa chọn mà nó không có đủ thông tin: nó phải từ chối nếu txA được xác nhận (để chắc chắn), nhưng ký txB nếu txA chưa được xác nhận (về tính sống động).

Để tránh những quyết định như vậy, chúng tôi mở rộng cơ chế ảnh chụp nhanh như sau. Lãnh đạo ảnh chụp nhanh đưa vào ảnh chụp nhanh của mình một *bộ thông báo* (Obituary Set) của các giao dịch mà nó coi là đã hết hạn và chỉ được xác nhận một phần. Nếu bất kỳ bên nào đang sở hữu đa chữ ký cho một giao dịch trong bộ thông báo, thì bên đó sẽ khôi phục giao dịch này bằng cách đưa đa chữ ký của mình vào ảnh chụp nhanh mà nó tạo ra trong lượt tiếp theo với tư cách là lãnh đạo ảnh chụp nhanh. Sau một chu kỳ đầy đủ của vòng quay ảnh chụp nhanh, do đó, chúng ta có thể chấp nhận một cách an toàn một giao dịch như đã được xác nhận nếu có đa chữ ký được thêm vào trong chu kỳ đó hoặc từ chối giao dịch đó khi đã hết hạn nếu không có đa chữ ký nào được giao.

D.3. Điều chỉnh giao dịch

Một lãnh đạo kẻ tấn công có thể đình trệ quá trình sản xuất ảnh chụp nhanh trong khi vẫn cho phép các giao dịch mới được xác nhận. Để ngăn chặn việc tấn công như vậy làm tăng lịch sử giao dịch được lưu trữ vượt quá giới hạn, quy mô của tất cả các giao dịch ảnh chụp nhanh chưa được xử lý chỉ có thể tăng đến một giới hạn nhất định. Miễn là đạt đến giới hạn này, không có giao dịch mới nào được xác nhận (hoặc cách khác là Head bị đóng khi đạt đến giới hạn).

Người dịch: Nguyễn Văn Tú

Telegram: https://t.me/Tulibra

Link gốc: https://iohk.io/en/research/library/papers/hydra-fast-isomorphic-state-channels/