텔레그램 오픈 네트워크*

Dr. 니콜라이 듀로브

2018년 1월 18일

톤코리아 옮김 (v1.0)

초록

이 텍스트의 목적은 TON (텔레그램 오픈 네트워크) 및 관련 블록체인, 피어-투-피어, 분산형 저장소 및 서비스 호스팅 기술에 대한 첫 번째 설명을 제공하는 것입니다. 이 문서의 크기를 적당한 비율로 줄이기 위해, 우리는 주로 명시된 목표를 달성하는데 중요한 TON 플랫폼의 독특하고 정의된 기능에 주로 중점을 둡니다.

소개

텔레그램 오픈 네트워크 (*TON*) 은 빠르고 안전하며 확장 가능한 블록체인 및 네트워크 프로젝트로서 필요할 경우 초당 수백만 건의 트랜잭션을 처리할 수 있으며 사용자와 서비스 제공 업체에게 친숙합니다. 우리는 현재 제안되고 고안된 합리적인 모든 애플리케이션을 호스팅 할 수 있도록 노력합니다. TON 은 거대한 분산형 슈퍼컴퓨터, 또는 다양한 서비스를 호스트 하고 제공하기 위한 거대한 "슈퍼서버"로 생각할 수 있습니다.

이 텍스트는 모든 구현 세부사항과 관련하여 궁극적인 참조가 되는 것은 아닙니다. 일부 세부사항은 개발 및 테스트 단계에서 변경될 수 있습니다.

* 이 본문의 버전은 프라이머 부록 A로 배포됩니다.

목차

1 TON 구성요소 설명 (Brief Description of TON Components)

텔레그램 오픈 네트워크 (*TON*) 는 다음 구성 요소들의 조합입니다.

- 유연한 멀티-블록체인 플랫폼 (*TON* 블록체인; cf. **2**장) 튜링-완전 스마트 컨트랙트를 통해 초당 수백만 건의 트랜잭션 처리가 가능하며, 업그레이드 가능한 공식 블록체인 사양, 멀티-암호화폐 값 전송, 소액결제채널 및 오프-체인 결제 네트워크 지원이 가능합니다. *TON* 블록체인은 "셀프-힐링" 수직 블록체인 메커니즘 (cf. **2.1.17**) 및 인스턴트 하이퍼큐브 라우팅 (cf. **2.4.20**)과 같은 새롭고 색다른 기능을 제공하여 TON 블록체인을 빠르고 안정적이며 확장 가능하며 동시에 일관성 있게 만듭니다.
- 피어-투-피어 네트워크 (*TON P2P* 네트워크, 또는 *TON* 네트워크; cf. **3**장 TON 블록체인에 액세스하고, 트랜잭션 후보를 보내고, 클라이언트가 관심이 있는 (예. 클라이언트의 계정 및 스마트 컨트랙트와 관련된) 블록체인의 부분에 대한 업데이트를 수신하는데 사용될 뿐만 아니라 블록체인 관련 임의의 분산형서비스를 지원할 수도 있습니다.
- 분산형 파일 저장기술 (*TON* 저장소; cf. **4.1.8**) *TON* 네트워크를 통해 액세스 할 수 있으며, *TON* 블록체인에서 블록 및 상태 데이터 (스냅샷)의 기록 복사본을 저장하는데 사용되지만 토렌트 같은 액세스 기술을 사용하여 플랫폼에서 실행중인 사용자 또는 기타 서비스에 대한 임의의 파일을 저장할 수도 있습니다.
- 네트워크 프록시/익명 서비스 레이어 (*TON* 프록시; cf. **4.1.11**, **3.1.6**) 필요시 *TON* 네트워크 노드의 신원과 IP 주소를 숨기기 위해 사용되는 I^2P 와 유사합니다 (예. 대량의 암호화폐를 갖는 계정으로부터 트랜잭션을 위탁하는 노드, 또는 DDoS 공격에 대한 대책으로 정확한 IP 주소와 지리적 위치를 숨기고자 하는 높은 지분의 블록체인 밸리데이터 노드).
- 카뎀리아 같은 분산형 해시테이블 (*TON DHT*; cf. **3.2**) *TON* 저장소 (cf. **3.2.10**)를 위한 "토렌트 추적기"와 *TON* 프록시의 "입력 터널 탐지기"로 사용되며 (cf. **3.2.14**) 그리고, *TON* 서비스를 위한 서비스 탐지기 (cf. **3.2.12**)로 사용됩니다.

- 임의 서비스를 위한 플랫폼 (*TON* 서비스, cf. **4**장) *TON* 네트워크 및 *TON* 프록시에 있으며 브라우저 같은 또는 스마트폰 애플리케이션 상호작용을 가능하게 하는 형식화된 인터페이스 (cf. **4.3.14**)가 있는 임의 서비스를 위한 플랫폼 (*TON* 서비스; cf. **4**장). 이러한 형식 인터페이스와 영구 서비스 진입점은 TON 블록체인에 게시할 수 있습니다 (cf. **4.3.17**). 주어진 순간에 서비스를 제공하는 실제 노드는 TON 블록체인에 게시된 정보를 시작으로 *TON DHT*를 통해 조회할 수 있습니다 (cf. **3.2.12**). 서비스는 TON 블록체인에서 클라이언트에게 보증을 제공하기 위해 스마트 컨트랙트를 체결할 수 있습니다 (cf. **4.1.7**).

- *TON DNS* (cf. **4.3.1**) 사람이 읽을 수 있는 이름을 계정, 스마트 컨트랙트, 서비스 및 네트워크 노드에 할당하는 서비스 입니다.

- *TON* 페이먼트 (cf. **5**장) 소액결제, 소액결제채널 및 소액결제채널 네트워크를 위한 플랫폼. 신속한 오프-체인 가치 이전 및 *TON* 서비스가 제공하는 서비스 비용 지불에 사용할 수 있습니다.

- TON은 타사 메시징 및 소셜 네트워킹 애플리케이션과의 손쉬운 통합을 가능하게 하여 블록체인 기술 및 분산형서비스를 그저 초기 암호화폐 수용자에게만이 아닌 마침내 일반 사용자가 이용하고 액세스할 수 있게 합니다 (cf. **4.3.24**). 우리는 이러한 통합의 예를 우리의 다른 프로젝트인 텔레그램 메신저 (cf. **4.3.19**)에서 제공할 것입니다.

TON 블록체인은 TON 프로젝트의 핵심이며 다른 구성 요소는 블록체인에 대한 지원역할을 하는 것으로 간주될 수 있지만 그것들은 스스로 유용하고 흥미로운 기능을 가지고 있습니다. 결합하면, TON 블록체인 (cf. **2.9.13**, **4.1**)만 사용하는 것보다 플랫폼이 더 다양한 애플리케이션을 호스트 할 수 있습니다.

2 TON 블록체인 (TON Blockchain)

먼저 프로젝트의 핵심 구성요소인 텔레그램 오픈 네트워크 (TON) 블록체인에 대한 설명부터 시작합니다. 여기서 우리의 접근 방법은 "하향식"입니다. 먼저 전반적인 설명을 한 다음 각 구성요소들에 대해 자세히 설명합니다.

원칙적으로 이 블록체인 프로토콜의 여러 인스턴스가 독립적으로 실행될 수도 있지만 (예. 하드 포크로 인해) TON 블록체인에 대해 간단히 설명합니다. 우리는 그들 중 하나만을 고려합니다.

2.1 2-블록체인 모음인 TON 블록체인 (TON Blockchain as a Collection of 2-Blockchains)

TON 블록체인은 사실 블록체인 모음입니다 (블록체인의 블록체인 모음 또는 2-블록체인; 이 부분은 **2.1.17**에서 나중에 설명될 것입니다). 왜냐하면 초당 수십 건의 트랜잭션이 아닌 초당 수백만 건의 트랜잭션 처리 목표를 달성할 수 있는 싱글 블록체인 프로젝트가 없기 때문입니다.

2.1.1. 블록체인 유형 목록 (List of blockchain types). 이 블록체인의 모음은 다음과 같습니다.

- 독특한 마스터 블록체인 (또는 줄여서 마스터체인)은 프로토콜 및 매개변수의 현재 값, 밸리데이터 및 지분세트, 현재 활성 워크체인 세트 및 해당 "샤드"와 가장, 중요한 사항인 모든 워크체인과 샤드체인의 최근 블록에 대한 해시세트를 보유하고 있습니다.
- 여러가지 (최대 2^{32} 개) 워킹 블록체인 또는 줄여서 워크체인은 실질적으로 **"work-horses"**이며 가치 전달 및 스마트 트랜잭션을 포함하고 있습니다. 워크체인마다 서로 다른 "규칙" 즉, 다양한 형식의 계정주소, 다양한 형식의 트랜잭션, 스마트 컨트랙트를 위한 다른 가상머신 (VM), 다른 기본 암호화폐 등등이 있을 수 있습니다. 그러나 서로 다른 워크체인 간의 상호작용을 가능하고 비교적 단순하게 만들기 위해서는 특정 기본 상호운용성 기준을 충족해야 합니다. 이 점에서 **TON** 블록체인은 EOS (cf. **2.9.7**) 및 폴카닷 (PolkaDot) (cf. **2.9.8**) 프로젝트와 마찬가지로 이기종 입니다 (cf. **2.8.8**).
- 각 워크체인은 차례대로 최대 2^{60} 개의 샤드 블록체인, 또는 줄여서 샤드체인으로 세분되며 워크체인 자체와 동일한 규칙 및 블록 형식을 갖고 있지만 일부 계정의

한) 비트의 계정주소에 따라 서브세트에 대해서만 책임이 있습니다. 즉, 한 샤딩 형태가 시스템에 내장되어 있습니다 (cf. **2.8.12**). 이러한 모든 샤드체인은 동일한 블록 형식과 규칙을 공유하기 때문에 이 점에서 TON 블록체인은 이더리움 확장 제안 중 하나에서 논의된 것과 마찬가지로 동종 입니다 (cf. **2.8.8**).[1]

- 샤드체인 (및 마스터체인)의 각 블록은 실제로는 그냥 블록이 아니라 작은 블록체인입니다. 일반적으로 이 "블록 블록체인" 또는 "수직 블록체인"은 정확하게 하나의 블록으로 구성되며, 이 블록은 샤드체인 (이 상황에서는 "수평 블록체인"이라고도 함)의 해당 블록이라고 생각할 수 있습니다. 그러나 올바르지 않은 샤드체인 블록을 수정해야 하는 경우 새 블록이 "수직 블록체인"에 위탁되어 유효하지 않은 "수평 블록체인" 블록에 대한 대체 또는 이 블록의 이전 버전에서 변경해야 하는 해당 부분의 설명만 포함한 "블록 차이점"이 포함됩니다. 이것은 모든 샤드체인이 진정한 포크를 만들지 않고 발견된 유효하지 않은 블록을 교체하는 TON 고유의 메커니즘으로 **2.1.17**에서 더 자세히 설명할 것입니다. 지금은 각 샤드체인 (및 마스터체인)을 기존의 블록체인이 아니라 블록체인의 블록체인 또는 2D-블록체인 또는 2-블록체인이라는 점만 주목 하십시오

2.1.2. 무한 샤딩 패러다임 (Infinite Sharding Paradigm). 거의 모든 블록체인 샤딩 제안은 "하향식"입니다: 먼저 싱글 블록체인을 상상한 다음, 몇 가지 상호작용 하는 샤드체인으로 분할하여 성능을 향상시키고 확장성을 달성하는 방법에 대해 설명합니다.

샤딩에 대한 TON 접근법은 "상향식"이며 다음과 같이 설명됩니다. 샤딩이 극단적으로 정확히 하나의 계정 또는 스마트 컨트랙트가 각 샤드체인에 남아 있다고 상상해보십시오. 그러면 우리는 단 하나의 계정에 대한 상태 및 상태전이를 설명하는 가치 및 정보를 전달하기 위해 가치가 있는 메시지를 서로에게 보내는 아주 많은 "어카운트 체인"이 있습니다.

물론, 수 억개의 블록체인을 갖는 것은 실용적이지 않으며 업데이트 (즉, 새로운 블록)는 대개 각 블록에 거의 나타나지 않습니다. 더 효율적으로 이행하기 위해서는 이러한 "어카운트 체인"을 "샤드체인"으로 그룹화하여 샤드체인의 각 블록이 근본적으로 이 샤드에 할당된 어카운트 블록체인의 모음이 됩니다.

[1] https://github.com/ethereum/wiki/wiki/Sharding-FAQ

따라서 "어카운트 체인"은 "샤드체인" 내부에만 순수가상 또는 논리적 존재를 갖습니다.

우리는 이 관점을 무한샤딩 패러다임이라고 부릅니다. 이 관점은 TON 블록체인에 대한 많은 설계 결정의 이유가 됩니다.

2.1.3. 메시지. 인스턴트 하이퍼큐브 라우팅 (Messages. Instant Hypercube Routing). 무한 샤딩 패러다임은 각 계정 (또는 스마트 컨트랙트)을 샤드체인 자체에 있는 것처럼 간주합니다. 그런 다음 한 계정이 다른 계정의 상태에 영향을 줄 수 있는 유일한 방법은 메시지를 보내는 것입니다 (이것은 "액터"라는 계정이 있는 소위 액터 모델의 특수한 경우입니다 (cf. **2.4.2**); 따라서, TON 블록체인과 같은 확장 가능한 시스템에서는 계정 (및 샤드체인; 소스 계정과 대상 계정이 일반적으로 서로 다른 샤드체인에 위치하기 때문에) 간의 메시지 시스템이 매우 중요합니다. 실제로, 인스턴트 하이퍼큐브 라우팅 (cf. **2.4.20**)이라고 불리는 TON 블록체인의 참신한 기능을 사용하면 시스템의 전체 샤드체인 수에 관계없이 하나의 샤드체인 블록에서 생성된 메시지를 대상 샤드체인의 다음 블록으로 전달하고 처리할 수 있습니다.

2.1.4. 마스터체인, 워크체인 및 샤드체인의 수량 (Quantity of masterchains, workchains and shardchains). TON 블록체인에는 정확히 하나의 마스터체인이 있습니다. 그러나, 이 시스템은 최대 2^{32} 개의 워크체인을 수용할 수 있으며 각 워크체인은 최대 2^{60} 개의 샤드체인으로 세분됩니다.

2.1.5. 진정한 블록체인이 아닌 가상 블록체인이 될 워크체인 (Workchains can be virtual blockchains, not true blockchains). 워크체인은 일반적으로 샤드체인으로 세분되기 때문에 워크체인의 존재는 "가상"입니다. 즉, 아래 **2.2.1** 에서 제공된 일반적인 정의로는 진정한 블록체인이 아니라 샤드체인의 모음일 뿐입니다. 하나의 샤드체인만 워크체인에 해당하는 경우, 이 독특한 샤드체인은 워크체인으로 식별될 수 있습니다. 이 경우 워크체인은 적어도 어느 기간 동안은 "참" 블록체인이 되어 싱글-블록체인 디자인의 피상적 유사성을 갖습니다. 그러나 무한 샤딩 패러다임 (cf. **2.1.2**)은 이 유사성이 실제로 피상적이라고 말합니다: 잠재적으로 엄청난 수의 "어카운트 체인"이 일시적으로 하나의 블록체인으로 그룹화 될 수 있다는 것은 단지 우연한 일치일 뿐입니다.

2.1.6. 워크체인의 식별 (Identification of workchains). 각 워크체인은 숫자 또는 워크체인 식별자 *(workchain_id : uint32)*로 식별됩니다.

이 식별자는 단순히 무부호 32-비트 정수입니다. 워크체인은 마스터체인의 특수 트랜잭션에 의해 생성되며 (이전에 사용되지 않은) 워크체인 식별자와 워크체인의 공식적인 설명을 정의합니다. 이것은 적어도 이 워크체인과 다른 워크체인 간의 상호작용 및 이 워크체인 블록의 피상적 검증에 충분합니다.

2.1.7. 새 워크체인의 구축과 활성화 (Creation and activation of new workchains). 새로운 워크체인을 구축하기 위해서는 새로운 워크체인의 공식사양을 게재하는데 필요한, 마스터체인 트랜잭션 (높은) 수수료를 지불할 준비가 되어있는 커뮤니티의 모든 구성원에 의해 개시될 수 있습니다. 그러나 새 워크체인이 활성화 되려면 밸리데이터의 3분의 2의 합의가 필요합니다. 새 워크체인 블록을 처리하기 위해 소프트웨어를 업그레이드 해야 하고 특수 마스터체인 트랜잭션을 사용하여 새 워크체인과 작업할 준비가 되었음을 알려야 하기 때문입니다. 새로운 워크체인 활성화에 관심이 있는 당사자는 밸리데이터에게 새 워크체인을 지원하는 대가로 스마트 컨트랙트로 배포한 일부 보상을 제공할 수 있습니다.

2.1.8. 샤드체인의 식별 (Identification of shardchains). 각 샤드체인은 한 쌍의 $(w, s) = (workchain\_id\ ,\ shard\_prefix)$ 로 식별됩니다. 여기서 $workchain\_id\ :\ uint_{32}$ 는 해당 워크체인을 식별하고, $shard\_prefix : \mathbf{2}^{0\ ..\ 60}$ 은 길이가 60-비트 이하인 비트 문자열로, 이 샤드체인이 책임지는 어카운트의 서브세트를 정의합니다. 즉, *shard_prefix* 로 시작하는 를 가진 모든 어카운트가 (예. *shard_prefix* 를 최상위 비트로 가짐) 이 샤드체인에 할당됩니다.

2.1.9. 어카운트 체인의 식별 (Identification of account-chains). 어카운트 체인은 단지 가상 존재만 가지고 있음을 상기하십시오 (cf. **2.1.2**). 그러나 어카운트 체인은 자연스런 식별자 즉, $(workchain\_id\ ,\ account\_id)$ 가 있습니다. 왜냐하면 어떤 어카운트 체인에는 정확히 하나의 어카운트 (간단한 어카운트 또는 스마트 컨트랙트 - 여기서는 구별이 중요하지 않음)에 대한 상태와 업데이트에 관한 정보가 포함되어 있기 때문입니다.

2.1.10. 샤드체인의 동적분할 및 병합; cf. 2.7 (Dynamic splitting and merging shardchains; cf. 2.7). 덜 정교한 시스템에서는 정적샤딩을 사용할 수 있습니다. 예를 들어, 의 상위 8-비트를 사용하여 256개의 사전정의된 샤드 중 하나를 선택합니다.

TON 블록체인의 중요한 특징은 동적샤딩을 구현한다는 것입니다. 즉, 샤드의 수가 고정되어 있지 않다는 것을 의미합니다. 대신, 일부 공식조건이 충족되면 (근본적으로 오리지널 샤드의 트랜잭션 부하가 장기간 동안 충분히 높으면) 샤드 (w, s) 를 샤드 $(w, s.0)$ 와 $(w, s.1)$ 로 자동으로 세분할 수 있습니다.

반대로 부하가 일정기간 동안 너무 낮으면 샤드 $(w, s.0)$ 와 $(w, s.1)$ 를 샤드 (w, s) 로 자동 병합할 수 있습니다.

처음에는 워크체인 w 에 대해 단 하나의 샤드 $(w, \varnothing)$만 생성됩니다. 만약 혹은 후에 필요하다면 더 많은 샤드로 세분됩니다 (cf. **2.7.6**, cf. **2.7.8**).

2.1.11. 기본 워크체인 또는 워크체인 (Basic workchains or Workchain Zero). 특정 규칙 및 트랜잭션을 통해서 최대 2^{32} 개의 워크체인을 정의할 수 있지만 초기에는 *workchain_id* = 0 을 사용하여 단 하나만 정의합니다. 워크체인 제로 또는 기본 워크체인이라고 불리는 이 워크체인은 *TON* 스마트 컨트랙트를 사용하여 *TON* 코인, 또는 그램 (*Grams*)을 전송하는데 사용됩니다 (cf. 부록 A). 대부분의 애플리케이션에는 워크체인 제로만 필요합니다. 기본 워크체인의 샤드체인을 기본 샤드체인이라고 칭합니다.

2.1.12. 블록 생성간격 (Block generation intervals). 각 샤드체인과 마스터체인에서 약 5초마다 한 번씩 새 블록을 생성할 것으로 예상됩니다. 이는 합리적으로 작은 트랜잭션 검증시간으로 이어질 것입니다. 모든 샤드체인의 새 블록은 거의 동시에 생성됩니다. 마스터체인의 새 블록은 모든 샤드체인의 최신 블록의 해시를 포함해야 하므로 약 1초 후에 생성됩니다

2.1.13. 마스터체인을 사용하여 워크체인과 샤드체인을 단단히 결합 (Using the masterchain to make workchains and shardchains tightly coupled). 샤드체인 블록의 해시가 마스터체인 블록에 통합되면 해당 샤드체인 블록과 모든 조상은 모든 정식 체인의 후속 블록에서 고정 및 불변으로 참고될 수 있음을 의미하는 "정식”으로 간주됩니다. 사실상 새로운 샤드체인 블록마다 가장 최근의 마스터체인 블록의 해시가 포함되어 있으며 해당 마스터체인 블록에서 참고되는 모든 샤드체인 블록은 새 블록에 의해 불변으로 간주됩니다.

본질적으로, 이것은 샤드체인 블록에 커밋된 트랜잭션이나 메시지가 EOS와 같이 일반적으로 제안된 “느슨하게-결합된” 시스템 (cf. **2.8.14**)처럼 이전 트랜잭션을 기반으로 메세지를 전달하거나 다른 작업을 수행하기 전에 예를 들어, 20개의 승인 (동일한 블록체인에서 원래 블록 다음에 생성된 20개의 블록)을 기다릴 필요없이 다른 샤드체인의 바로 다음 블록에서 안전하게 사용될 수 있음을 의미합니다. 다른 샤드체인에 있는 트랜잭션 및 메시지를 커밋된 후 불과 5초 후에 사용할 수 있는 능력은 우리가 "단단하게-결합된" 시스템이 전례없는 성능을 제공할 수 있다고 믿는 이유 중 하나입니다 (cf. **2.8.12**, **2.8.14**).

2.1.14. 글로벌 상태의 마스터체인 블록 해시 (Masterchain block hash as a global state). 2.1.13에 따르면 마지막 마스터체인 블록의 해시는 외부 관찰자의 관점에서 시스템의 전반적인 상태를 완전히 결정합니다. 모든 샤드체인 갈래를 별도로 모니터링 할 필요가 없습니다.

2.1.15. 밸리데이터에 의한 새 블록의 생성; cf. 2.6 (Generation of new blocks by validators; cf. 2.6). TON 블록체인은 샤드체인 및 마스터체인에서 새 블록을 생성하기 위해 지분증명 접근 방식을 사용합니다. 즉, 새로운 블록 생성 및 유효성 검증을 받기 위해 스페셜 마스터체인 트랜잭션으로 지분 (막대한 양의 TON 코인)을 입금한 예를 들어, 최대 수백 개의 밸리데이터 - 스페셜 노드가 있습니다.

그런 다음 밸리데이터의 더 작은 서브세트는 결정론적 의사난수 방식으로 각 샤드 (w, s) 에 할당되어 약 1024개의 블록이 변경됩니다. 이 밸리데이터의 서브세트는 제안된 적절한 트랜잭션을 클라이언트에서 새로운 유효한 블록후보로 수집하여 다음 샤드체인 블록이 무엇인지에 대해 의견을 제시하고 합의에 도달합니다. 각 블록에 대해 밸리데이터는 의사난수 순서로 선택되어 어떤 블록후보가 매 순서마다 커밋될 우선순위가 가장 높은지를 결정합니다.

밸리데이터 및 다른 노드는 제안된 블록후보의 유효성을 검사합니다. 밸리데이터가 유효하지 않은 블록후보에 서명할 경우, 자동으로 해당지분의 일부 또는 전부를 잃거나 밸리데이터 집합으로부터 잠시 중지당하여 처벌될 수 있습니다. 그 후에 밸리데이터는 PBFT [7] 또는 허니뱃저 BFT (Honey Badger BFT) [18]와 유사한 BFT (비잔틴 장애 허용(Byzantine Fault Tolerant); cf. **2.8.4**) 합의 프로토콜의 효율적인 변형을 통해 다음 블록의 선택에 대한 합의에 도달해야 합니다.[2] 만약 합의가 이루어지고 새로운 블록이 생성되면 밸리데이터들은 포함된 트랜잭션에 대한 트랜잭션 수수료와 새로 "생성된" 코인을 서로 나눠 갖습니다.

각 밸리데이터는 여러 밸리데이터 서브세트에 참여하도록 선출될 수 있습니다. 이 경우, 모든 유효성 검증 및 합의 알고리즘을 병렬로 실행할 것으로 예상됩니다.

모든 새로운 샤드체인 블록은 생성되거나 시간초과가 된 후, 모든 샤드체인의 최신 블록 해시를 포함한 새로운 마스터체인 블록이 생성됩니다. 이것은 모든 밸리데이터의 BFT 합의에 의해 수행됩니다.[3]

TON PoS 접근법과 이의 경제적 모델에 대한 자세한 내용은 **2.6** 에서 제공됩니다.

---

[2] 해시그래프 [1]와 유사한 가십 기반 비잔틴 합의 프로토콜도 이 목적과 관련이 있을 수도 있습니다.
[3] 사실 지분의 3 분의 2가 합의를 달성하기에 충분하지만 최대한 많은 서명을 수집하기 위한 노력이 이루어집니다.

2.1.16. 마스터체인의 포크 (Forks of the masterchain). 단단하게-결합된 접근법에서 발생하는 복잡성은 마스터체인에서 다른 포크로 전환할 경우 거의 반드시 일부 샤드체인에서 또한 다른 포크로 전환해야 한다는 것입니다. 다른 한편, 마스터체인에 포크가 없는 한, 샤드체인의 대체 포크는 마스터체인 블록에 통합되어 "정식"이 될 수 없으므로 샤드체인의 포크도 가능하지 않습니다.

일반적인 규칙은 마스터체인 블록 B' 가 B 의 전임인 경우, B' 는 (w, s)의 해시 $HASH(B'_{w,s})$를 포함하고 B 는 해시 $HASH(B_{w,s})$ 를 포함하고 $B''_{w,s}$ 는 반드시 $B_{w,s}$ 의 전임자여야 합니다; 그렇지 않으면 마스터체인 블록 B 는 유효하지 않습니다.

마스터체인 포크는 존재하지 않는것 다음으로 희귀할 것입니다. 왜냐하면 TON 블록체인이 채택한 BFT 패러다임에서는 대부분의 밸리데이터 (cf. **2.6.1**, **2.6.15**)가 잘못된 행동을 했을 경우에만 발생할 수 있으며 이는 위반자의 엄청난 지분손실을 의미하기 때문입니다. 따라서, 샤드체인에 있는 어떠한 "참" 포크도 기대되어서는 안됩니다. 대신 유효하지 않은 샤드체인 블록이 발견되면 2-블록체인의 "수직 블록체인" 메커니즘을 사용하여 수정되어 (cf. **2.1.17**) "수평 블록체인" (즉, 샤드체인)을 포킹하지 않고 이 목표를 달성할 수 있습니다. 동일한 메커니즘을 사용하여 마스터체인 블록에 있는 치명적이지 않은 실수도 수정할 수 있습니다.

2.1.17. 무효 샤드체인 블록 수정 (Correcting invalid shardchain blocks). 일반적으로 샤드체인에 할당된 밸리데이터는 새로운 블록을 커밋하기 전에 3 분의 2의 비잔틴 합의에 도달해야하기 때문에 유효한 샤드체인 블록만 커밋됩니다.
그러나 시스템은 이전에 커밋된 유효하지 않은 블록을 찾아내고 수정하는 것을 반드시 허용해야 합니다.

물론 유효하지 않은 샤드체인 블록이 - 밸리데이터 (꼭 이 샤드체인에 할당되지 않아도 되는) 또는 "어부" (블록 유효성에 대한 질문을 제기할 수 있도록 특정 보증금을 만든 시스템의 모든 노드; cf. **2.6.4**)에 의해 - 발견되면 무효주장 및 그 증명은 마스터체인에 커밋되며 유효하지 않은 블록에 서명한 밸리데이터는 지분의 일부를 잃어버리거나 및/또는 밸리데이터 세트에서 일시적으로 정지됨으로써 처벌당합니다 (공격자가 양성 밸리데이터의 개인 서명 키를 훔치는 경우 때문에 후자의 조치는 중요합니다).

그러나 이것이 충분하지 않은 이유는 시스템 (TON 블록체인)의 전체 상태가 이전에 커밋된 유효하지 않은 샤드체인 블록으로 인해 유효하지 않은 것으로 판명되기 때문입니다.

이 유효하지 않은 블록은 보다 최신의 유효한 버전으로 대체되어야 합니다.

대부분의 시스템은 이 샤드체인의 유효하지 않은 블록 이전의 마지막 블록과 다른 블록체인의 유효하지 않은 블록으로부터 전달된 메시지의 영향을 받지 않는 마지막 블록으로 "롤링 백 (rolling back)"하여 이 블록으로부터 새로운 포크를 생성합니다. 이 접근법의 단점은 다르게 정정되고 커밋된 많은 수의 트랜잭션이 갑자기 복귀된다는 것이며 그들이 나중에 포함될 것인지 여부가 불분명합니다.

TON 블록체인은 각 샤드체인과 마스터체인 ("수평 블록체인")의 각 "블록"을 이 "블록"의 다른 버전 또는 "차이점"을 포함하는 작은 블록체인 ("수직 블록체인")으로 만듦으로써 이 문제를 해결합니다. 일반적으로 수직 블록체인은 정확히 하나의 블록으로 구성되며 샤드체인은 고전적인 블록체인처럼 보입니다. 그러나 블록의 무효가 확인되고 마스터체인 블록에 커밋되면 유효하지 않은 블록의 "수직 블록체인"은 새로운 블록에 의해 수직방향으로 증가하여 잘못된 블록을 교체하거나 편집할 수 있습니다. 문제의 샤드체인을 대체하는 새 블록은 현재의 밸리데이터 서브세트에 의해 생성됩니다.
새로운 "수직" 블록의 유효규칙은 매우 엄격합니다. 특히, 유효하지 않은 블록에 포함된 가상 "어카운트 체인 블록" (cf. **2.1.2**)이 그 자체로 유효할 경우, 새로운 수직 블록에 의해 변경되지 않은 채로 남아 있어야 합니다.

새로운 "수직" 블록이 유효하지 않은 블록 위에 커밋되면 이의 해시는 새로운 마스터체인 블록에 게시되며 (또는 오히려 새로운 "수직" 블록에 유효하지 않은 샤드체인 블록의 해시가 당초에 게시되었던 원래의 마스터체인 블록 위에 있으며), 변경사항은 이 블록의 이전 버전을 참고하는 모든 샤드체인 블록 (예. 잘못된 블록에서 메시지를 수신한 블록)으로 더 전파됩니다. 이것은 이전에 "잘못된" 블록을 참고하는 모든 블록에 대해 수직 블록체인에 새로운 "수직" 블록을 커밋하여 해결됩니다.

새로운 수직블록은 가장 최근의 (수정된) 버전을 대신 참조할 것입니다. 엄격한 규칙은 실제로 영향을 받지 않는 (즉, 이전 버전과 동일한 메시지를 받는) 어카운트 체인을 변경하는 것을 금지합니다. 이렇게하면 잘못된 블록을 수정하면 영향을 받는 모든 샤드체인의 가장 최근 블록으로 궁극적으로 전파되는 "잔물결"이 생성됩니다. 이러한 변경 사항은 새로운 "수직" 마스터체인 블록에도 반영됩니다.

"이력 재작성" 잔물결이 가장 최근의 블록에 도달하면 새로운 샤드체인 블록이 하나의 버전에서만 생성되고 최신 블록 버전의 후속 버전이 됩니다. 즉, 처음부터 정확한

(최근) 수직블록에 대한 참조를 포함하게 됩니다.

마스터체인 상태는 각 "수직" 블록체인의 첫 번째 블록의 해시를 최신 버전의 해시로 변환하는 맵을 암시적으로 정의합니다. 이를 통해 클라이언트는 맨 처음 블록(그리고 일반적으로 유일한 블록)의 해시로 모든 수직 블록체인을 식별하고 찾을 수 있습니다.

2.1.18. TON 코인 및 다중-통화 워크체인 (TON coins and multi-currency workchains). TON블록체인은 32-비트 *currency_id*로 구별되는 최대 2^{32} 개의 "암호화폐", "코인" 또는 "토큰"을 지원합니다.

마스터체인에서 특별한 트랜잭션을 통해 새로운 암호화폐를 추가할 수 있습니다. 각 워크체인에는 기본 암호화폐가 있으며 몇 가지 추가 암호화폐가 있을 수 있습니다. *currency_id* = 0, 즉 그램으로 알려진 특별한 *TON* 코인 (cf. 부록 **A**)이 하나 있습니다. 그것은 워크체인 제로의 기본적인 암호화폐입니다. 이것은 트랜잭션 수수료 및 밸리데이터 지분에도 사용됩니다.

원칙적으로 다른 워크체인은 다른 토큰에서 트랜잭션 수수료를 징수할 수 있습니다. 이 경우, 이러한 거래 수수료를 그램으로 자동 변환하기 위해 일부 스마트 컨트랙트를 제공해야 합니다.

2.1.19. 메시징 및 가치이전 (Messaging and value transfer). 같거나 다른 워크체인에 속한 샤드체인은 서로 메시지를 보낼 수 있습니다. 허용되는 메시지의 정확한 형태는 수신 워크체인 및 수신계정 (스마트 컨트랙트)에 따라 다르지만 워크체인 간에 메시징을 가능하게 하는 몇 가지 공통 필드가 있습니다. 특히 각 메시지는 수신 워크체인이 허용하는 암호화폐일 경우 그램 (TON 코인) 및/또는 기타 등록된 암호화폐의 일정 금액으로 첨부된 값을 가질 수 있습니다.

이러한 메시징의 가장 간단한 형태는 하나의 계정 (일반적으로 스마트 컨트랙트가 아님)에서 다른 계정으로의 가치이전 입니다.

2.1.20. TON 가상머신 (TON Virtual Machine). *TON VM* 또는 *TVM*의 약자인 *TON* 가상머신은 마스터체인 및 기본 워크체인에서 스마트 컨트랙트 코드를 실행하는데 사용되는 가상머신 입니다. 다른 워크체인은 TVM과 함께 또는 대신 다른 가상머신을 사용할 수 있습니다. 여기에 그 기능 중 일부가 나열되어 있습니다. **2.3.12** , **2.3.14** 및 다른 곳에서 더 논의됩니다.

- TVM은 모든 데이터를 (*TVM*) 셀의 모음으로 나타냅니다 (cf. **2.3.14**). 각 셀에는 최대 **128** 데이터 바이트 및 다른 셀에 대한 최대 4개의 참조가 포함됩니다.

"모든것은 셀백이다" 철학 (cf. **2.5.14**)의 결과로, 필요하다면 TVM은 블록 및 블록체인 글로벌 상태를 포함하여 TON은 관련된 모든 데이터와 작업할 수 있습니다.

- TVM은 트리로 표현된 임의의 대수데이터 유형의 값 (cf. **2.3.12**) 또는 TVM 셀의 지시된 비순환적 그래프와 작업할 수 있습니다. 그러나 대수 데이터 유형의 존재에 대해 불가지론 적입니다; 그것은 단지 셀과 함께 작동합니다.
- TVM에는 해시맵에 대한 지원기능이 내장되어 있습니다. (cf. **2.3.7**)
- TVM은 스택머신 입니다. 스택은 64-비트 정수 또는 셀 참조를 보관합니다.
- 64-비트, 128-비트 및 256-비트 산술이 지원됩니다. 모든 n-비트 산술연산은 무부호 정수, 부호있는 정수 및 모듈로 2^n (후자의 경우 자동 오버플로우 검사 없음)의 세 가지 유형으로 나뉩니다.
- TVM은 n-비트에서 m-비트까지 무부호와 부호있는 정수 변환을 오버플로우 검사와 함께 모두 $0 \leq m, n \leq 256$에 대해 수행합니다.
- 모든 산술연산은 기본적으로 오버플로우 검사를 수행하므로 스마트 컨트랙트 개발이 매우 단순해집니다.
- TVM은 더 큰 정수유형에서 계산된 중간값을 사용하여 "곱셈-후-시프트" 및 "시프트-후-나누기" 산술연산을 가집니다. 이것은 고정소수점 산술을 단순화합니다.
- TVM은 비트 문자열과 바이트 문자열을 지원합니다.
- Curve25519 및 Ed25519를 포함한 일부 사전정의된 곡선에 대한 256-비트 엘립틱 커브 크립토그래피 (Elliptic Curve Cryptography, ECC) 지원이 제공됩니다.
- 페어링 기반 암호화 및 zk-SNARKs[9]의 빠른 검증에 유용한 일부 타원곡선에서 웨일(Weil), 테이트(Tate) 또는 관련 페어링을 지원합니다.[4]
- SHA256을 포함한 널리 사용되는 해시함수가 지원됩니다.

[4] 보다 구체적으로, 이러한 지원에는 사전 정의된 매개변수값 집합에 대한 바레토-나에리크 곡선 (Barreto-Naehrig curve) [12] 에 대한 Ate 페어링의 계산 [2] 이 포함됩니다.

- TVM은 머클증명과 함께 작동할 수 있습니다. (cf. **5.1.9**)
- TVM은 "대규모" 또는 "글로벌" 스마트 컨트랙트를 지원합니다. 그러한 스마트 컨트랙트는 샤딩을 인지해야 합니다 (cf. **2.3.18**, **2.3.16**). 일반적인 로컬 스마트 컨트랙트는 샤딩에 무관심 할 수 있습니다.
- TVM은 종결을 지원합니다.
- "스파인리스 태그리스 *G*-머신 (spineless tagless G-Machine)" [20] 은 TVM 내부에서 쉽게 구현될 수 있습니다.

"TVM 어셈블리" 외에도 TVM을 위해 몇 가지 고급 언어를 설계할 수 있습니다. 이 모든 언어는 정적 유형을 가지며 대수 데이터 유형을 지원합니다. 우리는 다음과 같은 가능성을 염두에 둡니다.

- 자바 같은 명령형언어. 스마트 컨트랙트는 각각 다른 분류와 유사합니다.
- 게으른(lazy) 함수언어 (하스켈에 대해 생각해 보십시오).
- 열망하는(eager) 기능언어 (ML에 대해 생각해 보십시오).

2.1.21. 구성 가능한 매개변수 (Configurable parameters). TON 블록체인의 중요한 특징은 많은 매개변수가 구성 가능하다는 것입니다. 즉, 마스터체인 상태에 속하며 하드포크가 필요없이 마스터체인에서 특정 제안/투표/결과 트랜잭션을 통해 변경될 수 있습니다. 그러한 매개변수를 변경하려면 밸리데이터의 3 분의 2의 표를 받고 또한 그 외에 제안서를 지지하고 투표과정에 참여하고 싶은 모든 참여자의 투표 수의 절반 이상을 받아야 합니다.

2.2 블록체인의 보편성 (Generalities on Blockchains)

2.2.1. 일반 블록체인 정의 (General blockchain definition). 일반적으로 모든 (참) 블록체인은 블록의 배열입니다. 이전 블록 (보통 이전 블록의 해시를 현재 블록의 헤더에 포함 시킴)에 대한 참고 BLK-PREV(B) 를 포함하는 각 블록 B 의및 트랜잭션 목록입니다. 각 트랜잭션은 글로벌 블록체인 상태의 일부 변환을 설명합니다. 블록에 나열된 트랜잭션은 순차적으로 적용되어 이전 상태에서 시작한 새로운 상태를 계산합니다. 여기서 새로운 상태는 이전 블록의 평가 후 결과 상태입니다.

2.2.2. TON 블록체인과의 관련성 (Relevance for the TON Blockchain). TON 블록체인은 참 블록체인이 아니라 2-블록체인 모음 (즉, 블록체인의 블록체인; cf. **2.1.1**) 이므로 상기 내용을 직접 적용할 수 없습니다. 그러나, 우리는 블록체인을 보다 복잡한 구조를 위한 빌딩 블록으로 사용하기 위해 참 블록체인에 대한 이러한 일반성으로 시작합니다.

2.2.3. 블록체인 인스턴스 및 블록체인 유형 (Blockchain instance and blockchain type). 일반적으로 블록체인이라는 단어를 사용하여 특정 조건을 만족하는 블록의 배열로 정의된 일반 블록체인 유형과 특정 블록체인 인스턴스를 나타냅니다. 예를 들어, **2.2.1**은 블록체인 인스턴스를 나타냅니다.

이러한 식으로, 블록체인 유형은 일반적으로 특정 호환성 및 유효성 조건을 만족시키는 블록의 배열로 구성된 블록의 목록 (즉, 유한 배열)의 *Block** 유형의 "서브유형" 입니다.

$$\text{Blockchain} \subset \text{Block}^* \tag{1}$$

블록체인을 정의하는 더 좋은 방법은 블록체인이 여러개의 $(\mathbb{B}, v)$와 첫째 구성 요소 $\mathbb{B}$: *Block** 유형 (즉, 블록 목록)의 *Block** 으로 구성된 의존형 커플 유형이라고 말할 수 있습니다. 둘째 구성요소 $\mathbb{B}$의 유효성을 증명하거나 증인인 $v : isValidBc(\mathbb{B})$. 이런식으로,

$$\text{Blockchain} \equiv \Sigma_{(\mathbb{B}:\text{Block}*)}\, \text{isValidBc}(\mathbb{B}) \tag{2}$$

여기서 우리는 [23]에서 빌린 유형의 의존형 합계에 대한 표기법을 사용합니다.

2.2.4. 의존형이론, Coq 및 TL (Dependent type theory, Coq and TL). 여기서 우리는 (마틴-로프, Martin-Löf) 의존형이론을 Coq[5] 증명 보조자에서 사용된 것과 유사하게 사용하고 있습니다. 의존형이론의 단순화된 버전은 TON 블록체인의 공식사양에 사용되어 모든 데이터 구조의 직렬화 및 블록, 트랜잭션 등의 레이아웃을 설명하는*TL* (유형언어)[6] 에서도 사용됩니다.

실제로 의존형이론은 증명이 무엇인지에 대한 유용한 형식화를 제공하며, 예를 들어, 일부 블록에 대해 유효성을 증명할 필요가 있을 때 이러한 공식증명 (또는 그 일련번호)이 유용할 수 있습니다.

2.2.5. TL 또는 유형언어 (TL, or the Type Language). TL (유형언어)은 TON 블록, 트랜잭션 및 네트워크 데이터그램의 공식사양에 사용되므로 간략한 설명이 필요합니다.

[5] https://coq.inria.fr
[6] https://core.telegram.org/mtproto/TL

TL은 숫자 (자연) 및 유형 매개변수를 가질 수 있는 종속대수 유형의 설명에 적합한 언어입니다. 각 유형은 여러 생성자를 사용하여 설명됩니다. 각 생성자는 (사람이 읽을 수 있는) 식별자와 비트 문자열 (기본적으로 32-비트 정수) 이름을 가지고 있습니다. 그 외에도 생성자의 정의에는 해당 유형과 함께 필드목록이 포함됩니다. 생성자와 타입 정의의 모음을 *TL*-계획이라고 합니다. 대개 .tl 접미사가 붙은 하나 또는 여러 개의 파일로 유지됩니다.

TL-계획의 중요한 특징은 정의된 대수유형의 값 (또는 객체)을 직렬화 및 비직렬화하는 확실한 방법을 결정한다는 것입니다. 즉, 값을 바이트 스트림으로 직렬화해야 하는 경우 먼저 이 값에 사용되는 생성자의 이름이 직렬화됩니다. 각 필드의 재귀적으로 계산된 직렬화가 이어집니다.

임의의 객체를 32-비트 정수로 직렬화 하는데 적합한 TL의 이전 버전에 대한 설명은 https:// core.telegram.org/mtproto/TL 에서 볼 수 있습니다. *TL-B*라 불리는 새로운 버전의 TL이 TON 프로젝트에서 사용되는 객체의 직렬화를 설명하기 위해 개발되고 있습니다. 이 새로운 버전은 객체를 바이트 및 심지어 비트 (32-비트 정수가 아닌)의 스트림으로 직렬화 할 수 있으며 TVM 셀의 트리로의 직렬화를 지원합니다 (cf. **2.3.14**). TL-B에 대한 설명은 TON 블록체인의 공식사양의 일부입니다.

2.2.6. 블록과 트랜잭션을 상태변환 연산자로 사용

일반적으로 모든 블록체인 (유형) *Blockchain* 에는 연관된 글로벌 상태 (유형) *State* 및 트랜잭션 (유형) *Transaction* 이 있습니다. 블록체인의 의미는 크게 트랜잭션 적용 함수에 의해 결정됩니다.

$$ev\_trans' : Transaction \times State \to State^{?} \tag{3}$$

여기 $X^{?}$ 는 MAYBE X 를 나타내며 이는 MAYBE monad 를 X 유형에 적용한 결과입니다. 이는 LIST X 에서 사용한 X^{*} 와 유사합니다. 본질적으로, 유형 $X^{?}$ 의 값은 유형 X 의 값이거나 실제 값 (널 포인터를 생각해 보면)이 없다는 것을 나타내는 특수한 값인 ⊥ 입니다. 우리의 경우에는 *State* 대신 $State^{?}$ 를 결과 유형으로 사용합니다. 특정의 원래 상태에서 호출될 경우 트랜잭션이 유효하지 않을 수 있기 때문입니다 (계정에서 실제 돈보다 더 많은 금액을 인출하려고 시도하는 것에 대해 생각해 보십시오).

우리는 *ev_trans'*의 커링 (curried) 버전을 선호할 수 있습니다:

$$ev\_trans : Transaction \to State \to State^{?} \tag{4}$$

기본적으로 블록은 트랜잭션 목록이므로 블록 평가함수

$$ev\_block : Block \to State \to State^{?} \tag{5}$$

는 *ev_trans* 에서 파생될 수 있습니다. 블록 B : *Block* 과 이전 블록체인 상태 S:*State* (이전 블록의 해시를 포함할 수 있음)를 가져와서 다음 블록체인 상태 $s' = ev\_block(B)(s)$: *State*를 계산합니다. 이는 다음 상태를 계산할 수 없음을 나타내는 참 상태 또는 특수 값 ⊥ 중 하나입니다 (즉, 주어진 시작 상태에서 평가된 경우 블록이 유효하지 않음을 나타냅니다. 예를 들어, 블록에는 빈 계정을 인출하려는 트랜잭션이 포함되어 있음).

2.2.7. 블록 배열번호 (Block sequence numbers). 블록체인의 각 블록 B 는 배열번호 BLK-SEQNO(B)로 지칭할 수 있습니다. 가장 첫번째 블록의 경우 0 부터 시작하여 다음 블록으로 넘어갈 때마다 하나씩 증가합니다. 보다 공식적으로,

$$\text{BLK-SEQNO}(B)) = \text{BLK-SEQNO}(\text{BLK-PREV}(B)) + 1 \tag{6}$$

배열번호는 포크가 있는 경우 블록을 고유하게 식별하지 못합니다.

2.2.8. 블록해시 (Block hashes). 블록 B 를 지칭하는 또 다른 방법은 실제로 블록 B 의 헤더의 해시인 해시 BLK-HASH(B) 입니다 (단, 블록 헤더에는 일반적으로 블록 B 의 모든 내용에 의존하는 해시가 포함됩니다). 사용된 해시함수에 대한 충돌이 없다고 가정하면 (또는 적어도 그 가능성은 거의 없음) 블록은 해시에 의해 고유하게 식별됩니다.

2.2.9. 해시 가정 (Hash assumption). 블록체인 알고리즘의 공식적인 분석동안, 사용된 k-비트 해시 함수 HASH : $Bytes^* \to \boldsymbol{2}^k$ 에 대해 충돌이 없다고 가정합니다.

$$\text{Hash}(s) = \text{Hash}(s') \Rightarrow s = s' \text{ for any } s, s' \in \text{Bytes}^* \tag{7}$$

여기서 $Bytes = \{0...255\} = \boldsymbol{2}^8$ 은 바이트 유형 또는 모든 바이트 값 세트이고, $Bytes^\star$ 는 바이트 목록의 임의의 (유한) 유형 또는 세트입니다. $\boldsymbol{2} = \{0, 1\}$은 비트 유형인 반면, $\boldsymbol{2}^k$ 는 모든 k-비트 배열 (즉, k-비트 수)의 세트 (또는 실제 유형)입니다.

물론 (7)은 수학적으로 불가능합니다. 왜냐하면 무한집합에서 유한집합으로의 맵이 주사적이기 때문입니다. 더 엄격한 가정은 다음과 같습니다.

$$\forall s, s' : s \neq s', P(\mathrm{HASH}(s) = \mathrm{HASH}(s')) = 2^{-k} \quad (8)$$

그러나 이것은 증명을 위해 그렇게 편리하지 않습니다. 어떤 작은 ϵ에 대해 ($\epsilon = 10^{-18}$과 같이) $2^{-k}N < \epsilon$ 인 증명에서 (8)이 최대 N 번 사용된다면, 우리는 실패 확률 (즉, 확률이 최소 $1 - \epsilon$ 이상일 경우 최종 결론은 참으로 나타납니다)을 받아들이면 (7)이 참이라고 판단할 수 있습니다.

마지막 발언: 수식 (8)의 확률 진술을 정말로 엄격하게하기 위해서는 모든 바이트 배열의 집합 $Bytes$*에 확률분포를 도입해야 합니다. 이를 수행하는 방법은 동일한 길이 l의 모든 바이트 배열이 동일하다고 가정하고 길이 $p \to 1-$ 의 배열을 관찰할 확률을 일부 p 에 대해 $p^l - p^{l+1}$ 와 동일하게 설정하는 것입니다. 그러면 (8)은 p가 아래에서 하나가 될 때 조건부 확률 $P(\mathrm{HASH}(s) = \mathrm{HASH}(s') \mid s \neq s')$ 의 한계로 이해해야 합니다.

2.2.10. TON 블록체인에 사용되는 해시 (Hash used for the TON Blockchain). 우리는 당분간 TON 블록체인을 위해 256-비트 SHA256 해시를 사용하고 있습니다. 만약 이것이 예상보다 약한 것으로 나타나면 미래에 다른 해시함수로 대체될 수 있습니다. 해시함수의 선택은 프로토콜의 구성 가능한 매개변수이므로 2.1.21에서 설명한대로 하드포크 없이 변경할 수 있습니다.

2.3. 블록체인 상태, 계정 및 해시맵 (Hash used for the TON Block-chain)

우리는 블록체인이 특정 글로벌 상태를 정의하고 각 블록과 각 트랜잭션이 이 글로벌 상태의 변환을 정의한다는 것을 위에서 언급했습니다. 여기서 우리는 TON 블록체인이 사용하는 글로벌 상태를 설명합니다.

2.3.1. 계정ID (Account IDs). TON 블록체인 또는 적어도 마스터체인 및 워크체인 제로에서 사용되는 기본 계정ID는 특정 타원곡선에 대한 256-비트 ECC 의 퍼블릭 키로 간주되는 256-비트 정수입니다.

이런식으로, $$account\_id : Account = uint_{256} = \mathbf{2}^{256} \quad (9)$$

여기서 $Account$ 는 계정 유형이고 $account\_id : Account$ 는 유형 $Account$의 특정 변수입니다.

다른 워크체인은 다른 계정ID 포맷 (256-비트 또는 기타) 을 사용할 수 있습니다. 예를 들어, ECC 퍼블릭 키의 SHA256과 동일한 비트코인-스타일의 계정ID를 사용할 수 있습니다.

다른 워크체인은 다른 계정ID 포맷 (256-비트 또는 기타) 을 사용할 수 있습니다. 예를 들어, ECC 퍼블릭 키의 SHA256과 동일한 비트코인-스타일의 계정ID를 사용할 수 있습니다.

그러나, 계정ID의 비트 길이 l은 (마스터체인에서) 워크체인을 만드는 동안 고정되어야 하며 *account_id*의 처음 64-비트가 샤딩 및 메시지 라우팅에 사용되므로 비트 길이는 64 이상이어야 합니다.

2.3.2. 주요 구성 요소: 해시맵 (Main component: Hashmaps). TON 블록체인 상태의 주요 구성요소는 해시맵 입니다. 어떤 경우에는 (부분적으로 정의된) "맵" $h : 2^n \dashrightarrow 2^m$ 을 고려합니다. 보다 일반적으로, 우리는 복합형 X에 대해 해시맵 $h : 2^n \dashrightarrow X$에 관심이 있을 수 있습니다. 그러나 원본 (또는 인덱스) 형식은 거의 항상 2^n 입니다.
때로는 "기본 값" $empty : X$ 와 해시맵 $h : 2^n \dashrightarrow X$ 는 "default value" $i \mapsto empty$ 로 " 초기화"됩니다.

2.3.3. 예: TON 계정잔액 (Example: TON account balances). 중요한 예는 TON 계정잔액에 의해 주어집니다. 이것은 해시맵

$$balance : Account \to uint_{128} \tag{10}$$

$Account = 2^{256}$을 $uint_{128} = 2^{128}$ 유형의 그램 (TON 코인) 잔액으로 맵핑합니다. 이 해시맵의 기본값은 0 이며, 처음에는 (첫 번째 블록이 처리되기 전에) 모든 계정의 잔액은 0 임을 의미합니다.

2.3.4. 예: 스마트 컨트랙트 영구저장소 (Example: smart-contract persistent storage). 또 다른 예는 스마트 컨트랙트 영구저장소에 의해 제공되며, 이는 (대략적으로) 해시맵으로 표시될 수 있습니다.

$$Storage : 2^{256} \dashrightarrow 2^{256} \tag{11}$$

이 해시맵의 기본값은 0 입니다. 즉, 영구저장소의 초기화되지 않은 셀은 0 으로 간주됩니다.

2.3.5. 예: 모든 스마트 컨트랙트의 영구저장소 (Example: persistent storage of all smart contracts). *account_id*로 구별되고 별도의 영구저장소가 있는 하나 이상의 스마트 컨트랙트를 가지고 있으므로 실제로 해시맵을 반드시 가져야 합니다.

$$Storage : Account \dashrightarrow (2^{256} \dashrightarrow 2^{256}) \tag{12}$$

스마트 컨트랙트의 *account_id*를 영구저장소에 매핑합니다.

2.3.6. 해시맵 유형 (Hashmap type). 해시맵은 추상적인 (부분적으로 정의된) 함수 $2^n \dashrightarrow X$ 가 아닙니다. 이것은 특정한 표현을 가지고 있습니다. 따라서 특수 해시맵 유형이 있다고 가정합니다.

$$Hashmap(n, X) : Type \tag{13}$$

(부분) 맵 $2^n \dashrightarrow X$ 를 인코딩하는 데이터 구조에 해당합니다. 우리는 또한 다음과 같이 작성할 수 있습니다.

$$Hashmap(n : nat)(X : Type) : Type \tag{14}$$

또는

$$Hashmap : nat \to Type \to Type \tag{15}$$

우리는 항상 $h : Hashmap(n, X)$ 을 맵 $hget\ (h) : 2^n \to X^?$ 로 변환할 수 있습니다. 우리는 이제부터 일반적으로 $hget\ (h)(i)$ 대신에 $h[i]$ 를 작성합니다:

$$h[i] :\equiv hget(h)(i) : X^? \ for\ any\ i : 2^n, h : Hashmap(n, X) \tag{16}$$

2.3.7. 패트리샤 트리로 해시맵유형 정의 (Definition of hashmap type as a Patricia tree). 논리적으로, $Hashmap(n, X)$을 가장자리 라벨 0 과 1을 가진 깊이 n 의 (불완전한) 이진트리로 정의할 수 있고 잎에 X 유형의 값을 가질 수 있습니다. 동일한 구조를 설명하는 또 다른 방법은 길이가 n 인 이진 문자열에 대한 (비트 단위) 트라이 입니다.

실제로는 우리는 부모와 함께 오직 하나의 자식을 가진 각 꼭지점을 압축함으로써 이 트라이의 압축된 표현을 사용하는 것을 선호합니다. 결과 표현은 패트리샤 트리 또는 이진 기수 트리라고 합니다. 각 중간 정점에는 정확히 두 개의 자식이 있으며 두 개의 비어있지 않은 이진 문자열로 명칭이 지정됩니다. 왼쪽 자식에 대해서는 0 으로 시작하고 오른쪽 자식에 대해서는 1 로 시작합니다.

즉, 패트리샤 트리에는 두 가지 유형 (비 루트) 노드가 있습니다:

- LEAF(x), 유형 X 의 값 x 를 포함합니다.
- NODE(l, s_l, r, s_r), 여기서 l 은 왼쪽 자식 또는 서브트리에 대한 참조이고, s_l 은 이 꼭지점을 왼쪽 자식 (항상 0 으로 시작)에 연결하는 가장자리를 표시하는 비트문자열이며, r 은 오른쪽 서브트리이고 s_r 은 오른쪽 자식에게 가장자리를 명칭하는 비트문자열입니다 (항상 1로 시작)

패트리샤 트리의 루트에서 한 번만 사용되는 노드의 세 번째 유형도 필요합니다.

- ROOT(n, s_0 t), 여기서 n 은 *Hashmap*(n, X)의 인덱스 비트문자열의 공통길이이고, s_0 는 모든 인덱스 비트문자열의 공통 접두어이며, t 는 잎 (LEAF) 또는 노드(NODE) 에 대한 참조입니다.

패트리샤 트리가 비어 있도록 하려면 네 번째 유형의 (루트) 노드가 사용됩니다:

- EMPTYROOT(n), 여기서 n 은 모든 인덱스 비트문자열의 공통길이입니다.

패트리샤 트리의 높이를 다음과 같이 정의합니다.

$$\text{HEIGHT}(\text{LEAF}(x)) = 0 \tag{17}$$

$$\text{HEIGHT}(\text{NODE}(l\ ,\ s_l,\ r\ ,\ s_r) = \text{HEIGHT}(l).\ \text{LEN}(s_l) = \text{HEIGHT}(r) + \text{LEN}(s_r) \tag{18}$$

$$\text{HEIGHT}(\text{ROOT}(n\ ,\ s_0,\ t)) = \text{LEN}(s_0) + \text{HEIGHT}(t) = n \tag{19}$$

마지막 두 수식의 마지막 두 표현식은 동일해야 합니다. *Hashmap*(n, X) 유형의 값을 나타내기 위해 높이 n 의 패트리샤 트리를 사용합니다.
트리에 N 잎이 있는 경우 (즉, 해시맵에 N 값이 포함된 경우) 정확히 N - 1 개의 중간 정점이 있습니다. 새로운 값을 삽입하려면 중간에 새 정점을 삽입하고 새 정점의 다른 자식으로 새 잎을 추가하여 기존 가장자리를 분할해야 합니다. 해시맵에서 값을 삭제하면 그 반대입니다: 잎과 부모가 삭제되고 부모의 부모와 그의 다른 자식이 직접 링크됩니다.

2.3.8. 머클-패트리샤 트리 (Merkle-Patricia trees). 블록체인으로 작업할 때 패트리샤 트리 (즉, 해시맵)와 서브트리를 싱글 해시값으로 축소하여 비교할 수 있기를 원합니다. 이것을 달성하는 고전적인 방법은 머클트리에 의해 주어집니다. 본질적으로 우리는 객체 $x : X$ 의 HASH(x) 를 계산하는 방법을 알고 있으면 이진문자열에 대해 정의된 해시함수 HASH 를 사용하여 *Hashmap*(n, X) 유형의 객체 h 를 해싱하는 방법을 설명하고자 합니다 (예. 객체 x 의 이진 직렬화에 해시함수를 적용함으로써).

다음과 같이 HASH(h)를 재귀적으로 정의할 수 있습니다.

$$\text{HASH}(\text{LEAF}(x)) := \text{HASH}(x) \tag{20}$$

$$\text{HASH}(\text{NODE}(l\ ,\ s_l,\ r\ ,\ s_r) := \text{HASH}(\text{HASH}(l).\ \text{HASH}(r).\ \text{CODE}(s_l).\ \text{CODE}(s_r)) \tag{21}$$

$$\text{HASH}(\text{ROOT}(n\ ,\ s_0,\ t)) := \text{HASH}(\text{CODE}(n)\ .\ \text{CODE}(s_0)\ .\ \text{HASH}(t)) \tag{22}$$

여기서, $s.t$ 는 (비트) 문자열 s 와 t 의 접합을 나타내며, CODE(s)는 모든 비트문자열 s 의 접두사 코드입니다. 예를 들어, 0 은 10 으로, 1 은 11 로, 문자열의 끝은 0[7] 으로 인코딩 할 수 있습니다.

나중에 임의의 (의존형) 대수유형의 값에 대해 재귀적으로 정의된 해시의 (약간 수정된) 버전이라는 것을 알 수 있습니다 (cf. **2.3.12, 2.3.14**).

2.3.9. 머클트리 해시를 재계산 (Recomputing Merkle tree hashes). 머클트리 해시라고 하는 HASH(h) 를 재귀적으로 정의하는 이 방법은 각 노드 h' 와 함께HASH(h') 를 명시적으로 저장하면 (머클트리 또는 이 경우에는 머클-패트리샤 트리라고 하는 구조가 됨), 해시맵에 요소가 추가되거나 삭제되거나 변경될 때 최대 n 개의 해시만 재계산해야 합니다.

이런 방식으로, 만약 적절한 머클트리 해시에 의한 글로벌 블록체인 상태를 나타내는 경우, 각 트랜잭션 후에 이 상태 해시를 쉽게 재계산할 수 있습니다.

2.3.10. 머클증명 (Merkle proofs). 선택된 해시함수 HASH 의 "압입"의 가정 (7) 하에서, 주어진 값 z 의 HASH(h), h : $Hashmap(n, X)$는 일부 i : 2^n 과 $x : X$ 에 대해 $hget(h)(i) = x$ 를 갖고 있다는 증거를 만들 수 있습니다. 이러한 증명은 머클-패트리샤 트리에서 i 에 해당하는 잎에서 루트까지의 경로로 구성되며, 이 경로에 모든 노드의 모든 형제 자매가 해시로 증가합니다.

이러한 방식으로, 일부 해시맵 h (예. 스마트 컨트랙트 영구저장소 또는 글로벌 블록체인 상태)에 대해 HASH(h) 값만 알고 있는 라이트 노드[8] 는 풀-노드[9] 로부터 값 $x = h\,[i]$ = hget(h)(i) 뿐만 아니라 이미 알려진 값 HASH(h) 에서 시작하는 머클증명과 함께 그러한 값을 요청할 수도 있습니다. 그런 다음, 가정 (7) 하에서, 라이트 노드는 x 가 실제로 h[i] 의 정확한 값임을 스스로 확인할 수 있습니다.

[7] 이 인코딩은 임의의 또는 연속 인덱스가 있는 패트리샤 트리의 모든 가장자리 명칭 중 약 절반에 대해 최적임을 보여줄 수 있습니다. 나머지 가장자리의 명칭은 길 확률이 많습니다 (즉, 거의 256-비트 길이). 따라서, 가장자리 명칭을 위한 거의 최적의 인코딩은 위의 코드를 사용하여 "짧은" 비트문자열에 접두어 0 을 사용하고, 1 을 인코딩 한 다음, 비트문자열 s 의 길이 l = |s| 를 포함하는 9-비트와 "긴" 비트 스트링 ($l \geq 10$)을 사용하는 것입니다.

[8] 라이트 노드는 샤드체인의 전체 상태를 추적하지 않는 노드입니다. 대신에 가장 최근의 여러 블록의 해시와 같은 최소한의 정보를 유지하고 전체 상태의 일부를 검사해야 하는 경우 풀-노드에서 얻은 정보에 의존합니다.

[9] 풀-노드는 문제의 샤드체인의 완전한 최신 상태를 추적하는 노드입니다.

경우에 따라 클라이언트는 대신 $y = \mathrm{HASH}(x) = \mathrm{HASH}(h\ [i])$ 값을 얻고자 할 수 있습니다 - 예를 들어, x 자체가 매우 큰 경우 (예. 해시맵 자체). 그러면 (i, y) 에 대한 머클증명이 대신 제공될 수 있습니다. 만약 x 또한 해시맵이라면, 값 $x[j] = h[i][j]$ 또는 그 해시를 제공하기 위해 $y = \mathrm{HASH}(x)$ 에서 시작하는 두 번째 머클증명이 풀-노드로부터 얻어질 수 있습니다.

2.3.11. TON과 같은 멀티체인 시스템에 대한 머클증명의 중요성 (Importance of Merkle proofs for a multi-chain system such as TON). 일반적으로 노드는 TON 환경에 있는 모든 샤드체인에 대한 풀-노드가 될 수 없습니다. 대다수의 샤드체인 - 예를 들어 자체계정, 관심있는 스마트 컨트랙트 또는 이 노드가 밸리데이터로 지정된 노드의 경우에만 풀-노드입니다. 다른 샤드체인의 경우 라이트 노드여야 합니다. 그렇지 않으면 저장소, 컴퓨팅 및 네트워크 대역폭 요구사항이 엄격합니다. 이것은 그러한 노드가 다른 샤드체인의 상태에 대한 주장을 직접 검사할 수 없고 샤드체인에 대한 풀-노드에서 얻은 머클증명에 의존해야 합니다. 이는 수식(7)이 실패하지 않는 한 (즉, 해시 충돌이 발견되지 않으면) 스스로 체크하는것 만큼 안전합니다.

2.3.12. TON VM의 특성 (Perculiarities of TON VM). 마스터체인 및 워크체인 제로에서 스마트 컨트랙트를 실행하는데 사용되는 TON VM 또는 TVM은 EVM (이더리움 가상머신)에서 영감을 얻은 기존 설계와는 상당히 다릅니다: 이는 256-비트 정수뿐만 아니라 사실상 (거의 모든) 임의의 "레코드", "구조체" 또는 "sum-product 유형"을 사용하여 하이-레벨 (특히 기능적) 언어로 작성된 코드를 실행하기에 더 적합합니다. 기본적으로 TVM은 프롤로그(Prolog) 또는 얼랑의 구현에 사용된 것과 다른 태그된 데이터 유형을 사용합니다.

먼저 TVM 스마트 컨트랙트의 상태가 그저 해시맵 $2^{256} \to 2^{256}$ 또는 $\mathit{Hashmap}(256, 2^{256})$이 아니라 첫 번째 단계인 $\mathit{Hashmap}(256, X)$ 로 상상할 수 있습니다. 여기서 X 는 여러 개의 생성자가 있는 유형으로, 256-비트 정수와 별도로 다른 해시맵 $\mathit{Hashmap}(256, X)$ 을 비롯한 다른 데이터 구조를 저장할 수 있습니다. 이는 TVM (영구 또는 임시) 저장소의 셀 또는 TVM 스마트 컨트랙트 코드의 변수 또는 배열 요소에 정수가 포함될 수 있지만 완전히 새로운 해시맵을 포함할 수 있음을 의미합니다. 물론 이것은 셀에 256-비트 뿐만 아니라 예를 들어 8-비트 태그가 포함되어 있어 이 256-비트를 어떻게 해석해야 하는지를 설명합니다.

실제로 값은 정확히 256-비트일 필요는 없습니다. TVM에서 사용하는 값 형식은 원시 바이트 배열과 임의의 순서로 혼합된 다른 구조에 대한 참고로 구성되며 원시 데이터 (예. 문자열 또는 정수)에서 포인터를 구별할 수 있도록 적절한 위치에 일부 설명자 바이트가 삽입됩니다 (cf. **2.3.14**).

이 원시값 형식은 임의의 sum-product 대수형식을 구현하는데 사용될 수 있습니다. 이 경우 값에는 원시 바이트가 먼저 포함되며 사용되는 "생성자" (고급 언어의 관점에서)와 원시 바이트 및 참고로 구성된 다른 "필드" 또는 "생성자 인수"가 설명됩니다 (cf. **2.2.5**). 그러나 TVM은 생성자와 그들의 주장 사이의 대응에 대해 아무것도 모릅니다. 바이트 및 참조의 혼합은 특정 설명자 바이트에 의해 명시적으로 설명됩니다.[10]

머클트리 해싱은 임의의 구조로 확장됩니다. 이러한 구조의 해시를 계산하기 위해 모든 참조가 참조된 객체의 해시로 재귀적으로 대체된 다음, 결과 바이트 문자열의 해시 (포함된 설명자 바이트)가 계산됩니다.

이런식으로, **2.3.8**에 설명된 해시맵에 대한 머클트리 해싱은 두 개의 생성자로 *Hashmap*(*n, X*) 유형에 적용되는 임의의 (종속) 대수 데이터 유형에 대한 해싱의 특수한 경우일 뿐입니다. [11]

2.3.13. TON 스마트 컨트랙트의 영구저장소 (Persistent storage of TON smart contracts). TON 스마트 컨트랙트의 영구저장소는 스마트 컨트랙트에 대한 호출 간에 보존되는 "글로벌 변수"로 구성됩니다. 따라서 이것은 하나의 "글로벌 변수"에 각각 해당하는 올바른 유형의 필드로 구성된 "제품", "튜플(tuple)" 또는 "레코드" 유형입니다. 글로벌 변수가 너무 많으면 TON 셀 크기에 대한 글로벌 제한으로 인해 하나의 TON 셀에 들어갈 수 없습니다. 이러한 경우 여러 레코드로 나뉘어 트리 형태로 구성되어 제품 유형 대신 본질적으로 "제품의 제품" 또는 "제품의 제품의 제품" 유형이 됩니다.

2.3.14. TVM 셀 (TVM Cells). 궁극적으로 TON VM은 모든 데이터를 (TVM) 셀 모음에 저장합니다.

[10] TVM 셀에 있는 이 두 설명자 바이트는 전체 참조 수와 총 원시 바이트 수만 설명합니다. 참조는 모든 원시 바이트 전 또는 후에 유지됩니다.

[11] 실제로 , 앞과 노드는 보조 유형인 *HashmapAux*(*n, X*)의 생성자입니다 . *Hashmap*(*n, X*) 유형은 *HashmapAux*(*n, X*) 유형의 값을 포함하는 ROOT와 생성자 ROOT 및 EMPTYROOT를 가지고 있습니다.

각 셀에는 먼저 두 개의 설명자 바이트가 포함되며 이것은 이 셀에 원시 데이터의 바이트 수 (최대 128) 및 다른 셀에 대한 참조 수 (최대 4)를 나타냅니다. 그런 다음 이러한 원시 데이터 바이트 및 참조가 이어집니다. 각 셀은 정확히 한 번 참조되므로 각 셀에 "부모" (이 셀을 참조하는 유일한 셀)에 대한 참조가 포함될 수 있습니다. 그러나 이 참조가 명시적일 필요는 없습니다.

이러한 방식으로, TON 스마트 컨트랙트의 영구 데이터 저장 셀은 스마트 컨트랙트 설명에 보관된 트리의 루트에 대한 참조와 함께 트리로 구성됩니다.[12] 필요하다면, 이 전체 영구저장소의 머클트리 해시가 잎에서 시작하여 셀의 모든 참조를 참조된 셀의 재귀적으로 계산된 해시로 간단히 대체한 다음, 그로 인해 얻어진 바이트 문자열의 해시를 계산합니다.

2.3.15. 임의의 대수유형의 값에 일반화된 머클증명 (Generalized Merkle proofs for values of arbitrary algebraic types). TON VM은 (TVM) 셀로 구성된 트리를 통해 임의의 대수유형의 값을 나타내고 각 셀에는 사실이 셀에 뿌리를 둔 전체 서브트리에 따라 잘 정의된 (재귀적으로 계산된) 머클해시가 있으므로, 알려진 머클해시가 있는 트리의 특정 서브트리가 특정값 또는 특정 해시값을 사용한다는 것을 증명하기 위해 임의의 대수유형의 값 (부분)에 대해 "일반화된 머클증명"을 제공할 수 있습니다. 이것은 2.3.10의 접근법을 일반화하는데, 여기서는 $x[i] = y$ 에 대한 머클증명만이 고려되었습니다.

2.3.16. TON VM 데이터 구조의 샤딩 지원 (Support for sharding in TON VM data structures). 우리는 지나치게 복잡하지 않은 TON VM이 어떻게 고급수준의 스마트 컨트랙트 언어에서 임의의 (종속) 대수 데이터 형식을 지원하는지 설명했습니다. 그러나, 대규모 (또는 글로벌) 스마트 컨트랙트를 분할하려는TON VM 수준에서 특별한 지원이 필요합니다. 이를 위해 해시맵 유형의 특수 버전이 시스템에 추가되어 "맵" $Account \dashrightarrow X$ 에 해당합니다. 이 "맵"은 $Account = 2^m$인 $Hashmap(m, X)$ 와 동일하게 보일 수 있습니다. 그러나, 샤드가 두 개로 분할되거나 두개의 샤드가 병합되면 해당 해시가 자동으로 두 개로 분할되거나 다시 병합되어 해당 샤드에 속한 키만 유지됩니다.

2.3.17. 영구저장소에 대한 결제 (Payment for persistent storage). TON 블록체인의 주목할 만한 특징은 영구적 데이터 (즉, 블록체인의 전체 상태를 확대하기 위함)를 저장하기 위해 스마트 컨트랙트에서 요구되는 지불입니다.

[12] 논리적으로; 2.5.5에 설명된 "셀백" 표현은 모든 중복 셀을 식별하여 직렬화 될 때 이 트리를 지시된 비순환적 그래프 (dag)로 변환합니다 .

이것은 다음과 같이 작동합니다.

각각의 블록은 블록체인 (보통 그램)의 주요 통화로 지정된 두 가지 비율을 선언합니다: 영구저장소에 하나의 셀을 유지하기 위한 가격, 그리고 영구저장소의 일부 셀에서 한 원시 바이트를 유지하기 위한 가격. 각 계정에 사용된 총 셀 수 및 바이트 수에 대한 통계는 상태의 일부로 저장되므로 이 수를 블록 머리글에 선언된 두 비율로 곱하면, 우리는 이전 블록과 현재 블록 사이에 데이터를 유지하기 위해 계정잔액에서 차감된 금액을 계산할 수 있습니다.

그러나 각 블록의 모든 계정 및 스마트 컨트랙트의 영구저장소 사용에 대한 지불은 요구되지 않습니다. 대신, 이 지불이 마지막으로 완료된 블록의 일련번호는 계정 데이터에 저장되며 계정에 대한 작업이 완료되면 (예. 가치이전 또는 메시지는 스마트 컨트랙트가 수신하여 처리), 이전 지불 이후의 모든 블록에 대한 스토리지 사용량 지불은 추가 작업을 수행하기 전에 계정잔액에서 공제됩니다. 이 후에 계정의 잔액이 마이너스가 되면 계정이 파괴됩니다.

워크체인은 하나 또는 두 개의 암호화폐에서 이러한 지속적인 지불균형을 유지하는 "간단한" 계정을 만들기 위해 계정 당 원시 데이터 바이트 수를 "무료" (즉, 영구저장소 지불에 참여하지 않음)로 선언할 수 있습니다. 아무도 계정에 메시지를 보내지 않으면 영구 저장 결제가 수집되지 않으며 무기한으로 존재할 수 있습니다. 그러나 누구나 빈 메시지를 보내어 그러한 계정을 파괴할 수 있습니다. 파괴될 계좌의 원래 잔액의 일부에서 수집된 작은 인센티브는 그러한 메시지를 보낸 사람에게 줄 수 있습니다. 그러나 밸리데이터는 이러한 부실계정을 무료로 파괴하고 단순히 글로벌 블록체인 상태 크기를 줄이고 보상없이 많은 양의 데이터를 보관하지 않기를 기대합니다.

영구 데이터를 유지하기 위해 수집된 지불은 샤드체인 또는 마스터체인의 밸리데이터에게 분배됩니다 (후자의 경우 지분에 비례).

2.3.18. 로컬 및 글로벌 스마트 컨트랙트; 스마트 컨트랙트 인스턴스 (Local and global smart contracts; smart-contract instances). 스마트 컨트랙트는 보통 "일반" 계정과 마찬가지로 스마트 컨트랙트의 에 따라 선택된 하나의 샤드에만 있습니다.

이것은 대부분의 애플리케이션에 통상 충분합니다. 그러나 일부 "고부하" 스마트 컨트랙트는 일부 워크체인의 각 샤드체인에 "인스턴스"를 갖고 싶어할 수 있습니다. 이를 달성하기 위해, 그들은 예를 들어 워크체인 w 의 "루트" 샤드체인 (w, Φ)[13] 에 이 트랜잭션을 커밋하고 큰 커미션[14] 을 지불하여 모든 샤드체인에 트랜잭션 생성을 전파해야 합니다.

이 작업은 각 샤드에 별도의 잔액이 있는 스마트 컨트랙트의 인스턴스를 효과적으로 만듭니다. 원래 생성한 트랜잭션에서 전송된 잔액은 샤드 (w, s) 의 인스턴스에 $2^{-|s|}$ 의 총 잔액의 일부를 제공하여 분배됩니다. 샤드가 두 개의 자식 샤드로 나뉘어지면 글로벌 스마트 컨트랙트의 모든 인스턴스의 잔액이 반으로 나뉩니다; 두 개의 샤드가 병합되면 잔액이 합산됩니다. 경우에 따라 글로벌 스마트 컨트랙트의 인스턴스 분할/병합에는 이러한 스마트 컨트랙트의 특수한 방법이 (지연되어) 실행될 수 있습니다. 기본적으로 잔액은 위에 설명된 대로 분할되고 병합되며 일부 특수 "계정 인덱스" 해시맵도 자동으로 분리되고 병합됩니다 (cf. **2.3.16**).

2.3.19. 스마트 컨트랙트의 분할 제한 (Limiting splitting of smart contracts). 글로벌 스마트 컨트랙트는 영구저장소 비용을 보다 예측 가능하게 만들기 위해 생성시 분할 깊이 d 를 제한할 수 있습니다. 이것은 $|s| \geq d$ 가 있는 샤드체인 (w, s)가 두 개로 나뉘면 두 개의 새로운 샤드체인 중 하나만 스마트 컨트랙트의 인스턴스를 상속받습니다. 이 샤드체인은 결정론적으로 선택됩니다: 각 글로벌 스마트 컨트랙트에는 본질적으로 트랜잭션 생성의 해시인 여러 "*account_id*" 가 있으며 해당 인스턴스는 올바른 샤드에 속하는데 필요한 적절한 값으로 첫 번째 $\leq d$ 비트가 대체된 동일한 를 갖습니다. 이 *account_id*는 분할 후 스마트 컨트랙트 인스턴스를 상속할 샤드를 선택합니다.

2.3.20. 계정/스마트 컨트랙트 상태 (Account/Smart-contract state). 위의 모든 내용을 요약하면 계정 또는 스마트 컨트랙트 상태가 다음으로 구성됩니다.

- 블록체인의 주요 통화의 잔액
- 블록체인의 다른 통화의 잔액
- 스마트 컨트랙트 코드 (또는 해시)

[13] 더 비싼 대안은 마스터체인에 그러한 "글로벌" 스마트 컨트랙트를 발행하는 것입니다.

[14] 이것은 모든 샤드에 대한 일종의 "브로드캐스트" 기능이며, 따라서 매우 비싸야 합니다.

- 스마트 컨트랙트 코드 (또는 해시)
- 스마트 컨트랙트 영구 데이터 (또는 머클해시)
- 사용된 영구저장소 셀 및 원시 바이트 수에 대한 통계
- 스마트 컨트랙트 영구저장소에 대한 지불이 수집된 마지막 시간 (실제로는 마스터체인 블록 번호)
- 통화를 전송하고 이 계정에서 메시지를 보내는데 필요한 퍼블릭 키 (선택 사항: 기본적으로 *account_id* 자체와 동일). 경우에 따라 비트코인 트랜잭션 출력에 대해 수행되는 것과 유사한 보다 정교한 서명 검사코드가 여기에 있을 수 있습니다. *account_id*는 이 코드의 해시와 동일합니다.

또한 다음 데이터를 계정 상태 또는 다른 인덱스된 계정 해시맵에서 보관해야 합니다.

- 계정의 출력 메시지 대기열 (cf. **2.4.17**)
- 최근 전달된 메시지의 모음 (해시) (cf. **2.4.23**)

이 모든 것이 모든 계정에 실제로 필요한 것은 아닙니다. 예를 들어, 스마트 컨트랙트 코드는 스마트 컨트랙트에만 필요하고 "간단한" 계정에는 필요하지 않습니다. 또한 모든 계정의 원금 통화 잔고가 0 이 아닌 경우 (예. 기본 워크체인의 마스터체인 및 샤드체인을 위한 그램) 다른 통화에서는 잔고가 0 일 수 있습니다. 사용되지 않는 데이터를 보관하지 않으려면 (워크체인에 따라) (워크체인 생성 도중) sum-product 유형이 정의됩니다. 사용된 여러 "생성자"를 구별하기 위해 다른 태그 바이트를 사용합니다 (예. TL 생성자; cf. **2.2.5**). 궁극적으로 계정 상태 자체는 TVM 영구저장소의 셀 모음으로 보관됩니다.

2.4 샤드체인간의 메시지 (Message Between Shardchains).

TON 블록체인의 중요한 구성 요소는 블록체인 간의 메시징 시스템입니다. 이 블록체인은 동일한 워크체인 또는 다른 워크체인의 샤드체인일 수 있습니다.

2.4.1. 메시지, 계정 및 트랜잭션: 시스템 조감도 (Messages, accounts and transactions: a bird's eye view of the system). 메시지는 한 계정에서 다른 계정으로 보냅니다. 각 트랜잭션은 하나의 메시지를 수신하고 특정 규칙에 따라 상태를 변경하며 다른 계정에 여러 개의 새 메시지 (아마도 1 또는 0)를 생성하는 계정으로 구성됩니다. 각 메시지는 정확히 한 번 생성되어 수신(또는 전달) 됩니다.

이는 메시지가 계정 (스마트 컨트랙드)과 비슷한 시스템에서 중요한 역할을 한다는 것을 의미합니다. 무한 샤딩 패러다임 (cf. **2.1.2**)의 관점에서, 각 계정은 별도의 "어카운트 체인"에 있으며, 다른 계정의 상태에 영향을 줄 수있는 유일한 방법은 메시지를 보내는 것입니다.

2.4.2. 프로세스 또는 액터로서의 계정; 액터모델 (Accounts as processes or actors; Actor model). 계정 (및 스마트 컨트랙트)을 들어오는 메시지를 처리하고 내부 상태를 변경하고 결과적으로 일부 아웃바운드 메시지를 생성할 수 있는 "프로세스" 또는 "액터"로 생각할 수 있습니다. 이것은 얼랑과 같은 언어에서 사용되는 소위 액터모델과 밀접한 관련이 있습니다 (그러나 얼랑의 액터는 일반적으로 "프로세스"라고 합니다). 인바운드 메시지를 처리한 결과로 새로운 액터 (즉, 스마트 컨트랙트)가 기존 액터에 의해 생성되기 때문에 액터모델과의 통신은 본질적으로 완전합니다.

2.4.3. 메세지 수신자 (Message recipient). 모든 메시지는 대상 워크체인 식별자 w (기본적으로 원래의 샤드체인과 같다고 가정 됨)를 특정으로 하는 수신계정 *account_id* 의 특성화된 수신자를 가집니다. *account_id*의 정확한 형식 (즉, 비트 수)은 *w* 에 따라 다릅니다. 그러나 샤드는 항상 첫 번째 (가장 중요한) 64 비트로 결정됩니다.

2.4.4. 메시지 발신자 (Message sender). 대부분의 경우, 메시지에는 (*w'*, *account_id*) 쌍으로 다시 특성화된 발신자가 있습니다. 만약 (발신자가) 있다면, 메시지 수신자와 메시지값 뒤에 위치합니다. 때로는, 발신자가 중요하지 않거나 블록체인 외부인일 경우 (즉, 스마트 컨트랙트가 아님) 이 필드가 없습니다.

액터모델에서는 메시지에 암시적 발신자가 필요하지 않습니다. 대신 메시지에는 요청에 대한 응답을 보내야 하는 액터에 대한 참조가 포함될 수 있습니다; 대개 발신자와 일치합니다. 그러나 암호화폐 (비잔틴) 환경에서 메시지에 명백한 위조가 불가능한 발신자 필드를 갖는 것이 유용합니다.

2.4.5. 메시지값 (Message value). 메시지의 또 다른 중요한 특성은 소스와 대상 워크체인에서 모두 지원되는 하나 또는 여러개의 암호화폐로 연결된 값입니다. 메시지값은 메시지 수신자 바로 다음에 표시됩니다. 이것은 본질적으로 (*currency_id, value*) 쌍의 목록입니다.

"간단한" 계정 간의 "간단한"값 전송은 일부 값이 첨부된 비어있는 메시지일 뿐입니다. 한편, 약간 더 복잡한 메시지 본문에는 간단한 텍스트 또는 이진 주석 (예. 결제 목적과 관련 있음)이 포함될 수 있습니다.

2.4.6. 외부 메세지 또는 "messages from nowhere" (External messages, or “messages from nowhere”). 일부 메시지는 즉, 그것은 블록체인에 있는 계정 (스마트 컨트랙트 또는 스마트 컨트랙트 아닌)에 의해 생성되지 않은 "아무데서나 오는" 메세지가 시스템에 도착합니다. 가장 일반적인 예는 사용자가 관리하는 계정에서 일부 다른 계정으로 일부 금액을 이전하려고 할 때 발생합니다. 이 경우, 사용자는 "message from nowhere"라는 메시지를 자신의 계정으로 보내어 수신 계정에 지정된 값을 전달하는 메시지를 생성하도록 요청합니다. 이 메시지가 올바르게 서명된 경우 계정이 메시지를 수신하고 필요한 아웃바운드 메시지를 생성합니다.

실제로 미리 정의된 코드를 사용하여 스마트 컨트랙트의 특수한 경우로 "간단한" 계정을 고려할 수 있습니다. 이 스마트 컨트랙트는 한 가지 유형의 메시지만 받습니다. 이러한 인바운드 메시지에는 서명과 함께 인바운드 메시지 전달 (프로세싱)의 결과로 생성될 아웃바운드 메시지 목록이 포함되어야 합니다. 스마트 컨트랙트는 서명을 검사하고, 서명이 올바른 경우 필요한 메시지를 생성합니다.

물론 "messages from nowhere"와 정상적인 메시지는 차이가 있습니다. 왜냐하면 "messages from nowhere"는 가치를 지닐 수 없으므로 "가스" (즉, 프로세싱) 비용을 지불할 수 없기 때문입니다. 대신, 그것은 작은 가스 한도로 시험적으로 버려지기 전에 새로운 샤드체인 블록에 포함되도록 제안됩니다. 실행이 실패한 경우 (서명이 올바르지 않은 경우) "messages from nowhere"는 잘못된 것으로 간주되고 버려집니다. 작은 가스한도로 실행이 실패하지 않으면 메시지는 새로운 샤드체인 블록에 포함되어 완벽하게 처리되고 지불된 가스(처리 용량)는 수신자의 계정에서 정확하게 소비됩니다 "messages from nowhere"는 밸리데이터로의 재분배를 위해 가스 지불금의 상위에 있는 수신자의 계좌에서 공제되는 트랜잭션 수수료를 정의할 수 있습니다.

이러한 의미에서 "messages from nowhere" 또는 "외부 메시지"는 다른 블록체인 시스템 (예. 비트코인 및 이더리움)에서 사용되는 트랜잭션 후보자의 역할을 합니다.

2.4.7. 로그메시지 또는 "messages to nowhere" (Log messages, or “messages to nowhere”). 마찬가지로, 때로는 특별한 메시지가 생성되어 수신자에게 전달되지 않는 특정 샤드체인으로 라우팅 될 수 있습니다.

하지만 문제의 샤드에 대한 업데이트를 받는 사람이 쉽게 관찰할 수 있도록 로그인해야 합니다. 이러한 기록된 메시지는 사용자 콘솔에 출력되거나 오프-체인 서버에서 일부 스크립트의 실행을 트리거 할 수 있습니다. 이러한 의미에서, "블록체인 슈퍼 컴퓨터"의 외부"출력"을 나타냅니다. "messages from nowhere"는 "블록체인 슈퍼 컴퓨터"의 외부 "입력"을 나타내는 것과 같습니다.

2.4.8. 오프-체인 서비스 및 외부블록체인과의 상호작용 (Interaction with off-chain services and external blockchains). 이러한 외부입력 및 출력 메시지는 오프-체인 서비스 및 비트코인 또는 이더리움과 같은 기타 (외부) 블록체인과 상호작용하는데 사용될 수 있습니다. 비트코인, 이더 또는 이더리움 블록체인에 정의된 ERC-20 토큰에 고정된 TON 블록체인 내부에 토큰이나 암호화폐를 생성할 수 있으며, TON 블록체인과 이들 외부블록체인 간에 필요한 상호작용을 구현하기 위해 일부 제 3자 오프-체인 서버에 있는 스크립트로 생성되고 처리되는 "messages from nowhere" 및 "messages to nowhere"를 사용할 수 있습니다.

2.4.9. 메시지 본문 (Message body). 메시지 본문은 단순히 바이트 배열이며, 그 의미는 수신 워크체인 및/또는 스마트 컨트랙트에 의해서만 결정됩니다. TON VM을 사용하는 블록체인의 경우, 이는 Send () 작업을 통해 자동으로 생성된 TVM 셀의 직렬화일 수 있습니다. 이러한 직렬화는 TON VM 셀의 모든 참조를 참조된 셀로 재귀적으로 대체하여 간단히 얻을 수 있습니다. 궁극적으로, 4-바이트 "메시지 유형" 또는 "메시지 생성자"가 앞에 붙는 원시 바이트 문자열이 나타나면 이는 수신 스마트 컨트랙트의 올바른 방법을 선택하는데 사용됩니다.

또 다른 옵션은 TL-직렬화된 객체 (cf. **2.2.5**)를 메시지본문으로 사용하는 것입니다. 이 기능은 다른 워크체인 간의 통신에 특히 유용할 수 있습니다 (하나 또는 둘 다 TON VM을 반드시 사용해야 하는 것은 아닙니다).

2.4.10. 가스한도 및 기타 워크체인 / VM-매개변수 (Gas limit and other work-chain/VM-specific parameters). 간혹 메시지에는 가스한도, 가스가격, 트랜잭션 수수료 및 수신 워크체인에 따라 달라지는 유사한 값에 대한 정보가 포함되어 있어야 합니다. 이는 수신 워크체인에는 관련이 있지만 발신 워크체인에는 해당되지 않을 수도 있습니다. 이러한 매개변수는 메시지본문에 또는 그 전에 포함되며 경우에 따라 (워크체인에 따라 다름) TL-계획에 의해 정의될 수 있는 특수한 4-바이트 접두사가 붙습니다 (cf. **2.2.5**).

2.4.11. 메시지 작성: 스마트 콘트랙트 및 트랜잭션 (Creating messages: smart contracts and transactions). 새 메시지에는 두 가지 출처가 있습니다. 대부분의 메시지는 들어오는 메시지를 처리하기 위해 일부 스마트 컨트랙트가 호출되어 스마트 컨트랙트 실행 (TON VM의 Send ( ) 작업을 통해) 중에 생성됩니다. 또는 외부에서 "외부 메시지" 또는 "messages from nowhere"로 메세지를 보내올 수도 있습니다 (cf. **2.4.6**).[15]

2.4.12. 메시지 전달 (Delivering messages). 메시지가 대상 계정이 포함된 샤드체인에 도달하면 대상 계정[16]에 "전달"됩니다. 다음에 일어나는 일은 워크체인에 따라 다릅니다; 외부 관점에서 볼 때 이러한 메시지를 이 샤드체인에서 더 이상 전달할 수 없도록 하는 것이 중요합니다.

기본 워크체인의 샤드체인의 경우, 수신 계정의 잔액에 메시지 값 (모든 가스 결제를 뺀 값)을 더하고 수신 계정이 스마트 컨트랙트일 경우, 수신 스마트 컨트랙트의 메시지-종속적 방법을 나중에 호출할 수 있습니다. 사실, 스마트 컨트랙트는 모든 수신 메시지를 처리할 수 있는 전입점이 하나뿐이므로 처음 몇 바이트를 보고 다양한 유형의 메시지를 구별해야 합니다 (예. TL 생성자를 포함하는 처음 4 바이트; cf. **2.2.5**).

2.4.13. 메시지 전달은 트랜잭션입니다 (Delivery of a message is a transaction). 메시지 전달은 계정 또는 스마트 컨트랙트의 상태를 변경하기 때문에 수신 샤드체인에서 특별 트랜잭션으로 명시적으로 등록됩니다. 본질적으로 모든 TON 블록체인 트랜잭션은 수신 계정 (스마트 컨트랙트)에 하나의 인바운드 메시지를 전달하는 것으로 구성되며 사소한 기술적인 세부 사항은 무시됩니다.

2.4.14. 동일한 스마트 컨트랙트의 인스턴스 간 메시지 (Messages between instances of the same smart contract). 스마트 컨트랙트는 로컬 일 수도 있고 (즉, 보통 계정처럼 한 샤드체인에 거주할 수도 있음) 글로벌일 수도 있습니다 (즉, 모든 샤드에 인스턴스가 있거나 적어도 알려진 *d* 깊이까지 샤드체인에 있어야합니다; cf. **2.3.18**). 글로벌 스마트 계약의 인스턴스는 필요한 경우 정보와 가치를 서로 전송하기 위해 특별한 메시지를 교환할 수 있습니다. 이 경우 (위조 할 수 없는) 발신자 *account_id*가 중요해집니다 (cf. **2.4.4**).

[15] 위의 내용은 기본 워크체인 및 해당 샤드체인에 대해서만 사실이어야 합니다; 다른 워크체인은 메시지를 만드는 다른 방법을 제공할 수 있습니다.

[16] 퇴화의 경우,이 샤드체인은 본래 샤드체인과 일치할 수 있습니다. 예를 들어 아직 분할되지 않은 워크체인 내부에서 작업하는 경우입니다.

2.4.15. 스마트 컨트랙트의 모든 인스턴스에 대한 메시지; 와일드카드 주소 (Messages to any instance of a smart contract; wildcard addresses). 때로는 메시지 (예. 클라이언트 요청)를 글로벌 스마트 컨트랙드의 인스턴스 (일반적으로 가장 가까운 인스턴스)에 전달해야 할 필요가 있습니다 (만약 발신자와 동일한 샤드체인에 거주하는 경우 해당 후보가 확실합니다). 이를 수행하는 한 가지 방법은 "와일드카드 수신자 주소"를 사용하여 대상 *account_id*의 첫 번째 *d* 비트가 임의의 값을 가질 수 있게 하는 것입니다. 실제로 이러한 *d* 비트는 일반적으로 발신자의 *account_id*와 동일한 값으로 설정됩니다.

2.4.16. 입력대기열 부재 (Input queue is absent). 블록체인 (일반적으로 샤드체인, 때로는 마스터체인) 또는 일부 샤드체인에 있는 "어카운트 체인"에 의해 수신된 모든 메시지는 즉각적으로 전달됩니다 (즉, 수신 계정에서 처리). 따라서 "입력대기열"이 없습니다. 대신 블록 및 가스 사용량의 전체 크기 제한으로 인해 특정 샤드체인에 대한 모든 메시지를 처리할 수 없는 경우 일부 메시지는 원래 샤드체인의 출력 대기열에 누적될 수 있습니다.

2.4.17. 출력대기열 (Output queue). 무한 샤딩 패러다임 (cf. **2.1.2**)의 관점에서, 각 어카운트 체인 (즉, 각 계정)은 자신이 생성한 모든 메시지로 구성되지만 아직 수신자에게 배달되지 않은 자체 출력대기열을 갖고 있습니다. 물론 어카운트 체인은 가상의 존재만 갖고 있습니다; 그들은 샤드체인으로 그룹화되어 있고, 샤드체인은 샤드체인에 속하는 모든 계정의 출력대기열의 합집합으로 구성된 출력"대기열"을 갖습니다.

이 샤드체인 출력"대기열"은 구성원 메시지에 부분 순서만 부과합니다. 즉, 이전 블록에서 생성된 메시지는 후속 블록에서 생성된 메시지보다 먼저 전달되어야 하며 동일한 계정에서 생성되고 동일한 대상을 가진 메시지는 생성 순서대로 전달되어야 합니다.

2.4.18. 신뢰성 있고 빠른 인터-체인 메시징 (Reliable and fast inter-chain messaging). 설령 메세지가 수백만 개가 있어도, TON과 같은 확장 가능한 멀티-블록체인 프로젝트가 다른 샤드체인간에 메시지를 전달하고 전달할 수 있어야 한다는 점이 매우 중요합니다 (cf. **2.1.3**). 메시지는 안정적으로 전달되어야 하며 (즉, 메시지가 한 번 이상 유실되거나 전달 되어서는 안 됨) 신속하게 전달되어야 합니다. TON 블록체인은 두 가지 "메시지 라우팅" 메커니즘을 결합하여 이 목표를 달성합니다.

[17] 이것은 하이퍼큐브 라우팅의 다음 홉을 계산하는데 사용되는 알고리즘의 최종 버전일 필요는 없습니다. 특히, 16진수는 *r*-비트 그룹으로 대체될 수 있으며, *r* 은 구성 가능한 매개변수이며 반드시 4와 같을 필요는 없습니다.

2.4.19. 하이퍼큐브 라우팅: 확실한 메시지 전달을 위한 "느린 경로" (Hypercube routing: “slow path” for messages with assured delivery). TON 블록체인은 필요한 경우 전송을 위해 여러 개의 중간 샤드체인을 사용하여 하나의 샤드체인에서 다른 샤드체인으로 메시지를 전달하는, 속도가 느리지만 안전하고 신뢰할 수 있는 방법으로 "하이퍼큐브 라우팅"을 사용합니다. 그렇지 않으면 주어진 샤드체인의 밸리데이터는 다른 모든 샤드체인의 (출력 대기열) 상태를 추적해야 하므로 샤드체인의 총량이 증가함에 따라 엄청난 양의 컴퓨팅 성능과 네트워크 대역폭이 필요하므로 시스템의 확장성이 제한됩니다. 따라서, 어떤 샤드에서 다른 모든 샤드로 직접 메시지를 전달할 수 없습니다. 대신에 각 샤드는 16진수 샤드 식별자 (cf. **2.1.8**) 하나로 정확하게 다른 샤드($w;\ s$)로 “연결”됩니다. 이 방법으로 모든 샤드체인은 "하이퍼큐브" 그래프를 구성하고 메시지는 이 하이퍼큐브의 가장자리를 따라 이동합니다.

메시지가 현재 샤드와 다른 샤드에 보내지는 경우, 현재 샤드 식별자의 16진수 중 하나가 (결정론적으로 선택됨) 대상 샤드의 해당 숫자로 대체되고 그 결과 식별자가 메시지를 전달하는 근접 목표로 사용됩니다.[17]

하이퍼큐브 라우팅의 가장 큰 장점은 블록 유효성 조건은 샤드체인 블록을 만드는 밸리데이터가 지분을 잃어버리는 고통을 감수하고 “이웃한" 샤드체인의 출력 대기열에서 메시지를 수집하고 처리해야 함을 의미합니다. 이 방법으로 모든 메시지는 조만간 최종 목적지에 도달할 것으로 예상할 수 있습니다; 메시지는 전송 중에 손실되거나 두 번 전달될 수 없습니다.

하이퍼큐브 라우팅은 메시지를 여러 중간 샤드체인을 통해 전달해야 하기 때문에 몇 가지 추가 지연 및 비용을 발생시킵니다. 그러나 이러한 중간 샤드체인의 수는 전체 샤드체인 수 N 의 대수 로그 N (더 정확하게는 $[\log_{16}N]$ - 1) 과 같이 매우 느리게 증가합니다. 예를 들어, $N \approx 250$ 이면 최대 1개의 중간 홉; $N \approx 4000$ 샤드체인의 경우 최대 2 개가 있게 됩니다. 4 개의 중간 홉을 사용하여 최대 1백 만개의 샤드체인을 지원할 수 있습니다. 이는 시스템의 본질적으로 무제한적인 확장성에 대해 지불하는 매우 적은 가격이라고 생각합니다. 실제로는 이 가격조차 지불할 필요가 없습니다:

2.4.20. 인스턴트 하이퍼큐브 라우팅: 메시지를 위한 "빠른 경로" (Instant Hypercube Routing: “fast path” for messages). TON 블록체인의 새로운 특징은 하나의 샤드체인에서 다른 샤드체인으로 메시지를 전달하는 "빠른 경로"를 도입한다는 것입니다.

[17] 이것은 하이퍼큐브 라우팅의 다음 홉을 계산하는데 사용되는 알고리즘의 최종 버전일 필요는 없습니다. 특히, 16 진수는 r-비트 그룹으로 대체될 수 있으며, r 은 구성 가능한 매개변수이며 반드시 4와 같을 필요는 없습니다.

대부분의 경우 **2.4.19**의 "느린"하이퍼큐브 라우팅을 우회하여 최종 목적지 샤드체인의 바로 다음 블록으로 메시지를 전달할 수 있습니다.

아이디어는 다음과 같습니다. "느린" 하이퍼큐브 라우팅 동안, 메시지는 하이퍼큐브의 가장자리를 따라 (네트워크에서) 이동하지만 항해를 계속하기 전에 각 중간 정점에서 지연되어 (약 5초 동안) 해당 샤드체인에 커밋됩니다.

불필요한 지연을 피하기 위해 중간 샤드체인에 커밋하지 않고 하이퍼큐브의 가장자리를 따라 적절한 머클증명과 함께 메시지를 중계할 수 있습니다. 사실 네트워크 메시지는 원본 샤드의 "작업그룹" (cf. **2.6.8**)의 밸리데이터로부터 대상 샤드의 "작업그룹"의 지정된 "블록생산자" (cf. **2.6.9**)로 전달되어야 합니다; 이것은 하이퍼큐브의 가장자리를 따라가지 않고 직접 행해질 수 있습니다. 머클증명을 가진 이 메시지가 대상 샤드체인의 밸리데이터 (보다 정확하게는 조합자; cf. **2.6.5**)에 도달하면 메시지가 "느린 경로"를 따라 이동을 완료할 때까지 기다리지 않고 즉시 새 블록으로 커밋할 수 있습니다. 그런 다음 적합한 머클증명과 함께 전달 확인이 하이퍼큐브 가장자리를 따라 되돌려 보내지며 특별한 트랜잭션을 커밋하여 "느린 경로"를 따라 메시지 이동을 중지하는데 사용될 수 있습니다.

이 "즉시전달" 메커니즘은 "느린", 하지만 **2.4.19**에서 설명된 오류 방지 메커니즘을 대체하지 않습니다. “느린 경로"가 여전히 필요한 이유는 밸리데이터가 잃어버리거나 단순히 “빠른 경로" 메시지를 블록체인의 새로운 블록 [18] 에 커밋하지 않기로 결정한 것에 대해서 처벌할 수 없기 때문입니다.

따라서 두 가지의 메시지 전달 방법은 병렬로 실행되며 "느린" 메커니즘은 "빠른" 메커니즘의 성공 증명이 중간 샤드체인 [19] 에 커밋된 경우에만 중단됩니다.

2.4.21. 인접한 샤드체인의 출력 대기열에서 입력메시지를 수집 (Collecting input messages from output queues of neighboring shardchains). 샤드체인을 위한 새로운 블록이 제안될 때, 인접한 샤드체인 (**2.4.19**의 라우팅 하이퍼큐브의 의미에서)의 출력 메시지의 일부가 새로운 블록에 "입력" 메시지로 포함되어 즉시전달 됩니다 (즉, 처리됩니다).

[18] 그러나 밸리데이터는 느린 경로에서 아직 소비되지 않은 메시지와 관련된 모든 전달 수수료를 수집할 수 있으므로 가능한 빨리 인센티브를 제공합니다.

[19] 사실, 일시적으로 또는 영구적으로 "즉시 전달" 메커니즘을 사용하지 못하게 할 수 있으며, 시스템은 더 천천히 작동하지만 계속 작동합니다.

이웃의 출력 메시지를 처리해야 하는 순서에 관한 규칙이 있습니다. 기본적으로 "오래된" 메시지 (이전 마스터체인 블록을 나타내는 샤드체인 블록에서 가져옴)는 "새로운" 메시지보다 먼저 배달되어야 합니다. 동일한 이웃 샤드체인에서 오는 메시지의 경우 **2.4.17**에 설명된 출력대기열의 부분 순서를 준수해야 합니다.

2.4.22. 출력대기열에서 메시지 삭제 (Deleting messages from output queues). 출력대기열 메시지가 인접 샤드체인에 의해 전달된 것으로 관측되면 특수 트랜잭션에 의해 출력대기열에서 명시적으로 삭제됩니다.

2.4.23. 메시지의 이중전달 방지 (Preventing double delivery of messages). 인접한 샤드체인의 출력대기열에서 가져온 메시지의 이중전달을 방지하기 위해 각 샤드체인 (더 정확하게는 그 내부의 각 어카운트 체인)은 최근 전달된 메시지 (또는 해시)의 상태를 해당 상태의 일부로 유지합니다. 전달된 메시지가 원래의 인접 샤드체인 (cf. **2.4.22**)에 의해 출력대기열에서 삭제된 것으로 관찰되면 최근 전달된 메시지 모음에서도 삭제됩니다.

2.4.24. 다른 샤드체인을 위한 메시지 전달 (Forwarding messages intended for other shardchains). 하이퍼큐브 라우팅 (cf. **2.4.19**)은 때로는 아웃바운드 메시지가 의도된 수신자를 포함하는 샤드체인이 아닌 목적지까지의 하이퍼큐브 경로에 있는 인접한 샤드체인에게 전달된다는 것을 의미합니다. 이 경우 "전달"은 인바운드 메시지를 출력대기열로 이동하는 작업입니다. 이것은 메시지 자체를 포함하는 특별 전달 트랜잭션으로 명시적으로 블록에 반영됩니다. 본질적으로 이것은 마치 샤드체인 내부의 누군가가 메시지를 받고 결과적으로 하나의 동일한 메시지가 생성된 것처럼 보여집니다.

2.4.25. 메세지 전달 및 보존에 대한 지불 (Payment for forwarding and keeping a message). 전달 트랜잭션은 실제로 (전달되는 메시지의 크기에 따라) 약간의 가스를 소비하므로 이 샤드체인의 밸리데이터를 대신하여 전달되는 메시지 값에서 가스 지불액이 차감됩니다. 이 결제전달은 하이퍼큐브 라우팅으로 인해 메시지가 여러번 전달된 경우에도 메시지가 최종적으로 수신자에게 전달될 때 요구되는 가스 지불보다 일반적으로 상당히 작습니다. 또한 여러 샤드체인의 출력대기열에 메시지가 보관되어 있는 경우 샤드체인의 글로벌 상태의 일부이므로 장기간 글로벌 데이터를 유지하기 위한 대금을 특별 트랜잭션으로 징수할 수도 있습니다

2.4.26. 마스터체인과 메시지 송수신 (Messages to and from the masterchain). 메시지는 모든 샤드체인에서 마스터체인으로 직접 보낼 수 있으며 반대의 경우도 마찬가지입니다. 그러나, 마스터체인에서 메시지를 전송하고 처리하기 위한 가스 가격이 매우 높기 때문에 이 기능은 필요한 경우에만 사용됩니다 (예. 밸리데이터가 그들의 지분을 보관하는 경우). 경우에 따라, 메시지가 수신자에 의해 "유효"한 것으로 간주되는 경우에만 마스터체인에 전송되는 메시지의 최소 보증금 (첨부 값)이 정의될 수 있습니다.

마스터 체인을 통해 메시지를 자동으로 라우팅 할 수 없습니다. *workchain_id* ≠ -1 이 첨부된 메시지 (-1 은 마스터체인을 나타내는 특수한 *workchain_id*)는 마스터체인에 전달될 수 없습니다.

원칙적으로, 마스터체인 내부에 메세지 전달 스마트 컨트랙트를 만들 수 있지만 이를 사용하는데 따르는 비용은 엄청납니다.

2.4.27. 동일한 샤드체인에 있는 계정간 메시지 (Messages between accounts in the same shardchain). 경우에 따라 동일한 샤드체인에 있는 다른 계정으로 지정된 일부 샤드체인에 속한 계정에서 메시지가 생성됩니다. 예를 들어, 부하가 관리 가능하기 때문에 아직 여러 샤드체인으로 분할되지 않은 새 워크체인에서 생성됩니다.

이러한 메시지는 샤드체인의 출력대기열에 누적되어 다음 블록에서 수신 메시지로 처리될 수 있습니다 (모든 샤드는 이 목적을 위해 자체의 이웃으로 간주됩니다). 그러나 대부분의 경우 이러한 메시지를 원래 블록 내에서 전달할 수 있습니다.

이를 달성하기 위해 샤드체인 블록에 포함된 모든 트랜잭션에 부분 명령이 부과되고 트랜잭션 (이 중 일부는 일부 계정에 대한 메시지 전달하는 것으로 구성됨)이 이 부분 순서와 관련하여 처리됩니다. 특히, 트랜잭션은 이 부분 순서와 관련하여 선행 트랜잭션의 일부 출력메시지를 처리 할 수 있습니다.

이 경우, 메시지 본문은 두 번 복사되지 않습니다. 대신 원본 및 처리 트랜잭션은 메시지의 공유사본을 참조합니다.

2.5 글로벌 샤드체인 상태. "셀백" 철학 (Global Shardchain State. “Bag of Cells” Philosophy).

이제 TON 블록체인의 글로벌 상태를 설명하거나 또는 적어도 기본 워크체인의 샤드체인을 설명할 준비가 되었습니다.

우리는 글로벌 상태가 대수유형 *ShardchainState* 의 값이라는 말로 구성된 "높은 수준" 또는 "논리적" 설명으로 시작합니다.

2.5.1. 샤드체인 상태는 어카운트 체인 상태의 모음 (Shardchain state as a collection of account-chain states). 무한 샤딩 패러다임 (cf. **2.1.2**)에 따르면 모든 샤드체인은 각각 하나의 계정을 포함하는 가상 "어카운트 체인"을 (임시로) 모아놓은 것입니다. 즉, 본질적으로 글로벌 샤드체인 상태는 해시맵이어야 합니다.

$$\text{ShardchainState} := (\text{Account} \dashrightarrow \text{AccountState}) \tag{23}$$

여기서 샤드의 상태 $(w; s)$ 를 논의할 때 이 해시맵의 인덱스로 나타나는 모든 *account_id* 가 접두어 s 로 시작해야 합니다 (cf. **2.1.8**).
실제로 *AccountState*를 여러 부분으로 분할하고 (예. 인접한 샤드체인에 의한 검사를 단순화하기 위해 계정 출력메시지 대기열을 별도로 유지) *ShardchainState* 내에 여러 해시맵 ($Account \dashrightarrow AccountStatePart_i$)을 만들 수 있습니다. *ShardchainState* 에 소수의 "글로벌" 또는 "필수적인" 매개변수 (예. 이 샤드에 속한 모든 계정의 총 잔액 또는 모든 출력대기열의 총 메시지 수)를 추가할 수도 있습니다.

그러나, $ShardchainState := Account \dashrightarrow AccountState$ 는 적어도 "논리적" ("높은 수준") 관점에서 글로벌 샤드체인 상태가 어떻게 보이는지에 대한 첫 번째 근사값입니다. *AccountState*와 *ShardchainState* 대수유형에 대한 공식적인 설명은 다른 곳에서 제공되는 TL-계획 (cf. **2.2.5**)의 도움으로 수행될 수 있습니다.

2.5.2. 샤드체인 상태 분할 및 병합 (Splitting and merging shardchain states). 샤드체인 상태에 대한 무한 샤딩 패러다임 설명은 샤드가 분할되거나 병합될 때 이 상태가 어떻게 처리되어야 하는지를 보여줍니다. 실제로 해시맵을 사용함으로 이러한 상태변환은 매우 단순한 연산으로 판명됩니다.

2.5.3. 어카운트 체인 상태 (Account-chain state). (가상) 어카운트 체인 상태는 *AccountState* 유형으로 설명되는 하나의 계정 상태입니다. 일반적으로 사용되는 특정 생성자에 따라 **2.3.20**에 나열된 필드 중 일부 또는 전부를 포함합니다.

2.5.4. 글로벌 작업체인 상태 (Global workchain state). 식 (23)과 마찬가지로 글로벌 워크체인 상태를 같은 수식으로 정의할 수 있지만 *account_id*는 하나의 샤드에 속한 값뿐만 아니라 어떤 값도 가질 수 있습니다. 이 경우에도 **2.5.1**에서 만든 것과 유사한 비고가 적용됩니다. 이 해시맵을 여러개의 해시맵으로 분할하고 전체 잔액과 같은 일부 "정수"매개변수를 추가할 수 있습니다.

근본적으로 글로벌 워크체인 상태는 샤드체인 상태와 동일한 유형의 *ShardchainState* 로 주어져야 합니다.

왜냐하면 이 워크체인의 모든 기존 샤드체인이 갑자기 하나로 합쳐지면 얻을 수 있는 샤드체인 상태이기 때문입니다.

2.5.5. 낮은 수준 관점: "셀백" (Low-level perspective: “bag of cells”). 위에 제공된 "높은 수준" 설명을 보완하는 어카운트 체인 또는 샤드체인 상태에 대한 "낮은 수준" 설명이 있습니다. 이 설명은 TON 블록체인에서 사용되는 거의 모든 데이터 (블록, 샤드체인 상태, 스마트 컨트랙트 스토리지, 머클증명 등)를 네트워크로 표현, 저장, 직렬화 및 전송하기 위한 공통기반을 제공하는 보편적인 것으로 판명되었기 때문에 매우 중요합니다. 동시에, 보편적인 "낮은 수준" 설명이 일단 이해되고 구현되면, 우리는 "높은 수준" 고려 사항에만 집중할 수 있게 해줍니다.

TVM은 TVM 셀의 트리 또는 간단하게 셀을 사용하여 임의의 대수유형 (예. (23)의 *ShardchainState* 포함)의 값을 나타냅니다 (cf. **2.3.14** 및 **2.2.5**).

이러한 셀은 특정 플래그 및 $0 \le b \le 128$, 원시바이트의 수 및 $0 \le c \le 4$ 인 다른 셀에 대한 참조 수를 정의하는 두 개의 설명자 바이트들로 구성됩니다. 그런 다음 b 원시 바이트 및 c 셀 참고가 이어집니다.[20]

셀 참조의 정확한 형식은 구현 및 셀이 RAM, 디스크, 네트워크 패킷, 블록 등에 있는지 여부에 따라 다릅니다. 유용한 추상 모델은 셀의 주소가 (SHA256) 해시와 동일한 내용 주소 지정 메모리에 모든 셀이 보관된다는 것을 상상하는 것입니다. 셀의 머클해시는 자식 셀에 대한 참조를 (재귀적으로 계산된) 해시로 대체하고 결과 바이트 문자열을 해싱하여 정확히 계산됩니다.

이러한 방식으로 셀 해시를 사용하여 셀을 참조하면 (예. 다른 셀의 내부설명) 시스템이 다소 단순화되고 셀의 해시가 이를 나타내는 바이트 문자열의 해시와 일치하기 시작합니다.

이제 우리는 샤드체인 상태를 포함하는 TVM이 표현할 수 있는 모든 객체가 "셀백"으로 표현 될 수 있음을 확인합니다. 즉, (예.해시에 의해) 셀들 중 하나에 대한 "루트" 참조와 함께 셀들의 집합을 저장합니다. 중복 셀은 이 설명에서 제거됩니다 ("셀백"은 셀의 멀티세트가 아니라 셀의 세트입니다). 따라서 추상 트리 표현은 실제로 직접 비순환 그래프 (dag) 표현이 될 수 있습니다.

[20] 셀의 트리에 저장된 모든 데이터가 동등하게 자주 필요하다는 머클증명이 있는 경우, 평균 머클증명 크기를 최소화 하기 위해 $b+ch \approx 2(h+r)$ 인 셀을 사용해야 합니다. 여기서 $h = 32$는 해시 사이즈 (바이트)이며, $r \approx 4$는 셀 참조의 "바이트 사이즈"입니다. 다시 말해, 셀은 두 개의 참조와 원시 바이트 몇 개, 또는 하나의 참조와 약 36개의 원시 바이트를 포함하거나 전혀 72개의 원시 바이트가 없는 참조를 포함해야 합니다.

셀 해시로 인덱싱된 문제의 모든 셀 (서브트리 높이 또는 참조 카운터와 같은 일부 추가 데이터가 있는 경우)이 포함된 *B*- 또는 *B*+- 트리에 이 상태를 디스크에 보관할 수도 있습니다. 그러나 이 아이디어를 순진하게 구현하면 스마트 컨트랙트가 디스크 파일의 먼 부분에 흩어지기 때문에 오히려 피해야 합니다.[21]

이제 우리는 TON 블록체인이 어떻게 "셀백"으로 표현되어 이 접근법의 보편성을 입증할 수 있는지 설명하겠습니다.

2.5.6. 샤드체인 블록 "셀백" (Shardchain block as a “bag of cells”). 샤드체인 블록 자체는 대수유형으로도 기술할 수 있으며 "셀백"으로 저장할 수 있습니다. 그런 다음 블록의 단순한 2진 표현은 "셀백"의 각 셀을 나타내는 바이트 문자열을 임의 순서로 연결하여 간단히 얻을 수 있습니다. 이 표현은 블록의 시작 부분에 있는 모든 셀의 오프셋 목록을 제공하고 가능한 경우 이 목록의 32-비트 인덱스로 다른 셀의 해시 참조를 대체함으로써 향상 및 최적화 될 수 있습니다. 그러나 블록이 본질적으로 "셀백"이라고 상상해야 하며 다른 모든 기술적 세부사항은 사소한 최적화 및 구현 문제일 뿐입니다.

2.5.7. 개체를 "셀백"으로 업데이트 (Update to an object as a “bag of cells”).
우리가 "셀백"으로 표현된 일부 객체의 구버전을 가지고 있다고 가정하고 이전 객체와 크게 다르지 않은 동일한 객체의 새 버전을 표현하고자 한다고 가정 해보십시오. 하나는 단순히 자신의 루트를 가진 또 다른 "셀백"으로 새로운 상태를 나타내고 오래된 버전에서 발생하는 모든 셀을 제거할 수 있습니다. 나머지 "셀백"은 근본적으로 개체에 대한 업데이트입니다. 이 객체의 오래된 버전과 업데이트를 가진 모든 사람들은 두 개의 셀 봉합과 이전 루트 제거 (참조 카운터 감소 및 참조 카운터가 0 이 되면 셀 할당 취소)를 통해 새 버전을 계산할 수 있습니다.

2.5.8. 계정 상태 업데이트 **(Updates to the state of an account).** 특히, 계정의 상태 또는 샤드체인의 글로벌 상태 또는 임의의 해시맵에 대한 업데이트는 **2.5.7**에 설명된 개념을 사용하여 나타낼 수 있습니다.

[21] 보다 나은 구현은 만약 작은 경우, 스마트 계약의 상태를 직렬화된 문자열에 또는 큰 경우에는 별도의 *B*-트리에 보관하는 것입니다. 블록체인의 상태를 나타내는 최상위 구조는 잎이 다른 *B*-트리에 대한 참조를 포함할 수 있는 *B*-트리가 됩니다.

즉, 새로운 샤드체인 블록 ("셀백")을 받으면 이 "셀백"을 그 자체로 해석하지 않고 먼저 이전 샤드체인 상태를 나타내는 "셀백"과 결합시킵니다. 이 의미에서 각 블록은 블록체인의 전체 상태를 "포함"할 수 있습니다.

2.5.9. 블록 업데이트 (Updates to a block). 블록 자체는 "셀백"이므로 블록을 수정해야 하는 경우 이와 유사하게 이 블록의 이전 버전인 “셀백"이 있는 것으로 해석되는 "블록 업데이트"를 "셀백"으로 정의할 수 있습니다. 이것은 **2.1.17**에서 논의된 "수직 블록"의 대략적인 아이디어입니다.

2.5.10. "셀백"으로 머클증명 (Merkle proof as a “bag of cells”). 공지된 (일반화된) 머클증명은, 예를 들어, 하나는 $x\ [i] = y$ 알려진 값인 $\mathrm{HASH}(x) = h$ (cf. **2.3.10, 2.3.15**)에서 시작한다는 것을 주장하는 "셀백"으로 표현할 수 있습니다. 즉, $x : \mathit{Hash\text{-}map}(n, X)$의 루트에서 인덱스 $i : 2^n$ 및 값 $y : X$ 를 갖는 원하는 잎까지의 경로에 해당하는 셀의 한 서브세트를 제공하면 됩니다. 이 경로에서 있지 않은 자식 세포의 참조는 셀 해시로 대표되는 이 증명에서 "해결되지 않음" 상태로 남을 것입니다. 또한, 루트로부터 두 경로의 합집합에 있는 셀을 "셀백"에 포함시킴으로써, $x[i] = y$와 $x[i'] = y'$의 동시 머클 증명을 제공 할 수 있습니다 인덱스 i 와 i' 에 해당하는 x부터 잎까지.

2.5.11. 풀-노드에서 쿼리 응답으로 머클증명 (Merkle proofs as query responses from full nodes). 본질적으로 샤드체인 (또는 어카운트 체인) 상태의 완전한 복사본을 가진 풀노드는 라이트노드 (예. TON 블록체인 클라이언트의 라이트 버전을 실행하는 네트워크 노드)가 요청할 때 머클증명을 제공하여 수신자는 이 머클증명에서 제공된 셀만 사용하여 외부 도움이 없이도 간단한 쿼리를 수행할 수 있습니다. 라이트노드는 쿼리를 직렬화된 형식으로 풀-노드에 전송하고 머클증명 (또는 머클증명만)을 사용하여 올바른 응답을 받을 수 있습니다. 왜냐하면 요청자가 머클증명에 포함된 셀만 사용하여 응답을 계산할 수 있어야 하기 때문입니다. 이 머클증명은 라이트 노드의 쿼리를 실행하는 동안 풀-노드가 액세스한 샤드체인 상태에 속한 셀만 포함하는 "셀백"으로 구성됩니다. 이 접근법은 특히 스마트 컨트랙트의 "get 쿼리" 실행에 사용할 수 있습니다 (cf. **4.3.12**).

2.5.12. 유효성의 머클증명으로 확장 업데이트 또는 상태 업데이트 (Augmented update, or state update with Merkle proof of validity). 우리는 "업데이트"를 통해 이전 값 $x : X$ 에서 새로운 값 $x' : X$ 로 객체 상태의 변화를 설명할 수 있습니다 (cf. **2.5.7**).

이 값은 단순히 "셀백"이며 새 값 x' 를 나타내는 서브트리에 있지만 이전 값 x 를 나타내는 서브트리에 없는 셀을 포함합니다. 왜냐하면 수신자는 이전 값 x 와 모든 셀의 사본을 갖고 있다고 가정하기 때문입니다.

그러나 수신자가 x 의 전체 사본을 가지고 있지는 않지만 (머클) 해시 $h = \mathrm{HASH}(x)$ 만 알고 있는 경우 업데이트의 유효성을 확인할 수 없습니다 (즉, 모든 "매달려 있는" 셀 업데이트의 참조는 x 의 트리에 있는 셀을 나타냅니다.) 하나는 기존의 모든 언급된 셀이 존재한다는 머클증명에 의해 보강된 "검증가능한" 업데이트를 원합니다. 그렇다면 오직 $h = \mathrm{HASH}(x)$ 만 알고 있는 사람이라면 업데이트의 유효성을 검사하고 새로운 $h' = \mathrm{HASH}(x')$ 를 계산할 수 있습니다.

머클증명은 "셀백" 자체이므로 (cf. **2.5.10**), x 의 오래된 루트와 x 의 루트에서부터 그들까지의 경로에 있는 후손 중 일부 및 x' 의 새로운 루트와 x 의 일부가 아닌 모든 후손을 포함하는 "셀백"과 같은 확장 업데이트를 구성할 수 있습니다.

2.5.13. 샤드체인 블록의 계정 상태 업데이트 (Account state updates in a shardchain block). 특히, 샤드체인 블록의 계정 상태 업데이트는 **2.5.12**에서 논의된 대로 확장되어야 합니다. 그렇지 않으면 누군가가 유효하지 않은 상태 업데이트를 포함하는 블록을 커밋하고 이전 상태에없는 셀을 참조할 수 있습니다; 그러한 블록의 무효를 증명하는 것은 문제가 될 것입니다 (도전자는 셀이 이전 상태의 일부가 아님을 어떻게 증명할 수 있습니까?).

이제 블록에 포함된 모든 상태 업데이트가 확장되면 유효성을 쉽게 확인할 수 있으며 무효성은 (일반화된) 머클 해시의 재귀적 정의 속성을 위반하는 것으로 쉽게 표시됩니다.

2.5.14. "모든 것은 셀백이다"라는 철학 (Everything is a bag of cells” philosophy). 이전의 고려사항은 우리가 TON 블록체인 또는 네트워크에서 저장 또는 전송해야 하는 모든 것이 "셀백"으로 표현될 수 있음을 보여줍니다. 이것은 TON 블록체인 디자인 철학의 중요한 부분입니다. 일단 "셀백" 접근법이 설명되고 "셀백"의 일부 "낮은 수준" 직렬화가 정의되면, 높은 수준의 추상 (종속) 대수 데이터 유형에 대한 모든 것을 (블록 형식, 샤드체인 및 계정상태 등등) 간단히 정의할 수 있습니다.

"모든 것은 셀백이다"라는 철학은 겉보기엔 무관한 서비스의 구현을 상당히 단순화합니다. 결제채널과 관련된 예는 **5.1.9**를 참조하십시오.

2.5.15. TON 블록체인을 위한 블록 "헤더" (Block “headers” for TON blockchains). 일반적으로 블록체인의 블록은 이전 블록의 해시, 생성시간, 블록에 포함된 모든 트랜잭션 트리의 머클 해시 등을 포함하는 작은 헤더로 시작합니다. 그런 다음 블록

블록 헤더는 궁극적으로 블록에 포함된 모든 데이터에 의존하기 때문에 해시를 변경하지 않고는 블록을 변경할 수 없습니다.

TON 블록체인의 블록에서 사용되는 "셀백" 접근 방식에서는 지정된 블록 헤더가 없습니다. 대신 블록 해시는 블록의 루트 셀의 (머클) 해시로 정의됩니다. 따라서 블록의 최상위 (루트) 셀은 이 블록의 작은 "헤더"로 간주될 수 있습니다.

그러나 루트 셀에는 일반적으로 이러한 헤더에서 예상되는 모든 데이터가 포함되지 않을 수 있습니다. 기본적으로 헤더에 블록 데이터 유형에 정의된 일부 필드가 포함되기를 원합니다. 일반적으로 이 필드는 루트를 포함하여 여러 셀에 포함됩니다. 이것은 함께 문제의 필드 값에 대한 "머클증명"를 구성하는 셀입니다. 어떤 이는 다른 모든 셀보다 맨 처음에 이러한 "헤더 셀"을 포함한다고 주장할 수도 있습니다. 그러면 모든 "헤더 셀"을 얻고 모든 예상 필드를 배우기 위해 블록 직렬화의 처음 몇 바이트만 다운로드 해야 합니다.

2.6 새 블록 생성 및 유효성 검사 (Creating and Validating New Blocks)

TON 블록체인은 궁극적으로 샤드체인 및 마스터체인 블록으로 구성됩니다. 이러한 블록은 시스템이 원활하고 올바르게 기능하도록 하기 위해 네트워크를 통해 생성, 검증 및 모든 관련 당사자에게 전파되어야 합니다.

2.6.1. 밸리데이터 (Validators). 새 블록은 밸리데이터라고 하는 특별지정된 노드에 의해 작성되고 검증됩니다. 근본적으로, 밸리데이터가 되기를 원하는 노드는 충분히 큰 지분 (TON 코인. 즉. 그램; cf. 부록 **A**)을 마스터체인에 보관할 수 있다면, 하나가 될 수 있습니다. 밸리데이터는 좋은 일에 대한 “보상", 즉 새로 생성된 블록에 투입된 모든 트랜잭션 (메시지)의 트랜잭션, 저장 및 가스 수수료 및 일부 신품 코인을 획득하여 TON 블록체인 작동을 유지한 전체 벨리데이터 커뮤니티에게 "사의" 를 반영합니다. 이 수입은 지분에 비례하여 모든 참여 밸리데이터에게 배분됩니다.

그러나 밸리데이터가 되는 것은 높은 책임을 지는 것입니다. 밸리데이터가 유효하지 않은 블록에 서명하는 경우, 지분의 일부 또는 전체를 잃어 버리거나 밸리데이터 집합에서 일시적으로 또는 영구적으로 제외되어 처벌될 수 있습니다. 밸리데이터가 블록을 만드는 데 참여하지 않으면 해당 밸리데이터는 해당 블록과 관련된 보상을 공유하지 않습니다. 밸리데이터가 오랫동안 새로운 블록을 만들지 않을 경우 스테이크의 일부가 손실되어 일시 중단되거나 밸리데이터 집합에서 영구적으로 제외 될 수 있습니다.

이 모든 것은 밸리데이터가 "아무것도 하지 않고" 돈을 얻지 못한다는 것을 의미합니다. 실제로 벨리데이터는 모든 또는 일부 샤드체인의 상태를 반드시 파악해야 하고 (각 밸리데이터는 특정 샤드체인의 서브세트에서 새 블록의 검증 및 생성을 담당합니다), 이 샤드체인의 스마트 컨트랙트에서 요청한 모든 계산을 수행해야 하는 등 다른 샤드체인에 대한 업데이트를 수신해야 합니다. 이 작업은 상당한 디스크 공간, 컴퓨팅 성능 및 네트워크 대역폭이 필요합니다.

2.6.2. 마이너 대신 밸리데이터 (Validators instead of miners). TON 블록체인은 비트코인, 현재 버전의 이더리움 및 기타 대부분의 암호화폐에서 채택한 작업증명방식 대신 지분증명방식을 사용합니다. 이것은 어떤 작업증명 (쓸모없는 해시를 많이 계산)을 제시하여 새로운 블록을 "채굴"하여 새로운 코인을 얻을 수 없다는 것을 의미합니다. 대신 하나의 밸리데이터가 되어야 하며 TON 블록체인의 요청 및 데이터를 저장하고 처리하기 위해 컴퓨팅 리소스를 사용해야 합니다. 간단히 말해서, 새로운 코인을 채취하는 밸리데이터여야 합니다. 이 점에서 밸리데이터는 새로운 마이너입니다.

그러나 밸리데이터가 되는것 이외에 코인을 얻는 다른 방법이 있습니다.

2.6.3. 추천인과 "마이닝 풀" (Nominators and “mining pools”). 밸리데이터가 되려면 일반적으로 여러대의 고성능 서버를 구입하여 설치하고 적절한 고성능 인터넷 연결을 확보해야 합니다. 이것은 비트코인을 채굴하는데 현재 필요한 ASIC 장비만큼 비싸지 않습니다. 그러나 스마트폰은 말할 것도 없고 가정용 컴퓨터로 새로운 TON 코인을 채굴할 수는 없습니다.

비트코인, 이더리움 및 기타 작업증명 암호화폐 채굴 커뮤니티에는 새로운 블록을 자체적으로 채굴할 수 있는 컴퓨팅 파워가 부족한 많은 노드가 자신의 노력을 결합하고 나중에 보상을 공유하는 마이닝-풀 개념이 있습니다. 지분증명 세계에서 상응하는 개념은 추천인 입니다. 본질적으로 이것(추천인)은 밸리데이터가 지분을 늘리는데 도움이되는 돈을 빌려주는 노드입니다. 그런 다음 밸리데이터는 보상의 해당 일부 (또는 이전에 동의한 부분 - 예. 50%)를 추천인에게 분배합니다.

이런 식으로, 추천인은 또한 "채굴"에 참여하여 이 목적을 위해 기탁하고자 하는 금액에 비례하여 약간의 보상을 얻을 수 있습니다. 이는 "자본"만 제공하기 때문에 밸리데이터 보상의 해당 지분 중 일부만 받지만 컴퓨팅 파워, 저장소 및 네트워크 대역폭은 구입할 필요가 없습니다.

그러나 잘못된 행동으로 인해 밸리데이터가 지분을 잃는 경우 추천인 또한 지분의 일부를 잃습니다. 이러한 의미에서 추천인은 위험을 공유합니다. 추천할 밸리데이터를 현명하게 선택해야 합니다. 그렇지 않으면 돈을 잃을 수 있습니다. 이러한 의미에서, 추천인은 가중치를 적용하여 특정 밸리데이터에게 자신의 자금으로 "투표"합니다.

반면에 이 추천인 지명 또는 대출 시스템은 먼저 그램 (TON 코인)에 많은 돈을 투자하지 않고도 밸리데이터가 될 수 있게 만듭니다. 다시 말해, 대량의 그램을 보유한 이들이 밸리데이터의 공급을 독점하지 못하게 합니다.

2.6.4. 어부: 다른 사람의 실수를 지적하여 돈을 얻는다 (Fishermen: obtaining money by pointing out others' mistakes). 밸리데이터가 되지 않고 보상을 받는 또 다른 방법은 어부 가 되는 것입니다. 근본적으로 모든 노드는 마스터체인에 작은 입금을 함으로써 어부가 될 수 있습니다. 그런 다음 특수 마스터체인 트랜잭션을 사용하여 밸리데이터가 이전에 서명하고 게시한 일부 블록 (일반적으로 샤드체인)의 유효성 검사를 게시합니다. 다른 밸리데이터가 이 무효증명에 동의하면 위반한 밸리데이터가 처벌되며 (그들 지분의 일부 잃음으로써) 어부는 보상 (위반한 밸리데이터에게서 압수한 코인의 일부)을 받습니다. 그 후, 유효하지 않은 샤드체인 블록은 **2.1.17**에 설명 된대로 수정되어야 합니다. 잘못된 마스터체인 블록을 수정하려면 이전에 커밋된 마스터체인 블록 위에 "수직" 블록을 생성해야 합니다 (cf. **2.1.17**); 마스터체인 포크를 생성할 필요가 없습니다. 일반적으로 어부는 적어도 일부 샤드체인에 대해 풀-노드가 되어야 하고, 적어도 일부 스마트 컨트랙트의 코드를 실행하여 일부 컴퓨팅 리소스를 사용해야 합니다. 어부는 밸리데이터 만큼의 컴퓨팅 성능을 가질 필요는 없지만, 어부가 되는 자연스런 후보자는 새로운 블록을 처리 할 준비가 된 아직 "선출되지 않은" 밸리데이터라고 생각합니다 (예. 충분히 큰 지분을 예금하지 못했기 때문).

2.6.5. 콜레이터 : 밸리데이터에게 새로운 블록을 제안하는 댓가로 얻은 돈(Collators: obtaining money by suggesting new blocks to validators). 밸리데이터가 되지 않고 일부 보상을 얻는 또 다른 방법은 콜레이터 가 되는 것입니다.

이것은 밸리데이터에게 새로운 샤드체인 블록 후보자를 제안하고, 이 샤드체인의 상태와 다른 머클증명과 함께 다른 샤드체인 (일반적으로 이웃)에서 가져온 데이터로 보완(조합)하는 노드입니다 (이것은 일부 메시지를 인접한 샤드체인에서 전달해야 하는 경우에 필요합니다). 그런 다음 밸리데이터는 이 샤드체인 또는 다른 샤드체인의 전체 상태를 다운로드하지 않고도 제안된 블록 후보의 유효성을 쉽게 확인할 수 있습니다.

밸리데이터는 일부 ("마이닝") 보상을 얻기 위해 새로운 (조합된) 블록 후보를 제출해야 하기 때문에 적절한 블록 후보를 제공하려는 콜레이터에게 보상의 일부를 지불하는 것이 좋습니다. 이 방법으로 밸리데이터는 인접한 샤드체인의 상태를 콜레이터에게 아웃소싱하여 볼 필요가 없습니다. 그러나 시스템의 초기 배포 단계에서는 모든 밸리데이터가 자체적으로 콜레이터로 작동할 수 있기 때문에 별도의 지정된 콜레이터가 없을 것으로 예상합니다.

2.6.6. 콜레이터 또는 밸리데이터: 사용자 트랜잭션 포함하는 댓가로 얻는 돈 (Collators or validators: obtaining money for including user transactions). 사용자는 일부 콜레이터 또는 밸리데이터에 대해 소액결제채널을 열어 사용자의 트랜잭션을 샤드체인에 거래를 포함하는 대가로 소량의 코인을 지불할 수 있습니다.

2.6.7. 글로벌 밸리데이터 세트 선출 (Global validator set election). 밸리데이터의 "글로벌" 세트는 매월 한 번 (실제로는 2^{19} 개의 마스터체인 블록마다) 선출됩니다. 이 세트는 보편적으로 1개월 전부터 결정되고 널리 알려집니다.

밸리데이터가 되려면 노드가 일부 TON 코인 (그램)을 마스터체인으로 전송해야 하고 제안한 지분 s 를 스페셜 스마트 컨트랙트로 보내야 합니다. 지분과 함께 전송되는 또 다른 매개변수는 $l \geq 1$이며, 이 노드가 최소한으로 허용하는 최대 유효성 검사 로드입니다. 또한 L 에 1 (예. 10)과 같은 글로벌 상한 (또 다른 구성 가능한 매개변수)이 있습니다.

그런 다음 밸리데이터의 글로벌 세트가 이 스마트 컨트랙트에 의해 선출됩니다. 간단히 말하면 최대한의 제안한 지분으로 최대 T 후보자를 선택하고 그들의 식별자를 알리는 것입니다. 원래 밸리데이터의 총 수는 $T = 100$입니다: 우리는 부하가 증가하면 1000으로 증가할 것으로 예상합니다. 이것은 구성 가능한 매개변수입니다 (cf. **2.1.21**).

각 밸리데이터의 실제 지분은 다음과 같이 계산됩니다: 만약 상위 T 제안된 지분이 $s_1 \geq s_2 \geq \cdots \geq s_T$ 인 경우 i-번째 밸리데이터의 실제 지분은 $s'_i := min(s_i, l_i \cdot s_T)$으로 설정됩니다. 이런 식으로, $s'_i / s'_T \leq l_i$이므로 i-번째 밸리데이터는

(로드가 궁극적으로 지분에 비례하므로) 가장 약한 밸리데이터의 부하보다 $l_i \leq L$ 배 이상을 얻지 못합니다.

그러면 선출된 밸리데이터는 그들의 지분의 사용하지 않은 일부, $s_i - s'_i$ 을 회수할 수 있습니다. 낙선한 밸리데이터 후보자는 제안한 모든 지분을 회수할 수 있습니다. 각 밸리데이터는 본인의 퍼블릭 서명 키를 게시합니다. 그것은 지분이 나온 계좌의 퍼블릭 키와 반드시 동일한 것은 아닙니다.[22]

밸리데이터의 지분은 선출된 기간이 끝나고 새로운 논쟁 (즉, 밸리데이터 중 한 명이 유효하지 않은 블록을 서명한 경우)이 발생할 경우를 대비해 그 이후로 한 달 후까지 동결됩니다. 그 후, 밸리데이터가 주조한 코인의 몫과 이 기간 동안 처리된 트랜잭션 수수료와 함께 지분을 돌려받습니다.

2.6.8. 밸리데이터 "작업그룹" 선출 (Election of validator “task groups”). 전체 밸리데이터 세트는 새 마스터체인 블록의 유효성 확인에만 사용됩니다 (각 밸리데이터는 그 지분과 동일한 다중도를 가진 것으로 간주됩니다 - 그렇지 않으면 밸리데이터가 여러 식별자로 그 지분을 분할하려는 유혹을 받을 수 있습니다). 샤드체인 블록은 **2.6.7**에서 설명한대로 선정된 글로벌 밸리데이터 세트에서 가져온 특별하게 선택된 밸리데이터 서브세트에 의해서만 유효성이 검사됩니다.

모든 샤드에 대해 정의된 이러한 밸리데이터의 "서브세트" 또는 "작업그룹"은 매시간 (실제로는 매 2^{10} 개의 마스터체인 블록마다) 순환되며 한 시간 전에 미리 알려지므로 모든 밸리데이터는 유효성을 검사를 위해 어떤 샤드가 필요한지 알고 이를 위해 준비할 수 있습니다 (예. 누락된 샤드체인 데이터 다운로드).

각 샤드 (w, s) 에 대한 밸리데이터 작업그룹을 선택하는데 사용되는 알고리즘은 결정론적 의사 랜덤입니다. 랜덤(임의의) 시드를 생성하기 위해 밸리데이터가 각 마스터체인 블록에 포함된 의사난수 (예. 임계값 서명 [4] [10]을 사용하여 합의에 의해 생성된)를 사용한 다음 각 밸리데이터에 대해 HASH(CODE(w). CODE(s). *validator_id.rand_seed*)를 계산합니다. 그런 다음 밸리데이터는 이 해시의 값에 따라 정렬되고 처음 몇개가 선택되어 전체 밸리데이터 지분 중 적어도 20/T 를 가지며 적어도 5개의 밸리데이터로 구성됩니다.

이 선택은 특별한 스마트 컨트랙트에 의해 수행될 수 있습니다. 이 경우 선택 알고리즘은 **2.1.21**에서 언급한 투표 메커니즘에 의해 하드포크 없이 쉽게 업그레이드 될 수 있습니다. 지금까지 언급된 다른 모든 "상수" (예를 들어 2^{19} , 2^{10} , T , 20 및 5)도 구성 가능한 매개변수입니다.

[22] 밸리데이터 선출마다 새로운 키 페어를 생성하여 사용하는 것이 좋습니다.

2.6.9. 각 작업그룹에서 우선순위 자전 (Rotating priority order on each task group). 마스터체인 블록 및 (샤드체인) 블록 일련번호의 해시에 따라 특정 "우선순위"가 샤드 작업그룹의 구성원에게 부과됩니다. 이 순서는 위에서 설명한대로 일부 해시를 생성하고 정렬하여 결정됩니다.

새 샤드체인 블록을 생성해야 하는 경우, 이 블록을 생성하기 위해 선택된 샤드 작업그룹 밸리데이터는 일반적으로 이 자전하는 "우선순위"와 관련하여 첫 번째 입니다. 블록을 만들지 못하면 두 번째 또는 세 번째 밸리데이터가 이를 수행할 수 있습니다. 본질적으로 이들 모두가 블록 후보자를 추천할 수 있지만 가장 우선순위가 높은 밸리데이터가 제안한 후보는 BFT 합의 프로토콜의 결과로 승리해야 합니다.

2.6.10. 샤드체인 블록후보의 전파 (Propagation of shardchain block candidates). 샤드체인 작업그룹 멤버십은 1시간 전에 미리 알려지기 때문에 회원들은 TON 네트워크의 일반적인 메커니즘을 사용하여 전용 "샤드 밸리데이터 멀티캐스트 오버레이 네트워크"를 구축할 수 있습니다 (cf. **3.3**). 새로운 샤드체인 블록을 생성해야 하는 경우 - 일반적으로 가장 최근의 마스터체인 블록이 전파된 후 1 ~ 2 초가 지나면 - 누구나 다음 블록을 생성하는데 가장 높은 우선순위를 가진 사람을 알 수 있습니다 (cf. **2.6.9**). 이 밸리데이터는 자체적으로 또는 콜레이터의 도움으로 새로운 조합된 블록후보를 생성합니다 (cf. **2.6.5**). 밸리데이터는 이 블록후보를 검증해야 하며 (특히 일부 콜레이터가 준비한 경우) (밸리데이터) 개인 키로 서명해야 합니다. 그런 다음 블록후보는 사전 구성된 멀티캐스트 오버레이 네트워크를 사용하여 나머지 작업그룹에 전파됩니다 (작업 그룹은 **3.3**에서 설명한 대로 자신만의 프라이빗 오버레이 네트워크를 만들고 **3.3.15**에서 설명한 스트리밍 멀티캐스트 프로토콜 버전을 사용하여 블록후보를 전파합니다).

이 방법을 수행하는 진정한 BFT 방법는 허니뱃저BFT [18]에서 사용된 것과 같은 비잔틴 멀티캐스트 프로토콜을 사용하는 것입니다. 블록후보를 (N; $2N$ / 3)-easure 코드로 인코딩하고 결과 데이터 1 / N을 그룹의 각 구성원에게 직접 보내고 해당 부분을 그룹의 다른 모든 구성원에게 데이터의 일부를 직접 멀티캐스트 하도록 합니다.

그러나 이것을 수행하는 더 빠르고 간단한 방법 (cf. also **3.3.15**)은 블록후보를 부호가 있는 1-킬로바이트 블록 ("청크")의 순서로 분할하고 리드-솔로몬(Reed-Solomon) 또는 파운틴 코드를 사용하여 배열을 증가하고 (예를 들어, RaptorQ 코드 [15] [21] 또는 온라인 코드 [17]와 같은) 이 청크를 더 전파할 것으로 기대하며 이를 이러한 "멀티캐스트 메시" (즉, 오버레이 네트워크)에 있는 이웃에게 전파하기 시작합니다.

밸리데이터가 블록후보를 재구성하기에 충분한 청크를 확보하면 확인 영수증에 서명하고 이를 이웃을 통해 그룹전체로 전파합니다. 그런 다음 이웃 노드는 새로운 청크를 보내는 것을 멈추고, 이 노드가 자체적으로 리드-솔로몬 또는 파운틴 코드를 적용(필요한 모든 데이터가 있음)하여 후속 덩어리를 생성하고 서명과 결합하여 아직 아직 준비가되어 있지 않은 이웃 노드에게 전파할 수 있다고 믿습니다.

모든 "불량" 노드를 제거한 후에 "멀티캐스트 메쉬" (오버레이 네트워크)가 연결된 채로 있으면 (비잔틴 방식에서는 노드의 1/3까지는 잘못을 허용, 즉 임의의 악의적인 방식으로 행동함), 이 알고리즘은 가능한 빨리 블록후보를 전파할 것입니다.

지정된 우선순위가 높은 블록 생성자는 블록후보를 그룹 전체에 멀티캐스팅 할 수 있습니다. 우선순위에 따라 두 번째 및 세 번째 밸리데이터는 블록후보를 즉시 또는 우선순위 밸리데이터에서 블록후보를 받지 못한 채 멀티캐스팅하기 시작할 수 있습니다. 그러나 일반적으로 최대 우선순위가 있는 블록후보만이 모든 (실제로는 작업그룹의 3 분의 2 이상이) 밸리데이터에 의해 서명되고 새 샤드체인 블록으로 커밋됩니다.

2.6.11. 블록후보의 검증 (Validation of block candidates). 블록후보는 밸리데이터에 의해 수신되고 본래 밸리데이터의 서명이 확인되면 수신 밸리데이터는 이 블록후보의 모든 트랜잭션을 수행하고 그 결과가 주장된 것과 일치하는지 검사하여 유효성을 검증합니다. 다른 블록체인에서 가져온 모든 메시지는 조합된 데이터에서 적절한 머클증명을 통해 지원되지 않으면 블록후보가 유효하지 않은 것으로 간주됩니다 (그리고 이 증거가 마스터체인에 위임되면 이 블록후보와 이미 서명한 밸리데이터는 처벌될 수 있습니다). 반면에 블록후보가 유효하다면 수신 밸리데이터는 서명을 하고 "메쉬 멀티캐스트 네트워크" 또는 직접 네트워크 메시지를 통해 그룹의 다른 밸리데이터에 서명을 전파합니다.

우리는 밸리데이터는 (조합된) 블록후보의 유효성을 검사하기 위해 이 샤드체인 또는 인접 샤드체인의 상태에 액세스 할 필요가 없다는 점을 강조하고자 합니다.

[23] 머클 증명의 크기가 이 경우 금지될 수 있기 때문에 **2.4.21**에서 설명된 메시지 순서화 요구사항을 보장하기 위해 필요한 인접한 샤드체인의 출력 대기열 상태는 예외일 수 있습니다.

이렇게하면 유효성 검사가 매우 빠르게 (디스크 액세스 없이) 진행될 수 있으며, 밸리데이터의 계산 및 저장 부담이 줄어듭니다 (특히 블록후보를 만들기 위해 외부 조합자의 서비스를 기꺼이 받아들이는 경우).

2.6.12. 다음 블록후보의 선출 (Election of the next block candidate). 블록후보가 작업그룹에서 밸리데이터의 유효한 서명 중 최소한 3 분의 2 (지분의)를 수집하면 다음 샤드체인 블록으로 커밋될 자격이 있습니다. BFT 프로토콜은 선택된 블록후보 (하나 이상의 제안이 있을 수 있음)에 대한 합의를 달성하기 위해 실행되며 모든 "좋은" 밸리데이터는 이 라운드에서 우선순위가 가장 높은 블록후보를 선호합니다. 이 프로토콜을 실행한 결과 블록은 적어도 밸리데이터의 (지분의) 3 분의 2 이상의 서명으로 보강됩니다. 이 서명은 해당 블록의 유효성 뿐만 아니라 BFT 프로토콜에 의해 선출된 것에 대해서도 증언합니다. 그 후, 블록 (조합된 데이터 없이)은 이러한 서명과 결합되고 결정론적 방식으로 직렬화되며 네트워크를 통해 관련된 모든 당사자에게 전파됩니다.

2.6.13. 밸리데이터는 서명한 블록을 보관 (Validators must keep the blocks they have signed). (샤드체인 작업그룹 멤버쉽은) 작업그룹에 참여하는 동안, 그리고 적어도 1시간 (또는 차라리 2^{10} 블록) 후에 밸리데이터는 서명하고 커밋한 블록을 유지해야 합니다. 다른 밸리데이터에 서명된 블록을 제공하지 않으면 처벌될 수 있습니다.

2.6.14. 모든 밸리데이터에 새 샤드체인 블록의 헤더와 서명을 전파 (Propagating the headers and signatures of new shardchain blocks to all validators). 밸리데이터는 각 작업그룹에 대해 생성된 것과 유사한 멀티캐스트 메시 네트워크를 사용하여 새로 생성된 샤드체인 블록의 헤더와 서명을 글로벌 밸리데이터 세트에 전파합니다.

2.6.15. 새로운 마스터체인 블록의 생성 (Generation of new masterchain blocks). 모든 (또는 거의 모든) 새로운 샤드체인 블록이 생성된 후에는 새로운 마스터체인 블록이 생성될 수 있습니다. 절차는 기본적으로 샤드체인 블록 (cf. **2.6.12**)과 동일하지만 모든 밸리데이터 (또는 적어도 3 분의 2)가 이 프로세스에 참여해야 한다는 차이점이 있습니다. 새로운 샤드체인 블록의 헤더와 서명은 모든 밸리데이터로 전파되므로 각 샤드체인의 최신 블록의 해시는 새 마스터체인 블록에 포함될 수 있고 반드시 포함되어야 합니다. 이러한 해시가 마스터체인 블록에 투입되면 외부 관찰자 및 다른 샤드체인은 새 샤드체인 블록은 커밋되고 불변인 것으로 생각할 수 있습니다 (cf. **2.1.13**).

2.6.16. 밸리데이터는 마스터체인의 상태를 반드시 유지 (Validators must keep the state of masterchain). 마스터체인과 샤드체인 간의 주목할 만한 차이점은 모든 밸리데이터가 조합된 데이터에 의존하지 않고 마스터체인 상태를 추적할 것으로 예상된다는 점입니다.

이는 밸리데이터 작업그룹에 대한 지식은 마스터체인 상태에서 파생되므로 중요합니다.

2.6.17. 샤드체인 블록들은 병렬로 생성되고 전파 (Shardchain blocks are generated and propagated in parallel). 일반적으로 각 밸리데이터는 여러 샤드체인 작업그룹의 구성원입니다; 그들의 양 (따라서 밸리데이터의 부하)은 밸리데이터의 지분과 거의 비례합니다. 이는 밸리데이터가 새로운 샤드체인 블록 생성 프로토콜의 여러 인스턴스를 병렬로 실행함을 의미합니다.

2.6.18. 블록 보유 공격의 완화 (Mitigation of block retention attacks). 총 밸리데이터 세트는 헤더와 서명만 본 후에 마스터체인에 새 샤드체인 블록의 해시를 삽입하기 때문에 이 블록을 생성한 밸리데이터가 속임수를 쓰고 새 블록을 전체를 게시하지 않을 확률은 적습니다. 이것은 해시가 마스터체인으로 커밋된 후에는 새 블록의 출력 메시지 대기열을 최소한 알아야하기 때문에 인접한 샤드체인의 밸리데이터가 새 블록을 만들 수 없게 됩니다.

이것을 방지하기 위해 새로운 블록은 일부 밸리데이터에게서 (예. 인접 샤드체인의 작업그룹 조합의 3 분의 2) 서명을 반드시 수집하여 이 밸리데이터가 이 블록의 사본을 가지고 있으며 필요한 경우 다른 밸리데이터에게 보낼 의향이 있다는 것을 증언해야 합니다. 이러한 서명이 제시된 후에만 새로운 블록의 해시가 마스터체인에 포함될 수 있습니다.

2.6.19. 마스터체인 블록은 샤드체인 블록보다 나중에 생성. 마스터체인 블록은 샤드체인 블록과 마찬가지로 약 5 초마다 약 한 번 생성됩니다. 그러나 모든 샤드체인에서 새로운 블록을 생성하는 것은 본질적으로 동시에 실행되지만 (일반적으로 새로운 마스터체인 블록이 생성됨으로써), 마스터체인의 새로 생성된 샤드체인 블록의 해시를 포함시키기 위해서 새로운 마스터체인 블록의 생성은 의도적으로 지연됩니다.

2.6.20. 느린 밸리데이터에게 더 낮은 보상. 밸리데이터가 "느린" 경우, 새 블록후보를 검증하는데 실패할 수 있으며 새 블록을 커밋하는데 필요한 서명의 3 분의 2가 "느린" 밸리데이터의 참여없이 수집될 수 있습니다.

이 경우 이 블록과 관련된 보상의 더 낮은 부분을 받게 됩니다. 이는 밸리데이터가 하드웨어, 소프트웨어 및 네트워크 연결을 최적화하여 가능한 빨리 사용자 트랜잭션을 처리하기 위한 인센티브를 제공합니다.

그러나 밸리데이터가 커밋되기 전에 블록에 서명하지 않으면 해당 서명이 다음 블록 중 하나에 포함될 수 있으며 보상의 일부 (이후에 생성된 블록 수에 따라 기하급수적으로 감소합니다 - 예 : 밸리데이터가 k 블록 지연된 경우 0.9^k)는 이 밸리데이터에게 계속 제공됩니다.

2.6.21. 밸리데이터 서명의 "깊이". 일반적으로 밸리데이터가 블록에 서명을 하면 서명은 블록의 상대적 유효성만 증언합니다: 이 블록은 이 샤드체인 및 다른 샤드체인의 모든 이전 블록이 유효하다는 전제하에 유효합니다. 밸리데이터가 이전 블록으로 커밋된 잘못된 데이터를 당연시 했다는 이유로 처벌할 수 없습니다.

그러나 블록의 밸리데이터 서명에는 "깊이"라는 정수 매개변수가 있습니다. 만약 그것이 0 이 아니면, 이는 밸리데이터가 이 지정된 수의 이전 블록의 (상대적) 유효성도 주장한다는 의미입니다. 이는 "느린" 또는 "일시적으로 오프라인인" 밸리데이터가 서명없이 커밋된 일부 블록을 따라잡고 서명하는 방법입니다. 그러면 블록 보상의 일부가 여전히 그들에게 주어질 것입니다 (cf. **2.6.20**).

2.6.22. 밸리데이터는 서명된 샤드체인 블록의 상대적인 유효성을 책임집니다; 절대적인 타당성은 다음과 같습니다 **(Validators are responsible for relative validity of signed shardchain blocks; absolute validity follows).** 샤드체인 블록 B 에 대한 밸리데이터의 서명은 해당 블록의 상대적 유효성만 증언한다는 점을 다시 한 번 강조하고 싶습니다. (또는 서명에 "깊이" d 를 갖는다면 d 의 이전 블록일 수도 있습니다 cf. **2.6.21**; 하지만 이것은 다음 토론에 많은 영향을 주지 않습니다). 다시 말해, 밸리데이터는 **2.2.6**에서 설명한 블록 평가 함수 ev_block 을 적용하여 샤드체인의 다음 상태 s' 가 이전 상태 s 에서 얻어졌다고 주장합니다.

$$s' = ev\_block\,(B)(s) \tag{24}$$

이러한 방식으로, 블록 B 에 서명한 밸리데이터는 원래 상태가 "부정확" (예. 이전 블록들 중 하나의 무효 때문에)한 것으로 판명되면 처벌될 수 없습니다. 어부 (cf. **2.6.4**)는 상대적으로 무효한 블록을 발견한 경우에만 불평해야 합니다. PoS 시스템은 모든 블록을 재귀적으로 (또는 절대적으로) 유효하지 않고 상대적으로 유효하게 만들기 위해 노력합니다. 그러나 블록체인의 모든 블록이 상대적으로 유효하다면, 모든 블록과 블록체인 전체가 절대적으로 유효하다는 점에 유의하십시오.

이 문장은 블록체인 길이에 수학적 유도를 사용하면 쉽게 표시됩니다. 이러한 방식으로 블록의 상대적 유효성에 대한 검증이 가능한 검증을 함께하면 전체 블록체인의 절대적 유효성이 훨씬 강력해집니다.

블록 B 에 서명함으로써, 밸리데이터는 블록이 원래 상태 (즉, 수식 (24)의 결과가 다음 상태가 계산될 수 없다는 것을 나타내는 값 "⊥"가 아닌)가 주어졌을 때 유효성을 주장합니다. 이 방법으로, 밸리데이터는 (24)의 평가중에 액세스되는 원래 상태의 셀에 대해 최소한의 형식 검사를 수행해야 합니다.

예를 들어, 블록에 커밋된 트랜잭션에서 액세스한 계정의 원래 잔액을 포함할 것으로 예상되는 셀이 예상된 8 또는 16 대신 0 원시바이트가 아닌 것으로 판명되었다고 상상해보십시오. 그러면 원래 잔액을 셀에서 검색할 수 없으며 블록을 처리하려고 할 때 "처리되지 않은 예외"가 발생합니다. 이 경우, 밸리데이터는 처벌되는 고통으로 감수하고 그러한 블록에 서명해서는 안됩니다.

2.6.23. 마스터체인 블록 서명 (Signing masterchain blocks). 마스터체인 블록의 상황은 다소 다릅니다: 마스터체인 블록에 서명함으로써 밸리데이터는 상대적 유효성 뿐만 아니라 이 밸리데이터가 책임을 맡았던 맨 처음 블록부터 상대적 유효성을 나타냅니다 (그러나 더 이상 뒤로가 아닌).

2.6.24. 밸리데이터의 총 수 (The total number of validators). 선출된 밸리데이터의 총 수에 대한 상한선 T 는 (cf. **2.6.7**) 지금까지 설명한 시스템에서 말하자면 수백 또는 수천개가 될 수 없습니다. 왜냐하면 모든 밸리데이터가 마스터체인 블록을 만들기 위해 BFT 합의 프로토콜에 참여할 것으로 예상되기 때문에 이러한 프로토콜이 수 천명의 참가자에게 확장될 수 있는지에 대한 여부가 분명하지 않기 때문입니다. 더 중요한 것은, 마스터체인 블록은 모든 적어도 3 분의 2의 밸리데이터 서명을 (지분으로) 수집해야 하며, 이러한 서명은 새로운 블록에 반드시 포함되어야 합니다 (그렇지 않으면 시스템의 다른 모든 노드는 새로운 블록을 그들 스스 로 검증하지 않고 신뢰할 이유가 없습니다). 말하자면, 1천개 이상의 밸리데이터 서명이 각 마스터체인 블록에 포함되어야 한다면, 이는 모든 마스터체인 블록에 더 많은 데이터를 포함하고[24] 모든 풀-노드에 의해 저장되고 네트워크를 통해 전파되며, 이러한 서명을 확인하는데 더 많은 처리능력이 소비됩니다 (PoS 시스템에서, 풀-노드는 블록을 스스로 검증할 필요가 없지만, 대신에 밸리데이터의 서명을 검증해야 합니다).

[24] 다중-서명에 참여하는 밸리데이터의 서브세트를 표현하기 위해 비트맵만 필요하기 때문에 합성가능 다중-서명 (예. [4]의 페어링 기반)은 공간 요구사항을 어느 정도 완화할 수 있습니다. 서명은 하나의 일반 서명을 검증하는 것보다 훨씬 비용이 많이 들지 않습니다. 그러나, 그러한 구성 가능한 다중-서명에 대한 시간 및 공간 요구사항은 비록 더 작은 상수 일지라도 여전히 선형적입니다.

T 를 1000 명의 밸리데이터로 제한하는 것이 TON 블록체인 배포의 첫 번째 단계에서는 충분하다고 생각되지만, 샤드체인의 총 수가 너무 커져서 수백 명의 밸리데이터가 모두 처리하기에 충분하지 못할 때 미래의 성장을 위해 반드시 대비해야 합니다. 이를 위해 추가 구성 가능한 매개변수 $T' \leq T$ (원래는 T 와 같음)를 소개하고 맨 위에 있는 T' 선택된 밸리데이터 (지분 별)만 이 새로운 마스터체인 블록을 만들고 서명해야 합니다.

2.6.25. 시스템의 분산화 (Decentralization of the system). 모든 샤드체인과 마스터체인 블록을 생성하기 위해 $T \approx 1000$ 명의 밸리데이터에 의존하는 TON 블록체인과 같은 지분증명 시스템은 "너무 중앙집중화" 될 수 밖에 없다고 의심할 수도 있습니다. 비트코인이나 이더리움과 같은 기존의 작업증명 블록체인과는 달리 마이너의 총 수에 대한 명시적인 상한선 없이 (원칙적으로) 어디에서 누구나 새로운 블록을 채굴할 수 있습니다.

그러나 비트코인 및 이더리움과 같은 인기있는 작업증명 블록체인은 무시할 수 없는 성공확률로 새로운 블록을 채굴하기 위해 막대한 양의 컴퓨팅 성능(높은 "해시비율")을 필요로 합니다. 그래서, 새로운 블록의 채굴은 채굴에 최적화된 맞춤설계형 하드웨어로 가득찬 데이터 센터에 엄청난 돈을 투자하는 여러 대형 플레이어의 손에 집중되는 경향이 있고, 그 자체만으로는 충분한 "해시비율"을 스스로 제공할 수 없는 많은 사람들의 노력을 집중하고 조정하는 몇 개 대규모 마이닝-풀의 손 안에 있습니다.

따라서, 2017년 현재 새로운 이더리움 또는 비트코인 블록의 75% 이상이 10명 미만의 마이너에 의해 생산됩니다. 실제로 두 개의 가장 큰 이더리움 마이닝-풀은 모든 새로운 블록의 절반 이상을 함께 생산합니다! 분명히, 이러한 시스템은 새로운 블록을 생성하기 위해 $T \approx 1000$ 개의 노드에 의존하는 시스템보다 훨씬 중앙 집중화되어 있습니다.

TON 블록체인 밸리데이터가 되기 위해 필요한 투자 - 즉, 하드웨어 (예. 여러 고성능 서버)와 지분 (필요에 따라 후보자를 통해 쉽게 수집할 수 있는; cf. **2.6.3**) - 는 성공적인 독립형 비트코인 또는 이더리움 마이너가 되기 위해 필요한 것보다 훨씬 적습니다. 실제로 **2.6.7** 의 매개변수 L 은 지명된 사람이 가장 큰 "마이닝-풀" (즉, 가장 큰 지분을 축적한 밸리데이터)에 가입하지 않고, 현재 지명된 사람으로부터 자금을 수령하는 더 작은 밸리데이터를 찾도록 강제할 것이며, 밸리데이터의 비율을 높일 수 있고, 또한 지명된 사람의 지분을 사용할 수 있기 때문에 새 밸리데이터를 만들 수도 있습니다.

따라서, 채굴에서 더 많은 보상을 얻을 수 있습니다. 이런 방식으로, TON 지분증명 시스템은 실제로 분산화를 장려하고 (더 많은 밸리데이터 생성 및 사용) 중앙집중형을 처벌합니다 .

2.6.26. 블록의 상대적 신뢰도 (Relative reliability of a block). 블록의 (상대적) 신뢰도는 단순히 이 블록에 서명한 모든 밸리데이터의 총 지분입니다. 즉, 이 블록이 무효인 것으로 판명되면 특정 액터가 잃게 될 금액입니다. 블록의 신뢰도보다 낮은 가치를 전달하는 트랜잭션에 관심이 있다면 충분히 안전하다고 생각할 수 있습니다. 이러한 의미에서 상대적 신뢰도는 외부 관찰자가 특정 블록에서 가질 수 있는 신뢰의 척도입니다.

블록의 상대적 신뢰도에 대해 언급하는 이유는 이전 블록과 언급된 다른 모든 샤드체인 블록이 유효한 경우 블록이 유효하다는 보장이기 때문입니다 (cf. **2.6.22**).
블록의 상대적 신뢰도는 커밋된 후에 증가할 수 있습니다 - 예를 들어 뒤늦은 밸리데이터의 서명이 추가된 경우 (cf. **2.6.21**). 다른 한편으로는, 만일 어떤 밸리데이터 중 하나가 다른 블록과 관련된 부정 행위로 인해 지분의 일부 또는 전부를 잃는다면, 블록의 상대적인 신뢰도가 떨어질 수 있습니다.

2.6.27. 블록체인 "강화" (“Strengthening” the blockchain). 가능한 한 블록의 상대적 신뢰도를 높이기 위해 밸리데이터에 인센티브를 제공하는 것은 중요합니다. 이를 수행하는 한 가지 방법은 다른 샤드체인의 블록에 서명을 추가하는 밸리데이터에게 작은 보상을 할당하는 것입니다. 지분으로 최상위 T 밸리데이터에 들어가기에 불충분한 지분을 기탁하고 글로벌 밸리데이터 세트에 포함될 (cf. **2.6.7**) “될 수 있는” 밸리데이터도 이 활동에 참여할 수 있습니다 (만약 선출에 패한 후 지분을 철수하지 않고 동결하는 것에 그들이 동의하면). 그러한 “될 수 있는” 밸리데이터는 어부의 두배가 될 수 있습니다 (cf. **2.6.4**): 그들이 어차피 특정 블록의 유효성을 검사해야 하는 경우 차라리 잘못된 블록을 보고하고 관련 보상을 받을 수도 있습니다.

2.6.28. 블록의 재귀적 신뢰성 (Recursive reliability of a block). 블록의 재귀적 신뢰도를 상대적 신뢰도와 그것이 지칭하는 모든 블록의 재귀적 신뢰도의 최소값으로 정의할 수 있습니다 (즉, 마스터체인 블록, 이전의 샤드체인 블록 및 인접한 샤드체인의 일부 블록). 다른말로, 블록 자체가 유효하지 않거나 블록에 속한 블록 중 하나가 유효하지 않기 때문에 블록이 유효하지 않은 것으로 판명되면 적어도 이 금액은 누군가에 의해 손실됩니다.

블록에서 특정 트랜잭션을 신뢰할 것인지 여부를 확실하게 알 수 없다면 상대 블록 뿐만 아니라 이 블록의 재귀적 신뢰도도 계산해야 합니다.

재귀적 신뢰도를 계산할 때 지나치게 뒤로 돌아가는 것은 의미가 없습니다. 왜냐하면 우리가 너무 멀리 뒤sti돌아 보았을 때 이미 지분이 동결되지 않고 출금된 밸리데이터가 서명한 블록을 볼 것이기 때문입니다. 어느 상황이든 우리는 밸리데이터가 오래된 (즉, 구성 가능한 매개변수의 현재 값이 사용되는 경우 2개월 이상 전에 생성된) 블록을 자동으로 재검토하도록 허용하지 않으며 그것들이 잘못된 것으로 밝혀 지더라도 "수직 블록체인" (cf. **2.1.17**)의 도움으로 그것들에서 시작하는 포크를 만들거나 수정합니다. 2개월 이라는 기간은 유효하지 않은 블록을 발견하고 보고할 수 있는 충분한 기회를 제공하므로 이 기간 동안 블록에 대해 이의 제기를 하지 않으면 전혀 도전할 여지가 없다고 가정합니다.

2.6.29. 라이트 노드에 대한 지분증명의 결과 (Consequence of Proof-of-Stake for light nodes). TON 블록체인에서 사용되는 지분증명 접근법의 중요한 결과는 TON 블록체인을 위한 라이트 노드 (라이트 클라이언트 소프트웨어를 실행하는)가 그것의 질의에 대한 응답으로서 풀-노드에 의해 제공되는 머클증명의 유효성을 자체적으로 검사하기 위해 모든 샤드체인 또는 마스터체인 블록의 "헤더"를 다운로드할 필요가 없다는 것입니다.

실제로 가장 최근의 샤드체인 블록 해시가 마스터체인 블록에 포함되어 있기 때문에 풀-노드는 주어진 샤드체인 블록이 마스터체인 블록의 알려진 해시에서부터 유효하다는 머클증명을 쉽게 제공할 수 있습니다. 다음으로 라이트 노드는 클라이언트 소프트웨어에 기본 제공되는 (또는 적어도 해시가 포함될 수 있음) 마스터체인의 첫 번째 블록 (밸리데이터의 첫 번째 세트가 발표되는 곳)만 알고 있어야합니다. 이 블록은 이전 밸리데이터 세트에 의해 서명되었으므로 새로 선출된 밸리데이터 집합이 발표된 후 매월 1개의 마스터체인이 차단됩니다. 이를 시작으로 몇 가지 가장 최근의 마스터체인 블록, 또는 적어도 그의 헤더와 밸리데이터 서명을 얻을 수 있으며 풀-노드에서 제공하는 머클증명을 확인하기 위한 기반으로 사용할 수 있습니다.

2.7 샤드체인 분할 및 병합 (Splitting and Merging Shardchains)

TON 블록체인의 가장 독특하고 고유한 특징 중 하나는 부하가 너무 높아지면 자동으로 샤드체인을 2개로 분할하고 부하가 가라 앉으면 다시 병합할 수 있다는 것입니다 (cf. **2.1.10**). 우리는 이 특징의 고유성과 전체 프로젝트의 확장성에 대한 중요성 때문에 이 부분에 대해 좀 더 자세하게 논의해야 합니다.

2.7.1. 샤드 구성 (Shard configuration). 어떤 주어진 순간에, 각 워크체인 w 는 하나 또는 여러 개의 샤드체인 (w, s) 으로 분할되어 있습니다 (cf. **2.1.8**). 이 샤드체인은 자식 $(w, s.0)$ 및 $(w, s.1)$ 을 갖는 각 비-잎 노드(w, s) 와 루트(root) $(w, \emptyset)$를 갖은 이진 트리의 잎으로 나타낼 수 있습니다. 이렇게 하면 워크체인 w 에 속한 모든 계정이 정확히 하나의 샤드에 할당되고 현재 샤드체인 구성을 알고 있는 모든 사람이 *account_ id* 계정을 포함하는 샤드 (w, s) 를 결정할 수 있습니다. 이진 문자열 s 가 의 접두어로 사용되는 유일한 샤드입니다.

샤드 구성 - 즉, 이 샤드 이진 트리 또는 주어진 w (샤드 이진 트리의 잎에 해당)의 모든 활성 (w, s) 모음 - 은 마스터체인 상태의 일부이며 마스터체인을 추적하는 모든 사람이 사용할 수 있습니다.[25]

2.7.2. 가장 최근의 샤드 구성과 상태 (Most recent shard configuration and state). 가장 최근의 샤드체인 블록의 해시가 각 마스터체인 블록에 포함되어 있음을 상기하십시오.

이 해시는 샤드 이진 트리 (실제로는 각 워크체인에 대한 트리 모음)로 구성됩니다. 이 방법으로 각 마스터체인 블록에는 가장 최근의 샤드 구성이 포함됩니다.

2.7.3. 샤드 구성 변경사항 발표 및 수행 (Announcing and performing changes in the shard configuration). 샤드 구성은 두 가지 방법으로 변경할 수 있습니다: 샤드 (w, s) 는 두 개의 샤드 $(w, s.0)$ 와 $(w, s.1)$ 로 분할 되거나 또는 두 개의 "형제" 샤드 $(w, s.0)$와 $(w, s.1)$은 하나의 샤드 (w, s) 로 병합될 수 있습니다.

이 분할/병합 작업은 먼저 응답하는 샤드체인 블록의 "헤더"에서 여러 가지 (예. 2^6; 이것은 구성 가능한 매개변수) 블록으로 발표되고, 이어서 이 샤드체인 블록을 나타내는 마스터체인 블록에서 발표됩니다. 이 사전 발표는 모든 당사자가 계획된 변경을 준비하는데 필요합니다 .

[25] 사실 샤드 구성은 마지막 마스터체인 블록에 의해 완전히 결정됩니다. 이렇게 하면 샤드 구성에 쉽게 액세스할 수 있습니다.

(예**. 3.3** 에서 설명한대로 새로 만든 샤드체인의 새 블록을 배포하기 위해 오버레이 멀티캐스트 네트워크를 구축). 그런 다음 변경 사항이 커밋되고 먼저 샤드체인 블록(헤더로)으로 (분할의 경우 병합을 위해 두 샤드체인의 블록이 변경을 커밋해야 함), 그런 다음 마스터체인 블록으로 전파됩니다. 이런 방식으로, 마스터체인 블록은 생성전에 가장 최근의 샤드 구성 뿐만 아니라 바로 다음의 샤드 구성을 정의합니다.

2.7.4. 새 샤드체인에 대한 밸리데이터 작업그룹 (Validator task groups for new shard-chains). 각 샤드 즉, 각각의 샤드체인은 일반적으로 해당 샤드체인에서 새로운 블록을 생성하고 유효성을 검증하는데 사용되는 밸리데이터(밸리데이터 작업그룹)의 서브세트로 지정됩니다 (cf. **2.6.8**). 이 작업그룹은 일정 기간동안 (약 1시간) 선출되며 사전에 알 수 있으며 (또한 약 1시간) 이 시간 동안 불변입니다.[26]

그러나 실제 샤드 구성은 분할/병합 작업으로 인해 이 시간 동안 변경될 수 있습니다. 새로 생성된 샤드에 작업그룹을 할당해야 합니다. 이것은 다음과 같이 수행됩니다.

모든 활성 샤드 (w, s) 는 고유하게 결정된 오리지널 샤드 (w, s') 의 자손이거나 s' 가 s 의 접두사이거나 s 는 모든 s'의 접두어가 되는 오리지널 샤드의 서브트리의 루트가 됩니다. 첫번째 경우에는 간단하게 오리지널 샤드 (w, s') 의 작업그룹을 갖고 새 샤드 (w, s) 의 작업그룹으로 두 배로 만듭니다. 후자의 경우, 새 샤드 (w, s) 의 작업그룹은 샤드트리의 (w, s) 자손인 모든 오리지널 샤드 (w, s') 의 작업그룹의 합집합이 됩니다.

이러한 방식으로 모든 활성 샤드 (w, s) 에 밸리데이터 (작업그룹)의 잘 정의된 서브세트가 지정됩니다. 샤드가 분할되면 두 자식 모두 오리지널 샤드에서 작업그룹 전체를 상속받습니다. 두 개의 샤드가 병합되면 해당 작업그룹도 병합됩니다.

마스터체인 상태를 추적하고 있는 사람은 누구나 활성 샤드 각각에 대한 밸리데이터 작업그룹을 계산할 수 있습니다.

2.7.5. 원래 작업그룹의 책임기간 동안 분할/병합 작업 제한 (Limit on split/merge oper-ations during the period of responsibility of original task groups). 궁극적으로 새
샤드 구성이 고려되며 새 전용 밸리데이터 서브세트 (작업그룹)이 각 샤드에 자동으로 할당됩니다. 그런 일이 일어나기 전에 분할/병합 작업에 일정한 제한을 두어야 합니다.

[26] 유효하지 않은 블록에 서명하여 일부 밸리데이터가 일시적으로 또는 영구적으로 금지되지 않는 한 그들은 자동으로 모든 작업그룹에서 제외됩니다.

그렇지 않으면 원본 작업그룹이 2^k 개의 새 샤드로 빠르게 분할되는 경우 큰 k 에 대해 2^k 개의 샤드체인을 동시에 검증하게 됩니다.

이는 활성 샤드 구성이 현재 샤드 구성 (현재 담당중인 밸리데이터 작업그룹을 선택하는데 사용된 구성)에서 제거될 수 있는 범위를 제한함으로써 수행됩니다. 예를 들어, s' 가 s 의 전임자인 경우 (즉, s'가 이진 문자열의 접두사인 경우) 활성 샤드 (w, s) 에서 원래 샤드 (w, s')까지 샤드 트리의 거리가 3을 초과하지 않고 s' 가 s 의 후속 (즉, s 는 s' 의 접두사인 경우)이면 2를 초과하지 않을 것을 요구할 수도 있습니다. 그렇지 않으면 분할 또는 병합 작업이 허용되지 않습니다.

대략 말하면, 주어진 밸리데이터 작업그룹의 모음의 책임기간 동안 샤드가 분할 (예를 들어, 3 회)되거나 병합 (예를 들어, 2 회) 될 수 있는 횟수에 제한을 부과하는 것입니다. 그 외에도 병합이나 분할로 샤드가 생성된 후에는 일정 기간 (일부 블록 수를) 재구성 할 수 없습니다.

2.7.6. 분할작업의 필요성 결정 (Determining the necessity of split operations). 샤드체인에 대한 분할작업은 특정 형식조건 (예를 들어, 64개의 연속 블록에 대해 샤드체인 블록이 90% 이상 채워진 경우)에 의해 실행됩니다. 이러한 조건은 샤드체인 작업그룹에서 모니터링 합니다. 조건이 일치하는 경우 먼저 "분할 준비" 플래그가 새 샤드체인 블록의 헤더에 포함됩니다 (그리고 이 샤드체인 블록을 참조하는 마스터체인 블록으로 전파됨). 그런 다음 몇 개의 블록 이후에 "분할 커밋" 플래그가 샤드체인 블록의 헤더에 포함됩니다 (그리고 다음 마스터 체인 블록으로 전파됩니다).

2.7.7. 분할작업 수행 (Performing split operations). 샤드체인 (w, s) 의 블록 B 에 "분할 커밋" 플래그가 포함된 후 해당 샤드체인에 다음 블록 B'가 있을 수 없습니다. 대신에, 샤드체인 (w, s') 과 $(w, s.1)$ 의 두 블록 B'_0 와 B'_1 이 생성될 것이며, 두 블록 모두 B 블록을 이전 블록으로 참고합니다 (그리고 둘 다 헤더의 플래그로 샤드가 분할되어 있음을 표시합니다). 다음 마스터체인 블록에는 새 샤드체인의 블록 B'_0 와 B'_1 의 해시가 포함됩니다; "분할커밋" 이벤트가 이미 이전 마스터체인 블록에 커밋되었기 때문에 샤드체인 (w, s) 의 새 블록 B'의 해시를 포함할 수 없습니다.

두 개의 새로운 샤드체인은 이전 그룹과 동일한 밸리데이터 작업그룹에 의해 유효성이 검사되므로 자동으로 해당 상태의 사본을 갖게됩니다. 상태 분할연산 자체는 무한 샤딩 패러다임의 관점에서 보면 아주 간단합니다 (cf. **2.5.2**).

2.7.8. 병합 작업의 필요성 결정 (Determining the necessity of merge operations). 샤드 병합작업의 필요성은 또한 특정 공식조건들 (예. 64 개의 연속블록 동안 형제 샤드체인들의 두 블록들의 크기들의 합이 최대 블록 크기의 60%를 초과하지 않는 경우)에 의해 감지됩니다.

이러한 공식조건들은 또한 이 블록에 의해 소비된 총 가스량를 고려해야 하며 현재 블록의 가스 한도와 비교해야 합니다. 그렇지 않으면 더 많은 트랜잭션을 포함하지 못하게 하는 계산 집약적인 트랜잭션이 있기 때문에 블록이 작아질 수 있습니다. 이러한 조건들은 형제 샤드 $(w, s.0)$ 및 $(w, s.1)$ 의 밸리데이터 작업그룹에 의해 모니터됩니다. 형제는 하이퍼큐브 라우팅 (cf. **2.4.19**)과 관련하여 필연적으로 이웃이므로 모든 샤드의 작업그룹에 있는 밸리데이터는 형제 샤드를 어느 정도 모니터링 합니다.

이러한 조건들이 충족되면 밸리데이터 서브그룹 중 하나는 다른 그룹에게 특수한 메시지를 병합을 제안할 수 있습니다. 그런 다음 BFT 합의 알고리즘 실행하고 필요한 경우 블록 업데이트를 전파하고 후보를 차단할 수 있는 결합된 멤버십과 함께 잠정적인 "병합 작업그룹"으로 결합합니다.
그들이 병합의 필요성과 준비성에 대한 합의에 이르면 형제 작업그룹 최소 3 분의 2의 밸리데이터의 서명과 함께 각 샤드체인의 일부 블록 헤더에 "병합준비" 플래그가 커밋됩니다 (그리고 마스터체인 블록으로 전파되어 모든 사람이 임박한 재구성을 준비할 수 있습니다). 그러나 사전 정의된 수의 블록에 대해서는 별도의 샤드체인 블록을 계속 생성합니다.

2.7.9. 병합작업 수행 (Performing merge operations). 그 다음, 두 개의 본래 작업그룹의 합집합에 있는 밸리데이터가 병합된 샤드체인의 밸리데이터가 될 준비가 되면 (이 작업은 형제 샤드체인에서 상태전달과 상태 병합작업을 포함할 수 있음) 그들은 샤드체인 블록의 헤더에 "병합 커밋" 플래그를 커밋하고 (이 이벤트는 다음 마스터체인 블록으로 전파됩니다), 별도의 샤드체인에 있는 블록을 만드는 것을 중지합니다 (병합 커밋 플래그가 나타나면 별도의 샤드체인에 블록을 만드는 것을 금지합니다). 대신 병합된 샤드체인 블록이 (두 개의 본래 작업그룹의 합집합에 의해) 생성되어 "헤더"의 "앞선 블록"을 참조합니다. 이것은 병합된 샤드체인의 새로 생성된 블록의 해시를 포함하는 다음 마스터체인 블록에 반영됩니다. 그런 다음 병합된 작업그룹은 병합된 샤드체인에 블록을 계속 생성합니다.

2.8 블록체인 프로젝트의 분류 (Classification of Blockchain Projects)

TON 블록체인에 대한 간략한 논의는 기존 및 제안된 블록체인 프로젝트와 비교하여 마무리 할 것입니다. 그러나 이렇게 하기 전에 블록체인 프로젝트의 충분히 일반적인 분류를 소개해야 합니다. 이 분류를 기반으로 한 특정 블록체인 프로젝트의 비교는 **2.9** 까지 연기됩니다.

2.8.1. 블록체인 프로젝트의 분류 (Classification of blockchain projects). 첫 번째 단계로 블록체인(블록체인 프로젝트)에 대한 분류 기준을 제안합니다. 그러나 이러한 분류는 다소 불완전하고 피상적인데, 이는 고려중인 프로젝트의 가장 독특한 특징 중 일부를 무시해야 하기 때문입니다. 그러나 우리는 이것이 블록체인 프로젝트 영토에 대한 아주 대략적인 지도를 제공하는데 필요한 첫 걸음이라고 생각합니다.

우리가 고려하는 기준목록은 다음과 같습니다.

- 싱글-블록체인 vs. 멀티-블록체인 아키텍처 (cf. **2.8.2**)
- 합의 알고리즘: 지분증명 vs. 작업증명 (cf. **2.8.3**)
- 지분증명 시스템의 경우 정확한 블록 생성, 유효성 검증 및 합의 알고리즘이 사용됩니다. (두 가지 주요 옵션은 DPOS vs. BFT; cf. **2.8.4**)
- "임의적" (튜링 완전) 스마트 컨트랙트 지원 (cf. **2.8.6**)

멀티-블록체인 시스템은 추가 분류기준을 가집니다 (cf. **2.8.7**):

- 멤버 블록체인의 유형과 규칙: 동질적, 이질적 (cf. **2.8.8**), 혼합 (cf. **2.8.9**), 연합 (cf. **2.8.10**).
- 내부 또는 외부의 마스터체인 부재 또는 존재 (cf. **2.8.11**)
  샤딩에 대한 네이티브(native) 지원 (cf. **2.8.12**). 정적 또는 동적 샤딩 (cf. **2.8.13**).
  멤버 블록체인 간의 상호작용: 느슨하게 및 단단히 결합된 시스템 (cf. **2.8.14**)

2.8.2. 싱글-블록체인 vs. 멀티-블록체인 프로젝트 (Single-blockchain vs. multi-blockchain projects). 첫 번째 분류기준은 시스템의 블록체인 수량입니다.가장 오래되고 단순한 프로젝트는 싱글-블록체인으로 구성됩니다 (간단히 "싱글체인 프로젝트"). 보다 정교한 프로젝트는 멀티플 블록체인 ("멀티체인 프로젝트")을 사용합니다 (또는 정확히 말하면 사용 계획).

싱글체인 프로젝트는 일반적으로 더 간단하고 테스트하기 더 쉽습니다; 그들은 시간의 시험에 저항했습니다. 그들의 주요 단점은 낮은 성능 또는 적은 트랜잭션 처리량인데, 이는 범용 시스템에 대해 초당 10개 (비트코인)에서 초당 100개[27] (이더리움) 미만의 트랜잭션 수준입니다. 비트쉐어와 같은 일부 특수시스템은 초당 수 천개의 특수 트랜잭션을 처리할 수 있습니다. 블록체인 상태를 메모리에 맞추고 처리를 사전 정의된 특수 트랜잭션 세트로 변환한 다음 C++과 같은 언어로 작성된 고도로 최적화된 코드 (여기에는 VM이 없음)에 의해 실행됩니다.

멀티체인 프로젝트는 모두가 갈망하는 확장성을 약속합니다. 프로젝트를 훨씬 더 복잡하게 만들고 그 구현을 어렵게하는 대신, 더 큰 전체 상태와 초당 트랜잭션을 지원할 수 있습니다. 결과적으로, 이미 실행중인 멀티-프로젝트는 거의 없지만 그러나 대부분의 제안된 프로젝트는 멀티-체인입니다. 우리는 미래가 멀티-체인 프로젝트에 속한다고 믿습니다.

2.8.3. 블록 생성 및 검증 : 작업증명과 지분증명 (Creating and validating blocks:Proof-of-Work vs. Proof-of-Stake). 또 다른 중요한 차이점은 새로운 블록을 만들고 전파하고, 유효성을 확인하고, 나타날 경우 여러 포크 중 하나를 선택하는데 사용되는 알고리즘과 프로토콜입니다.

가장 보편적인 두가지 패러다임은 작업증명(*Pow*) 및 지분증명(*Pos*)입니다. 작업증명 접근법은 다른 경쟁자가 이 작업을 하기 전에 다르게 쓸모없는 계산상의 문제 (일반적으로 많은 양의 해시계산 포함)를 해결할 만큼 운이 좋은 경우 일반적으로 모든 노드가 새 블록 (해당 블록을 채굴하는 것과 관련된 보상을 얻음)을 생성 ("채굴") 할 수 있습니다. 포크의 경우 (예를 들어, 두 노드가 유효하지만 다른 두 블록을 게시하여 이전 블록을 따라 간다면) 가장 긴 포크가 이깁니다. 이런 방식으로 블록체인의 불변성 보장은블록체인을 생성하는데 소요되는 작업 량 (계산 리소스)에 기반합니다; 이 블록체인의포크를 만들고 싶은 사람은 이미 커밋된 블록의 대체 버전을 만들기 위해 이 작업을 다시해야 합니다.

[27] 당분간은 15 이상 입니다. 그러나 일부 업그레이드는 이더리움 트랜잭션 처리량을 몇 배로 늘릴 계획입니다.

이를 위해, 새 블록을 생성하는데 소비되는 총 컴퓨팅 파워의 50% 이상을 제어해야 합니다. 그렇지 않으면 대체 포크가 기하 급수적으로 가장 길어질 기회가 낮아집니다.

지분증명 접근법은 일부 특수노드 (밸리데이터)가 일부 블록을 검사 (유효성 검사)하고 올바른 것으로 판명했다고 주장하는 대용량 지분 (암호화폐에 지정된)을 기반으로 합니다. 밸리데이터는 블록에 서명하고 이에 대한 약간의 보상을 받습니다. 그러나 밸리데이터가 부정확한 블록에 서명을 하고 그 증거가 제시되면 지분의 일부 또는 전부가 상실됩니다. 이런식으로 블록체인의 유효성과 불변성에 대한 보증은 블록체인의 유효성에 대한 밸리데이터의 지분 총량에 의해 결정됩니다.

지분증명 접근법은 유효하지 않은 해시를 계산하는 대신 유용한 계산(특히 블록에 나열된 모든 트랜잭션을 수행하여 새 블록을 확인 또는 작성해야 함)을 수행하도록 (*PoW* 마이너를 대체하는) 밸리데이터에게 인센티브를 제공한다는 점에서 더 자연스럽습니다. 이런식으로, 밸리데이터는 이러한 트랜잭션과 관련된 보상을 받기 위해 사용자 트랜잭션을 처리하는데 더 적합한 하드웨어를 구매할 것이며 시스템 전체의 관점에서 볼 때 매우 유용한 투자로 보입니다.

그러나, 지분증명 시스템은 희귀하지만 가능한 많은 조건을 제공해야 하므로 구현하기가 다소 어려워집니다. 예를 들어, 일부 악성 밸리데이터는 시스템을 혼란에 빠뜨리면서 (예. 자체 암호화폐 잔고를 변경하여) 약간의 이익을 추출할 수 있습니다. 이것은 사소하지 않은 게임이론 문제를 일으킵니다.

요컨대, 지분증명은 더 자연스럽고 유망하며, 특히 멀티-체인 프로젝트의 경우 (많은 블록체인이 있을 경우 엄청난 양의 계산 리소스가 필요하기 때문에), 신중하게 생각하고 구현해야 합니다. 현재 실행중인 대부분의 블록체인 프로젝트, 특히 가장 오래된 프로젝트 (비트코인과 적어도 원래의 이더리움)는 작업증명을 사용합니다.

2.8.4. 지분증명의 변종. DPOS vs. BFT (Variants of Proof-of-Stake. DPOS vs. BFT). 작업증명 알고리즘은 서로 매우 유사하며 새 블록을 채굴하기 위해 계산해야 하는 해시함수가 대부분 다르지만, 지분증명 알고리즘에는 더 많은 가능성이 있습니다. 그들 자신의 하위 분류가 그들 장점입니다.

본질적으로, 지분증명 알고리즘에 관한 다음 질문에 답해야 합니다.

- 새로운 블록을 생성 (“채굴”) 할 수 있는 사람은 누구입니까? 풀-노드 입니까?또는 밸리데이터의 (상대적으로) 작은 서브세트의 구성원입니까? (대부분의지분증명 시스템에서는 여러 개의 지정된 밸리데이터 중 하나가 새 블록을 생성하고 서명해야 합니다.)
- 밸리데이터는 서명을 통해 블록의 유효성을 보장합니까? 아니면 모든 풀-노드가 모든 블록을 자체적으로 유효성을 검사할 것으로 예상합니까? (확장가능한지분증명 시스템은 모든 노드가 모든 블록체인의 모든 블록을 검증하도록 요구하는 대신 밸리데이터의 서명에 의존해야 합니다.)
- 다른 블록을 대신 생산할 수 없도록 사전에 알려진 다음 블록체인 블록의 지정생산자가 있습니까?
- 새로 생성된 블록은 원래 하나의 밸리데이터 (해당 생성자)만 서명 되었습니까? 아니면 처음부터 밸리데이터 서명의 대부분을 수집해야 합니까?

이 질문에 대한 답변에 따라 PoS 알고리즘의 가능한 클래스는 24 가지인 것처럼 보이지만 실제로는 PoS 에 대한 두 가지 주요 접근 방식에 따라 달라집니다. 실제로 확장 가능한 멀티체인 시스템에서 사용하도록 설계된 대부분의 최신 PoS 알고리즘은 처음 두 가지 질문에 동일한 방식으로 대답합니다; 밸리데이터만 새로운 블록을 생성할 수 있으며 모든 풀-노드가 모든 블록의 유효성을 자체적으로 검사하지 않고도 블록 유효성을 보장합니다.

2개의 마지막 질문에 관해서는, 그들의 대답은 상호 연관성이 높아지고 기본적으로 두 가지 기본 옵션만 남게됩니다.

- 위임지분증명 (*Delegated Proof-of-Stake, DPOS*): 모든 블록에 대해 널리알려진 지정 생성자가 있습니다; 그외 아무도 그 블록을 생산할 수 없습니다. 새블록은 원래 그것을 생성한 밸리데이터에 의해서만 서명됩니다.
- BFT (*Byzantine Fault Tolerant*) PoS 알고리즘: 밸리데이터의 알려진 서브세트가 있으며 그 중 하나가 새 블록을 제안할 수 있습니다; 다른 노드에공개되기전에 대다수의 밸리데이터가 유효성을 확인하고 서명해야 하는몇가지 제안후보자 중 실제로 다음 블록을 선택하는 것은 비잔틴 장애 허용합의 프로토콜버전에 의해 달성됩니다.

2.8.5. DPOS와 BFT PoS의 비교 (Comparison of DPOS and BFT PoS). BFT 접근법은 새로 생성된 블록의 유효성을 증명하는 대다수의 밸리데이터 서명을 맨 처음부터 가지고 있다는 이점이 있습니다

또 다른 장점은 대다수의 밸리데이터가 BFT 합의 프로토콜을 올바르게 실행하면 포크가 전혀 나타나지 않을 수 있다는 것입니다. 다른 한편으로, BFT 알고리즘은 상당히 복잡한 경향이 있으며 밸리데이터의 서브세트가 합의에 도달하는데 더 많은 시간이 필요합니다. 따라서 블록을 너무 자주 생성할 수 없습니다. 이것이 우리가 TON 블록체인 (이 분류 관 초당 한 번 또는 매초 마다[28] 한 번씩 생성할 수 있습니다.

DPOS 알고리즘은 매우 간단하고 직관적이라는 장점이 있습니다. 사전에 알려진 지정된 블록 생성자에 의존하기 때문에 새로운 블록을 말하자면 2초당 한 번 또는 매초 마다[28] 한 번씩 생성할 수 있습니다.

그러나, DPOS는 모든 노드 (또는 적어도 모든 밸리데이터)가 수신된 모든 블록의 유효성을 검증해야 합니다. 왜냐하면 새로운 블록을 생성하고 서명하는 밸리데이터가 이 블록의 상대적 유효성 뿐만 아니라 그것이 참고하는 이전 블록과 체인의 예전 블록(아마도 밸리데이터의 현재 서브세트의 책임 기간의 시작점)까지 검사하기 때문입니다. 밸리데이터의 현재 서브세트에는 미리 지정된 순서가 있으므로 각 블록마다 지정된 생성자 (즉, 해당 블록을 생성할 것으로 예상되는 밸리데이터)가 있습니다; 이러한 지정된 생성자는 라운드-로빈 방식으로 교대됩니다. 이런 방식으로, 블록은 생성자 밸리데이터에 의해서만 처음 서명됩니다. 그 다음 블록이 채굴될 때 그 생산자는 이전 블록 중 하나가 아닌 이 블록을 참조하기로 선택하면 (그렇지 않으면 블록이 더 짧은 체인에 놓여 미래에 "가장 긴 포크" 경쟁을 상실하게 됩니다), 다음 블록의 서명은 본질적으로 이전 블록상의 추가 서명입니다. 이런 방식으로, 새로운 블록은 점차적으로 더 많은 밸리데이터의 서명 예를 들어, 다음 20개의 블록을 생성하는데 필요한 시간 내에 20개의 서명을 수집합니다. 풀-노드는 이러한 20개의 서명을 기다리거나 또는 충분하게 확인된 블록 (예. 20개의 블록)에서 시작하여 자체적으로 블록의 유효성을 검사해야 합니다. 이것은 그렇게 쉬운 일은 아닙니다.

[28] 일부 사람들은 DPOS가 0.5 초의 블록 생성 시간을 주장하기도 합니다. 하지만 밸리데이터가 여러 대륙에 흩어져있는 경우 이것은 현실적이지 않습니다.

DPOS 알고리즘의 명백한 단점은 이 수준의 신뢰 (예: 20개의 서명)를 즉시 제공하는 BFT 알고리즘에 비해 새로운 블록 (그리고 그것에 커밋된 트랜잭션)이 20개의 더 많은 블록을 채굴한 후에만 동일한 수준의 신뢰 (**2.6.28** 에서 논의된 "재귀적 신뢰성")를 달성한다는 것입니다. 또 다른 단점은 DPOS 가 다른 포크로 전환할 때 "가장 긴 포크가 이긴다" 접근법을 사용한다는 것입니다. 이는 적어도 일부 생산자가 우리가 관심이 있는 블록 이후에 다음블록을 생성하지 못하면 (또는 네트워크 파티션이나 정교한 공격으로 인해 이러한 블록을 관찰하지 못하는 경우) 포크를 만들 가능성이 큽니다.

우리는 BFT 접근법이 DPOS 보다 구현하기가 더 정교하고 블록간에 더 긴 시간 간격을 요구하는 반면, "단단히 결합된" (cf. **2.8.14**) 멀티체인 시스템에 더 적합하다고 믿습니다. 왜냐하면 20번의 유효성 검증 (즉, 다음 20개의 블록)을 기다리지 않고 또 다음 6개의 블록이 포크가 나타나지 않고 새로운 블록을 그들 스스로 검증 하는지를 기다리지 않고 새로운 블록에서 커밋된 트랜잭션 (예. 그들을 위한 메시지 생성)을 보고 난 후 다른 블록체인이 거의 즉시 작동을 시작할 수 있기 때문입니다 (확장 가능한 멀티체인 시스템에서는 다른 블록체인의 블록 확인이 금지될 수 있음). 따라서 높은 안정성과 가용성을 유지하면서 확장성을 달성할 수 있습니다 (cf. **2.8.12**).

다른 한편, DPOS는 블록체인 간의 빠른 상호작용이 필요하지 않은 "느슨하게 결합된" 멀티체인 시스템에 적합한 선택일 수 있습니다. 예를 들어, 각 블록체인 ("워크체인")이 개별 분산형 교환 및 블록체인 간 상호작용은 드물게 한 워크체인에서 다른 워크체인으로 토큰을 전송하는 경우에만 제한됩니다 (오히려 한 워크체인에 있는 한 알트코인을 다른 워크체인에 1:1로 근접한 비율로 거래). 이것은 DPOS를 사용하는 BitShares 프로젝트에서 실제로 수행되는 것입니다.

요약하자면, DPOS는 새로운 블록을 생성할 수 있고 (블록간 간격이 작은) 신속하게 트랜잭션을 포함할 수 있지만 이러한 트랜잭션은 다른 블록체인 및 오프-체인 애플리케이션에서 "커밋된" 및 "변경 불가능한" BFT시스템보다 훨씬 느리게, 말하자면, 5초가 아닌 30초[29]가 걸립니다. 빠른 트랜잭션 포함은 더 빠른 트랜잭션 약속을 의미하지 않습니다. 빠른 블록간 상호작용이 필요한 경우 큰 문제가 될 수 있습니다. 이 경우DPOS를 포기하고 대신 BFT PoS를 선택해야 합니다.

[29] 예를 들어, 현재까지 제안된 최고의 DPOS 프로젝트 중 하나인 EOS는 45초간의 확인 및 블록간 상호작용 지연을 약속합니다 (cf. [8], "트랜잭션 확인" 및 "인터-체인 커뮤니케이션 대기시간" 섹션).

2.8.6. 트랜잭션 (즉, 본질적으로 임의의 스마트 컨트랙트)에서 튜링-완전 코드 지원 (Support for Turing-complete code in transactions, i.e., essentially arbitrary smart contracts). 블록체인 프로젝트는 일반적으로 블록에서 일부 트랜잭션을 수집하여 유용한 것으로 간주되는 방식으로 블록체인 상태를 변경합니다 (예. 한 계정에서 다른 계정으로 일정량의 암호화폐를 전송). 일부 블록체인 프로젝트는 사전 정의된 특정유형의 트랜잭션 (예: 올바른 서명이 제시된 경우 한 계정에서 다른 계정으로의 가치 이전)만 허용할 수 있습니다. 다른 사람들은 트랜잭션에서 제한된 형식의 스크립팅을 지원할 수 있습니다.

마지막으로, 일부 블록체인은 트랜잭션에서 임의로 복잡한 코드 실행을 지원하므로 시스템의 성능이 허용하는 한, 시스템이 (적어도 원칙적으로) 임의의 애플리케이션을 지원할 수 있습니다. 이것은 일반적으로 "튜링-완전 가상머신 및 스크립팅 언어" (다른 컴퓨팅 언어로 작성될 수 있는 프로그램은 블록체인 내에서 수행되도록 다시 작성될 수 있음을 의미함) 및 "스마트 컨트랙트" (블록체인에 상주하는 프로그램)와 관련되어 있습니다.

물론, 임의의 스마트 컨트랙트를 지원하면 시스템이 진정으로 유연해집니다. 반면에, 이러한 유연성은 비용이 발생합니다. 이러한 스마트 컨트랙트의 코드는 일부 가상 머신에서 실행되어야 하며, 누군가 블록을 작성하거나 유효성을 검증하려는 경우 블록의 각 트랜잭션마다 매번 이를 수행해야 합니다. 이렇게하면 미리 정의되고 불변인 단순 트랜잭션 유형의 경우와 비교하여 시스템 성능이 저하되며, 일부 가상 시스템 대신 C ++와 같은 언어로 처리들어져야만 했던 주요 이유는 비트코인 블록체인에서 스마트 컨트랙트 지원이 부족했기 때문입니다.

멀티-체인 시스템 (이기종. cf. **2.8.8**)에서는 일부 블록체인 (즉, 워크체인)에서 튜링-완전 스마트 컨트랙트를 지원하고 다른 일부에서는 사전 정의된 작은 최적화된 트랜잭션 세트를 지원함으로써 "두 세계의 장점”을 모두 지니고 있습니다.

2.8.7. 멀티체인 시스템의 분류. 지금까지의 분류는 싱글-체인 및 멀티-체인 시스템 모두에 유효했습니다. 그러나, 멀티-체인 시스템은 시스템의 다른 블록체인 간의 관계를 반영하여 여러가지 분류기준을 허용합니다. 이제 우리는 이러한 기준을 논의합니다.

2.8.8. 블록체인 유형: 동종 및 이기종 시스템. 멀티-체인 시스템에서 모든 블록체인은 본질적으로 동일한 유형이며 동일한 규칙을 가질 수 있습니다 (예. 똑같은 형식의 트랜잭션 사용, 스마트 컨트랙트 코드 실행을 위한 동일한 가상시스템, 동일한 암호화폐 공유 등). 이 유사점은 명시적으로 활용되지만 각 블록체인마다 다른 데이터가 사용됩니다. 이 경우, 우리는 그 시스템이 동질적이라고 말합니다. 그렇지 않으면, 다른 블록체인 (이 경우 보통 워크체인이라고 부름)은 서로 다른 "규칙"을 가질 수 있습니다. 우리는 그 시스템을 이기종이라고 합니다.

2.8.9. 혼합된 이기종 - 동종 시스템. 때로는 블록체인에 대한 유형이나 규칙 집합이 여러개 있는 혼합시스템이 있지만 동일한 규칙을 가진 많은 블록체인이 있으며 이 사실을 명시적으로 활용합니다. 이것은 이기종-동종 혼합 시스템 입니다. 우리가 아는 한, TON 블록체인은 그러한 시스템의 유일한 예입니다.

2.8.10. 동일한 규칙 또는 연합 을 갖는 여러 작업 체인이 있는 이기종 시스템 (Heterogeneous systems with several workchains having the same rules, or confederations). 경우에 따라 동일한 규칙을 사용하는 여러 블록체인 (작업체인)이 이기종 시스템에 있을 수 있지만 상호작용은 규칙이 다른 블록체인과 동일합니다 (즉, 그들의 유사성이 명시적으로 악용되지 않음). 그들이 "같은" 암호화폐를사용하는 것으로 보일지라도, 사실 그들은 다른 "알트코인" (암호화폐의 독립적인화신)을 사용합니다. 때로는 이러한 알트코인을 1:1에 가까운 비율로 변환할 수있는 특정 메커니즘을 사용할 수도 있습니다. 그러나 이것이 시스템을 동질적으로 만드는 것은 아닙니다. 그것은 이기종으로 남아 있습니다. 우리는 같은 규칙을 가진 이런 종 류의 워크체인 모음을 연합이라고 말합니다.

이질적 시스템을 도입하여 동일한 규칙 (즉, 연합)을 가진 여러 워크체인을 만드는 것은 확장 가능한 시스템을 구축하는 저렴한 방법처럼 보일 수 있지만 이 접근법에도 많은 단점이 있습니다. 본질적으로 누군가가 동일한 규칙을 가진 많은 워크체인에서 대규모 프로젝트를 호스트 한다면 그는 큰 프로젝트가 아닌 이 프로젝트의 작은 사례를 많이 얻습니다. 이는 어떤 채팅 (또는 한 게임) 방에서 최대 50명의 회원을 보유할 수 있는 채팅 애플리케이션 (또는 게임)과 같지만, 필요한 경우 더 많은 사용자를 수용할 수 있는 새 방을 만들어 "확장"합니다. 결과적으로, 많은 사용자가 채팅이나 게임에 참여할 수 있지만, 우리는 이러한 시스템이 진정으로 확장성이 있다고 말할 수 있습니까?

2.8.11. 외부 또는 내부 마스터체인의 존재 (Presence of a masterchain, external or internal). 때로는 멀티-체인 프로젝트에는 예를 들어, 시스템의 전체 구성 (모든 활성 블록체인 세트, 또는 오히려 워크체인), 현재 밸리데이터 세트 (지분증명 시스템용) 등을 저장하는데 사용되는 차별화된"마스터체인" (때로는 "컨트롤 블록체인"이라고 함)이 있습니다. 때로는 다른 블록체인이 최신 블록의 해시를 적용하여 마스터체인에 묶여있습니다 (이것은 TON 블록체인도 마찬가지 입니다).

경우에 따라 마스터체인은 외부에 있습니다. 즉, 프로젝트의 일부가 아니라 기존 프로젝트 블록체인의 사용과 관련이 없고 기존의 블록체인과는 완전히 별개의 블록체인입니다. 예를 들어, 이더리움 블록체인을 외부 프로젝트의 마스터체인으로 사용하고 이러한 목적으로 이더리움 블록체인에 특수한 스마트 컨트랙트를 게시할 수 있습니다 (예. 밸리데이터 선출 및 처벌).

2.8.12. 샤딩 지원 (Sharding support). 일부 블록체인 프로젝트 (또는 시스템)는 샤딩을 기본적으로 지원하므로 여러 (필연적으로 동종; cf. **2.8.8**) 블록체인은 하나의 (고급의 관점에서) 가상 블록체인에 대한 샤드로 간주됩니다. 예를 들어, 256개의 샤드 블록체인 ("샤드체인")을 동일한 규칙으로 생성하고 의 첫 번째 바이트에 따라 선택된 하나의 샤드에서 계정의 상태를 유지할 수 있습니다.

샤딩은 블록체인 시스템을 확장하는 자연스러운 접근 방식입니다. 제대로 구현되면 시스템의 사용자 및 스마트 컨트랙트가 샤딩의 존재를 전혀 알 필요가 없기 때문입니다. 실제로, 로드가 너무 높아지면 기존 싱글-체인 프로젝트 (예. 이더리움)에 샤딩을 추가하려고 합니다. 확장성에 대한 또 다른 접근법은 **2.8.10** 에 설명된 것과 같이 이기종 워크체인의 "연합"을 사용하여 각 사용자는 자신이 선택한 하나 또는 여러 개의 워크체인에 계정을 유지하고 필요할 때 한 워크체인의 계정에서 다른 워크체인으로 자금을 이체하여 1:1 알트코인 교환작업을 수행하는 것입니다. 이 접근법의 단점은 이미 **2.8.10** 에서 논의되었습니다.

그러나, 샤딩은 다른 샤드체인 간에 많은 메시지를 포함하기 때문에 빠르고 안정적으로 구현하기가 쉽지 않습니다. 예를들어, 계정들이 N 개의 샤드 간에 균등하게 분배되고 트랜잭션이 오로지 하나의 계정에서 다른 계정으로 단순한 자금 이체인 경우 모든 트랜잭션 중 작은 부분 $(1/N)$ 만 싱글 블록체인 내에서 수행됩니다; 거의 모든 $(1 - 1/N)$ 트랜잭션에는 내부-블록체인 간 통신을 필요로 하는 두 개의 블록체인이 관련됩니다.

시스템이 필요합니다. 다시 말해, 블록체인 프로젝트는 **2.8.14**에서 설명한 의미로 "단단히 결합" 되어야 합니다.

2.8.13. 동적 샤딩과 정적 샤딩 (Dynamic and static sharding). 샤딩은 동적 (필요한 경우 추가 샤드가 자동으로 생성되는 경우)일 수도 있고 또는 정적 (하드포크를 통해서만 변경할 수 있는 미리 정의된 개수의 샤드가 있는 경우) 일 수도 있습니다. 대부분의 샤딩 제안은 정적입니다. TON 블록체인은 동적샤딩을 사용합니다 (cf. **2.7**).

2.8.14. 블록체인 간의 상호 작용: "느슨하게-결합된" 시스템과 "단단히-결합된"시스템 (Interaction between blockchains: loosely-coupled and tightly-coupled systems). 멀티-블록체인 프로젝트는 구성 블록체인 간에 지원되는 상호작용 수준에 따라 분류할 수 있습니다.

지원 수준이 가장 낮으면 서로 다른 블록체인 간의 상호작용이 없습니다. 우리가 이 경우를 고려하지 않는 이유는 이들 블록체인은 하나의 블록체인 시스템의 일부가 아니라 동일한 블록체인 프로토콜의 인스턴스를 분리한 것이라고 말하기 때문입니다.

다음 수준의 지원은 원칙적으로 상호작용을 가능하게 하지만 블록체인 간의 메시징에 대한 특정 지원이 부족합니다, 그러나 어색합니다. 이러한 시스템을 "느슨하게-결합된" 시스템이라고 부릅니다. 이 시스템에서는 완전히 분리된 블록체인 프로젝트에 속한 블록체인이었던 것처럼 블록체인간에 메시지를 보내고 값을 전송해야 합니다 (예. 비트코인 및 이더리움; 두 당사자가 비트코인 블록체인에 보관된 일부 비트코인을 이더리움 블록체인에 보관된 이더로 교환하려고 하는 경우). 다른 말로, 소스 블록체인 블록에 아웃바운드 메시지 (또는 그 생성 트랜잭션)를 포함시켜야 합니다. 그런 다음 그녀 또는 다른 당사자는 원래 트랜잭션이 "커밋된" 및 "불변의"인 것으로 간주하여 해당 존재를 기반으로 외부 조치를 수행할 수 있도록 충분한 확인 (예를 들어, 주어진 수의 후속 블록들)을 기다려야 합니다. 그런 다음에만 메시지를 대상 블록체인으로 중계하는 트랜잭션 (아마도 원래 트랜잭션에 대한 참조 및 머클증명과 함께)을 커밋할 수 있습니다.

메시지를 전송하기 전에 충분히 기다리지 않거나 다른 이유로 포크가 발생하면 두 블록체인의 결합된 상태가 일관성이 없는 것으로 판명됩니다: 메시지는 첫 번째 블록체인에서 생성된 적이 없는 (궁극적으로 선택된 포크의) 두 번째 블록체인으로 전달됩니다.

때로는 메시징에 대한 부분적인 지원이 모든 워크체인 블록에서 메시지 형식과 입출력 메시지 대기열의 위치를 표준화하여 추가됩니다 (이는 이기종 시스템에서 특히 유용합니다). 이렇게하면 메시징이 어느 정도까지 용이해지지만 개념적으로 이전 사례와 크게 다르지 않으므로 이러한 시스템은 여전히 "느슨하게- 결합된" 상태입니다.

대조적으로, "단단히-결합된" 시스템에는 모든 블록체인 간에 빠른 메시징을 제공하는 특별한 메커니즘이 포함됩니다. 바람직한 행동은 기존 블록체인 블록에서 메시지가 생성된 직후 다른 워크체인으로 메시지를 전달할 수 있도록 하는 것입니다. 반면에, "단단히-결합된" 시스템은 포크의 경우 전반적인 일관성을 유지할 것으로 예상됩니다. 이 두 가지 요구사항은 언뜻보기에 모순되는 것처럼 보이지만 TON 블록체인에서 사용되는 메커니즘 (샤드체인 블록 해시를 마스터체인 블록에 포함; 유효하지 않은 블록을 수정하기 위해 "수직" 블록체인 사용, cf. **2.1.17**; 하이퍼큐브 라우팅, cf. **2.4.19**; 인스턴트 하이퍼큐브 라우팅, cf. **2.4.20**)은 아마 지금까지 단 하나의 시스템인 "단단히-결합된" 시스템이 될 수 있습니다.

물론 "느슨하게-결합된" 시스템을 구축하는 것이 훨씬 간단합니다. 그러나 신속하고 효율적인 샤딩 (cf. **2.8.12**)은 시스템이 "단단히 결합" 되어야 합니다.

2.8.15. 단순화된 분류. 블럭체인 프로젝트의 세대 (Simplified classification. Generations of blockchain projects). 지금까지 우리가 제안한 분류법은 모든 블록체인 프로젝트를 많은 수의 분류로 나눕니다. 그러나 우리가 사용하는 분류 기준은 실제로 상관 관계가 있습니다. 이를 통해 블록체인 프로젝트의 분류에 대한 단순화된 "세대적" 접근법을 제안할 수 있습니다. 아직 구현 및 배포되지 않은 프로젝트는 기울임 꼴로 표시됩니다; 세대의 가장 중요한 특성은 굵게 표시됩니다.

- 1 세대 : 싱글체인, **PoW**, 스마트 컨트랙트를 지원하지 않습니다. 예: 비트코인 (2009) 및 기타 흥미롭지 않은 모방자 (Litecoin, Monero, ...).
- 2 세대 : 싱글체인, PoW, 스마트 컨트랙트 지원. 예: 원래 형태의 이더리움(2013년, 2015년 배포)

- 3 세대 : 싱글체인, **PoS**, 스마트 컨트랙트를 지원. 예: 미래의 이더리움 (2018년 또는 그 이후).
- 대안 3' 세대 : 멀티체인, PoS, 느슨하게 결합됨. 스마트 컨트랙트 지원하지않은. 예: 비트쉐어(2013-2014, DPOS 사용).
- 4 세대 : 멀티체인, **PoS**, 스마트 컨트랙트 지원. 느슨하게 결합됨 예: EOS (2017; DPOS 사용), 폴카닷 (2016; BFT 사용).
- 5 세대 : 멀티체인, BFT가 있는 PoS, 스마트 계약 지원, 단단히 결합됨, 샤딩. 예: *TON* (2017).

2.8.16. 블록체인 프로젝트의 "게놈" 변경의 문제 (Complications of changing the "genome" of a blockchain project). 위의 분류는 블록체인 프로젝트의 "게놈"을 정의합니다. 이 게놈은 매우 "엄격"합니다. 일단 프로젝트가 배포되고 많은 사람들이 사용하면 그것을 변경하는 것은 거의 불가능합니다. 하나는 일련의 하드 포크(대다수 커뮤니티의 승인을 필요로 함)가 필요할 것이며, 심지어는 이전 버전과의호환성을 유지하기 위해서는 그 변경은 매우 보수적일 필요가 있습니다 (예. 가상머신의 의미를 변경하면 기존의 스마트 컨트랙트이 손상될 수 있습니다). 다른 방법은서로 다른 규칙으로 새로운 "사이드체인"을 만들고 원래 프로젝트의 블록체인 (또는 블록체인)에 어떻게든 묶는 것입니다. 기존의 싱글-블록체인 프로젝트의 블록체인을 본질적으로 새롭고 개별적인 프로젝트를 위한 외부 마스터체인으로 사용할 수 있습니다. [30]

우리의 결론은 일단 배포되면 프로젝트의 게놈을 변경하기가 매우 어렵다는 것입니다. PoW로 시작하여 향후 PoS로 교체하려는 계획은 매우 복잡합니다.[31] 원래 지원하지 않고 설계된 프로젝트에 샤드를 추가하는 것은 거의 불가능한 것처럼 보입니다.[32] 사실, 그러한 기능을 지원하지 않고 원래 설계된 프로젝트 (즉, 비트코인)에 스마트 컨트랙트에 대한 지원을 추가하는 것은 불가능한 것으로 간주되어 (또는 적어도 비트코인 커뮤니티의 대다수에게는 바람직하지 않아) 결국 새로운 블록체인 프로젝트인 이더리움이 만들어지게 되었습니다.

[30] 예를 들어, 플라즈마 프로젝트는 이더리움 블록체인을 (외부) 마스터체인으로 사용할 계획입니다. 그렇지 않으면 이더리움과 많이 상호작용하지 않으며 이더리움 프로젝트와 무관한 팀에 의해 제안 및 구현될 수 있습니다.

[31] 이더리움은 2017년 현재 PoW 에서 PoW+PoS 시스템으로 전환하는데 어려움을 겪고 있습니다. 언젠가 진정한 PoS 시스템이 되기를 바랍니다.

[32] 이더리움에 대한 샤딩 제안은 2015년으로 거슬러 올라갑니다. 이더리움을 방해하거나 본질적으로 독립된 병렬 프로젝트를 만들지 않고 구현 및 배포할 수 있는 방법이 명확하지 않습니다.

2.8.17. TON 블록체인의 게놈 (Genome of the TON Blockchain). 따라서 확장가능한 블록체인 시스템을 구축하려면 처음부터 게놈을 신중하게 선택해야 합니다. 시스템이 향후 배포시 알려지지 않은 일부 추가기능을 지원하기 위한 것이라면 처음부터 "이기종" 워크체인 (잠재적으로 다른 규칙을 가짐)을 지원해야 합니다. 시스템이 진정으로 확장 가능하려면 처음부터 샤딩을 지원해야 합니다; 샤딩은 시스템이 "단단히 결합된" 경우에만 의미가 있습니다 (cf. **2.8.14**). 이것은 차례로 마스터체인의 존재, 블록간메시징의 빠른 시스템, BFT PoS 사용 등을 의미합니다.

이러한 모든 함의를 고려할 때, TON 블록체인 프로젝분의 설계 선택은 자연스럽게 보일뿐 아니라 거의 유일한 것입니다.

2.9 다른 블록체인 프로젝트와의 비교 (Comparison to Other Blockchain Projects)

TON 블록체인에 대한 간략한 토론과 기존 및 제안된 블록체인 프로젝트가 포함된 지도에서 TON 블록체인의 가장 중요한 고유기능에 대해 간략히 논의합니다. 우리는 **2.8**에서 설명한 분류기준을 사용하여 다른 블록체인 프로젝트를 일관된 방식으로 논의하고 그러한 "블록체인 프로젝트 맵"을 구성합니다. 우리는 이 지도를 표1로 표현한 다음 몇 가지 프로젝트를 간단히 토론하여 일반 분류에 맞지 않는 특성을 짚어보겠습니다.

2.9.1. 비트코인 [19]; https://bitcoin.org/. 비트코인 (2009)은 최초이자 가장 유명한 블록체인 프로젝트입니다. 이것은 전형적인 1세대 블록체인 프로젝트입니다: 싱글-체인이며 "가장 긴 포크가 이긴다" 포크 선택 알고리즘과 함께 작업증명을 사용하며 튜링-완전 스크립팅 언어는 없습니다 (단, 루프없는 간단한 스크립트가 지원됩니다). 비트코인 블록체인에는 계정 개념이 없습니다. 대신에, "소비되지 않은 트랜잭션 출력" (UTXO Unspent Transaction Output) 모델을 사용합니다.

2.9.2. 이더리움 [5]; https://ethereum.org/ 이더리움 (2015)은 튜링-완전스마트 컨트랙트를 지원하는 최초의 블록체인 입니다.

따라서 이 프로젝트는 전형적인 2세대 프로젝트이며 그 중에서도 가장 많이 사용되는 프로젝트입니다. 싱글-블록체인에 작업증명을 사용하지만 스마트 컨트랙트와 계정이 있습니다.

2.9.3. NXT; https://nxtplatform.org/. *NXT* (2014)는 최초의 PoS 기반 블록체인 및 통화입니다. 여전히 싱글-체인이며 스마트 컨트랙트 지원이 없습니다.

2.9.4. Tezos; https://www.tezos.com/. *Tezos* (2018 또는 그 이후)는 제안된 PoS 기반 싱글-블록체인 프로젝트입니다. 이 프로젝트를 여기서 언급하는 이유는 이것의 고유한 특징 때문입니다: 이의 블록 해석 함수인 *ev_block* (cf. **2.2.6**)은 고정되어 있지 않지만 새 버전을 블록체인에 적용하여 (그리고 제안된 변경사항에 대한 투표를 수집하여) 업그레이드 할 수 있는 OCaml 모듈에 의해 결정됩니다. 이런 방식으로 먼저 "바닐라(vanilla)" 테조스 블록체인을 배포한 다음 하드포크가 필요없이 원하는 방향으로 블록 해석기능을 점진적으로 변경하여 사용자 정의 싱글-체인프로젝트를 생성 할 수 있습니다.

이 아이디어는 흥미롭지만 C++과 같은 다른 언어로 최적화된 구현을 금지하는 명백한 단점이 있으므로 테조스 기반 블록체인은 성능이 낮을 수 밖에 없습니다. 특정 구현을 고치지 않고 제안된 블록 해석 함수 *ev_trans* 의 공식 사양을 게재함으로써 비슷한 결과가 얻어졌을 것으로 생각됩니다.

2.9.5. 캐스퍼 (Casper). [33] 캐스퍼는 이더리움을 위한 곧 있을 PoS 알고리즘입니다. 2017년 (또는 2018년)에 점진적 배치가 성공하면, 이더리움을 스마트 컨트랙트 지원을 통해 싱글-체인 PoS 또는 혼합 PoW + PoS 시스템으로 변경할 예정이며, 이더리움을 3세대 프로젝트로 전환시킵니다.

2.9.6. 비트쉐어 (BitShares) [14]; https://bitshares.org.비트쉐어 (2014)는 분산형 블록체인-기반 교환을 위한 플랫폼입니다. 스마트 컨트랙트가 없는 이기종 멀티-블록체인 DPoS 시스템입니다. 블록체인 상태가 메모리에 적합하다고 가정할 때 C ++로 효율적으로 구현할 수 있는 사전 정의된 특수 트랜잭션 유형만 허용하여 높은 성능을 달성합니다. 또한 위임지분증명을 사용하는 최초의 블록체인 프로젝트로서 최소한 여러 특수한 목적을 위해 적어도 그 생존력을 보여줍니다.

2.9.7. EOS [8]; https://eos.io. EOS (2018년 또는 그 이후)는 스마트 컨트랙트 지원과 메시징에 대한 최소한의 지원 (**2.8.14**에 설명된 의미에서 여전히 "느슨하게 결합된")을 갖춘 제안된 이기종 멀티-블록체인 DPoS 시스템입니다. 이전에 비트쉐어 및 SteemIt 프로젝트를 성공적으로 만들었던 동일한 팀의 시도로서 DPoS 합의 알고리즘의 장점을 보여줍니다. 확장성은 이를 필요한 프로젝트 (예: 분산형 거래소는 비트쉐어의 기능과 마찬가지로 최적화된 일련의 특수 트랜잭션을 지원하는 워크체인을 사용할 수 있음)를 위한 특수 워크체인을 만들고 동일한 규칙을 사용하여 여러 워크체인을 생성함으로써 확장됩니다 (**2.8.10**에서 설명된 의미에서의 연합). 이 확장성 접근법의 단점과 한계는 위의 인용문 (*loc. cit.*)에서 논의되었습니다. 또한 DPoS, 샤딩, 워크체인 간의 상호작용 및 블록체인 시스템의 확장성에 대한 함의에 대한 자세한 내용은 **2.8.5, 2.8.12** 및 **2.8.14**를 참조하십시오.

동시에, EOS 또는 다른 블록체인도 "블록체인 안에 페이스북 만들기" (cf. **2.9.13**)를 할 수 없더라도, 우리는 EOS가 비트쉐어 (분산형 교환) 및 SteemIt (분산형 블로그 플랫폼)과 유사하게 고도로 전문화된 약간 약하게 상호작용하는 분산형 애플리케이션을 위한 편리한 플랫폼이 될 수 있다고 생각합니다.

2.9.8. 폴카닷 (PolkaDot) [24]; https://polkadot.io/. 폴카닷 (2019년 또는 그 이후)은 최고의 사상과 가장 상세히 제안된 멀티체인 지분증명 프로젝트 중 하나입니다; 이것의 개발은 Ethereum 공동 창립자 중 한 명이 주도합니다.

[33] https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/

이 프로젝트는 우리 지도상의 TON 블록체인에 가장 가까운 프로젝트 중 하나입니다. (사실, 우리는 폴카닷 프로젝트의 "어부"와 "지명자"라는 용어에 대해 빚을 지고 있습니다.)

폴카닷은 새로운 블록 및 마스터체인 (예: Ethereum 블록체인)을 생성하기 위해 BFT 합의가 있는 이기종의 "느슨하게 결합된" 멀티체인 지분증명 프로젝트입니다. 또한 **2.4.19**에서 설명한 (느린 버전의) TON과 비슷하게 하이퍼큐브 라우팅을 사용합니다.

이것의 유일한 특징은 퍼블릭뿐만 아니라 프라이빗 블럭체인을 생성할 수 있다는 것입니다. 이러한 프라이빗 블록체인은 다른 퍼블릭 블록체인인 폴카닷 또는 다른 것과도 상호 작용할 수 있습니다.

이와 같이 폴카닷은 대규모 프라이빗 블록체인을 위한 플랫폼이 될 수 있습니다. 예를 들어 은행 컨소시엄이 서로 자금을 신속하게 전송하거나 대기업이 프라이빗 블록체인 기술에 대해 사용할 수 있는 다른 용도로 사용할 수 있습니다. 그러나 폴카닷에는 샤딩 지원이 없으며 단단히 결합되어 있지 않습니다. 이것은 이오스와 비슷하게 확장성을 다소 방해합니다 (어쩌면 폴카닷은 DPoS 대신 BFT PoS를 사용하므로 조금 더 나을 수 있습니다).

2.9.9. 유니버사 (Universa); https://universa.io. 이 이색적인 블록체인 프로젝트에 대해 언급하는 유일한 이유는 우리의 무한 샤딩 패러다임 (cf. **2.1.2**)과 유사한 것을 명시적으로 언급하는 지금까지 유일한 프로젝트이기 때문입니다. 또 다른 특이점은 비잔틴 결함 허용과 관련된 모든 문제의 우회는 프로젝트의 신뢰할 수 있고 라이센스가 있는 파트너 만이 밸리데이터로 인정될 것이므로 결코 무효한 블록을 커밋하지 않을 것이라고 약속한 것입니다. 이는 흥미로운 결정입니다; 그러나 본질적으로 어떤 블록체인 프로젝트가 일반적으로 피하고 싶어하는 의도적으로 중앙집중화 블록체인 프로젝트를 만듭니다. (신뢰할 수 있는 중앙집중화된 환경에서 작업하려면 왜 블록체인이 필요합니까?).

2.9.10. 플라즈마 (Plasma); https://plasma.io. 플라즈마 (2019?)는 이더리움의 다른 공동설립자의 참신한 블록체인 프로젝트입니다. 샤딩을 도입하지 않고 이더리움의 일부 한계를 완화하기로 되어 있습니다. 본질적으로 이더리움과는 별개의 프로젝트로 최상위 레벨에서 이더리움 블록체인 (외부 마스터체인으로 사용됨)에 묶여진 (이기종) 워크체인의 계층구조를 도입합니다. 자금은 계층구조의 블록체인 (루트로 이더리움 블록체인에서 시작)에서 수행할 작업에 대한 설명과 함께 전송할 수 있습니다.

그런 다음 필요한 계산이 자식 워크체인에서 수행 (가능하면 원본 작업의 일부를 트리 아래로 더 전송해야 할 수도 있음)되고 결과가 전달되고 보상이 수집됩니다. 일관성을 유지하고 이러한 워크체인의 유효성을 검사하는 문제는 사용자가 잘못하여 워크체인에서 상위 워크체인 (느리긴 하지만) 으로 일방적으로 자금을 인출하고 자금 및 작업을 다른 워크체인에 다시 할당할 수 있게 해주는 (결제채널에서 영감을 받은) 메커니즘을 통해 우회됩니다.

이런식으로 플라즈마는 "수학적 보조프로세서"와 같은 이더리움 블록체인에 연결된 분산형 계산을 위한 플랫폼이 될 수 있습니다. 그러나 이것이 진정한 범용 확장성을 달성하는 방법처럼 보이지는 않습니다.

2.9.11. 특수 블록체인 프로젝트 (Specialized blockchain projects). 또한 FileCoin (사용자가 비용을 지불할 다른 사용자의 파일을 저장하기 위한 디스크 공간을 제공하도록 유도하는 시스템), Golem (3D 렌더링과 같은 전문 애플리케이션을 위해 컴퓨팅 능력을 임대 및 대여를 위한 블록체인 기반 플랫폼) 또는 SONM (또 다른 유사한 컴퓨팅 능력 임대 프로젝트) 같은 특수 블록체인 프로젝트가 있습니다. 그러한 프로젝트는 개념적으로 블록체인 조직 수준에서 새로운 것을 도입하지 않습니다; 오히려 이들은 특정 블록체인 애플리케이션이며, 필요한 성능을 제공할 수 있다면 범용 블록체인에서 실행중인 스마트 컨트랙트를 구현할 수 있는 특정 블록체인 애플리케이션입니다. 따라서, 이런 종류의 프로젝트는 EOS, 폴카닷 또는 TON과 같은 기존 또는 계획된 블록체인 프로젝트 중 하나를 기반으로 사용할 것입니다. 프로젝트가 "샤딩"에 기반한 "진정한" 확장성을 필요로 한다면 TON을 사용하는 것이 좋습니다; 목적에 맞게 명시적으로 최적화된 자체 워크체인 제품군을 정의하여"연합된" 컨텍스트에서 작업하는데 만족하면 EOS 또는 폴카닷을 선택할 수 있습니다.

2.9.12. TON 블록체인 (The TON Blockchain). TON (텔레그램 오픈 네터워크) 블록체인 (2018년 계획)은 이 문서에서 설명하는 프로젝트입니다. 이것은 최초의 5세대 블록체인 프로젝트, 즉 BFT PoS 멀티 프로젝트, 동종/이기종 혼합 프로젝트이며, 고유 샤딩 지원 및 단단히 결합된 (샤드 가능한) 사용자 정의 워크체인을 지원하도록 설계되었습니다 (특히, 모든 샤드체인의 일관된 상태를 유지하면서 거의 즉시 샤드간에 메시지를 전달할 수 있음). 이와 같이, 블록체인에서 구현할 수 있는 모든 애플리케이션을 기본적으로 수용할 수 있는 진정한 확장성을 갖춘 범용 블록체인 프로젝트 입니다. TON 프로젝트의 다른 구성 요소들 (1장)에 의해 보강되면, 그 가능성은 더욱 확대됩니다.

2.9.13. "페이스북을 블록체인에 업로드" 할 수 있습니까? (Is it possible to “upload Facebook into a blockchain”?) 때때로 사람들은 블록체인에 있는 분산형 애플리케이션으로 페이스북의 규모의 소셜 네트워크를 구현할 수 있다고 주장합니다. 일반적으로 좋아하는 블록체인 프로젝트는 이러한 애플리케이션의 가능한 "호스트"로 인용됩니다.

이것이 기술적으로 불가능하다고 말할 수는 없습니다. 물론, 그러한 대형 애플리케이션이 너무 느리게 작동하지 않도록 하기위해 진정한 샤딩 (즉, TON)을 갖는 단단히 결합된 블록체인 프로젝트가 필요합니다. (예: 하나의 샤드체인에 거주하는 사용자로부터 다른 샤드체인에 거주하는 친구에게 합리적인 지연으로 메시지 및 업데이트 전달). 그러나 우리는 이것이 필요하지 않으며 결코 이루어지지 않을 것이라고 생각합니다. 왜냐하면 가격이 엄청나기 때문입니다.

"페이스북을 블록체인으로 업로드하기"를 고려해 보겠습니다; 유사한 규모의 다른 프로젝트도 예제로 사용될 수 있습니다. 페이스북이 블록체인에 업로드되면 현재 페이스북의 서버에 의해 수행된 모든 작업이 특정 블록체인 (예: TON의 샤드체인)에서 트랜잭션으로 직렬화되며 이 블록체인의 모든 밸리데이터에서 수행됩니다. 모든 블록이 적어도 20개의 밸리데이터 서명 (즉, DPOS 시스템에서 처럼 즉시 또는 결과적으로)을 수집할 것으로 예상할 경우 각 작업을 적어도 20회 수행해야 합니다. 비슷하게 페이스북의 서버가 디스크에 보관하는 모든 데이터는 해당 샤드체인에 대한 모든 밸리데이터의 디스크 (즉, 최소 20개 사본)에 보관됩니다.

밸리데이터는 기본적으로 페이스북에서 사용되는 것과 동일한 서버 (또는 서버 클러스터이지만 이 주장의 유효성에는 영향을 미치지 않음)이기 때문에 블록체인에서 페이스북 실행과 관련된 총 하드웨어 비용은 최소한 기존 방식으로 구현된 것보다 20배 더 큽니다.

실제로, 블록체인의 가상 머신이 최적화된 컴파일된 코드를 실행하는 "베어 CPU"보다 느리고 그 스토리지가 페이스북 특정 문제에 최적화되어 있지 않기 때문에 비용은 여전히 훨씬 더 높을 것입니다. 페이스북에 적합한 특정 트랜잭션을 사용하여 특정 작업체인을 작성함으로써 이 문제를 부분적으로 완화할 수 있습니
다; 이것은 비트쉐어와 EOS가 고성능을 달성하기 위한 접근 방식이며, TON 블록체인에서도 사용 가능합니다. 그러나 일반적인 블록체인 설계는 여전히 그 자체로 몇 가지 추가 제한을 가합니다. 예를 들어, 모든 작업을 블록에 트랜잭션으로 등록하고, 머클트리에서 이러한 트랜잭션을 구성하고, 머클 해시를 계산 및 확인하고, 이 블록을 추가 전파하는 등의 작업이 필요합니다.

따라서, 보수적인 추정은 그 규모의 소셜 네트워크를 호스트하는 블록체인 프로젝트의 유효성을 검사하기 위해 페이스북에서 사용하는 것과 동일한 성능의 서버를 100배 더 필요로 한다는 것입니다. 분산형 애플리케이션을 소유한 회사 (각 페이스북 페이지에 7개가 아닌 700개의 광고를 본다고 상상해 보십시오) 또는 사용자 중 누군가가 서버에 비용을 지불해야합니다. 어느 쪽이든, 이것은 경제적으로 실용적이지 않습니다.

우리는 모든것이 블록체인에 업로드 되어야 한다는 것은 사실이 아니라고 생각합니다. 예를 들어, 블록체인에 사용자 사진을 보관할 필요는 없습니다. 블록체인에 이러한 사진의 해시를 등록하고 분산형 오프-체인 저장소 (예. FileCoin 또는 TON 저장소)에 사진을 보관하는 것이 더 좋습니다. 이것이 TON이 단순한 블록체인 프로젝트가 아니라 1장과 4장에서 요약된 TON 블록체인을 중심으로 한 여러 구성 요소 (TON P2P 네트워크, TON 저장소, TON 서비스)의 모음임을 보여줍니다.

3 TON 네트워킹 (TON Networking)

블록체인 프로젝트에는 블록형식 및 블록체인 유효성 검사규칙의 지정뿐만 아니라 새 블록을 전파하고 트랜잭션 후보를 보내고 수집하는데 사용되는 네트워크 프로토콜이 필요합니다. 블록체인 프로젝트는 일반적으로 분산화 될 것으로 예상되기 때문에 이 네트워크는 반드시 피어-투-피어 이어야합니다. 따라서 예를 들어 고전적인 온라인 뱅킹 애플리케이션과 같이 서버의 중앙형 그룹에 의존할 수 없으며 기존의 클라이언트-서버 아키텍처를 사용 할 수 없습니다. 클라이언트-서버와 같은 방식으로 전체 풀-노드에 연결해야 하는 라이트 클라이언트 (예. 라이트 암호화폐 월렛 스마트폰 애플리케이션)조차도 전체 노드에 연결하는데 사용될 프로토콜이 충분히 표준화되어 있으면 이전 피어가 다운된 경우에도 실제로 다른 노드에 자유롭게 연결할 수 있습니다.

비트코인이나 이더리움과 같은 싱글-블록체인 프로젝트의 네트워킹 요구는 매우 쉽게 충족될 수 있지만 (본질적으로 "임의의" 피어-투-피어 오버레이 네트워크를 구성하고 모든 새 블록 및 트랜잭션 후보를 가십 프로토콜에 의해 전파해야 합니다), TON 블록체인과 같은 멀티-블록체인 프로젝트는 훨씬 더 까다로운 작업입니다. (예. 반드시 그들 전부가 아닌 일부 샤드체인에 대한 업데이트를 구독할 수 있어야 합니다). 따라서 TON 블록체인과 TON 프로젝트의 네트워킹 부분은 전체적으로 적어도 간단한 논의가 필요합니다.

반면 TON 블록체인을 지원하는데 필요한 보다 정교한 네트워크 프로토콜이 마련되면 TON 블록체인의 즉각적인 요구사항과 관련이 없는 용도로도 쉽게 사용할 수 있어 TON 생태계에서 새로운 서비스를 창출할 수 있는 더 많은 가능성과 유연성을 제공합니다.

3.1 추상 데이터그램 네트워크 계층 (Abstract Datagram Network Layer)

TON 네트워킹 프로토콜을 구축하는 초석은 (TON) 추상 (데이터그램) 네트워크 계층입니다. 모든 노드가 256-비트 "추상 네트워크 주소"로 표시되는 특정 "네트워크 ID"를 사용하도록 허용하고 발신자와 수신자를 식별하기 위해 이 256-비트 네트워크 주소만을 사용하여 통신합니다 (첫 번째 단계로 데이터그램을 서로 보냅니다). 특히, IPv4 또는 IPv6 주소, UDP 포트 번호 등을 걱정할 필요가 없습니다. 그들은 추상 네트워크 계층에 의해 숨겨져 있습니다.

3.1.1. 추상 네트워크 주소 (Abstract network addresses). 추상 네트워크 주소 또는 추상주소 또는 간단하게 주소 는 기본적으로 256-비트 ECC 퍼블릭 키와 동일한 256-비트 정수입니다. 이 퍼블릭 키는 임의로 생성될 수 있으므로 노드가 좋아하는 만큼 많은 다른 네트워크 ID를 생성할 수 있습니다. 그러나, 이러한 주소에 대한 메시지를 수신 (및 해독) 하려면 해당 프라이빗 키를 알아야 합니다.

실제로 주소는 공개 키 자체가 아닙니다. 대신 생성자 (처음 4-바이트)에 따라 몇 가지 유형의 퍼블릭 키와 주소를 설명할 수 있는 직렬화된 TL-객체 (cf. **2.2.5**)의 256-비트 해시(HASH = SHA256)입니다. 가장 단순한 경우, 이 직렬화된 TL-객체는 4-바이트 매직넘버와 256-비트 타원곡선 암호 (ECC) 퍼블릭 키로 구성됩니다; 이 경우, 주소는 이 36-바이트 구조의 해시와 같습니다. 그러나 2048-비트RSA 키 또는 다른 퍼블릭 키 암호화 체계를 대신 사용할 수 있습니다.

노드가 다른 노드의 추상주소를 알게되면, "프리이미지(preimage)" (즉, 직렬화된 TL-객체, 해시가 추상주소와 같음)를 수신해야 합니다. 그렇지 않으면 데이터그램을 암호화하여 해당 주소로 전송할 수 없습니다.

3.1.2. 낮은 수준 네트워크. UDP 구현 (Lower-level networks. UDP implementation). 거의 모든 TON 네트워킹 구성 요소의 관점에서 볼 때, 존재하는 유일한 것은 하나의 추상주소에서 다른 주소로 데이터그램을 전송할 수 있는 네트워크 (추상 데이터그램 네트워킹 계층)입니다. 원칙적으로 ADNL (추상 데이터그램 네트워크 계층)은 다른 기존 네트워크 기술을 통해 구현될 수 있습니다. 그러나, 우리는 IPv4 / IPv6 네트워크 (인터넷 또는 인트라넷과 같은)에서 UDP를 통해 UDP를 사용할 수 없는 경우 선택적인 TCP 을 사용하여 이를 구현하려고 합니다.

3.1.3. UDP를 통한 ADNL의 가장 단순한 경우 (Simplest case of ADNL over UDP). 발신자의 추상주소에서 다른 추상주소 (알려진 프리이미지와 함께)로 데이터그램을 전송하는 가장 간단한 경우는 다음과 같이 구현할 수 있습니다.

발신자가 대상 추상주소를 소유한 수신자의 IP 주소와 UDP 포트를 어떻게든 알고 있고 수신자와 발신자 모두 256-비트 ECC 퍼블릭 키에서 파생된 추상주소를 사용한다고 가정합시다.

이 경우, 발신자는 단순히 프라이빗 키로 이루어진 ECC 서명과 발신주소(또는 수신자가 아직 해당 프리이미지를 알지 못하는 경우 원본주소의 이미지)로 전송할 데이터그램을 간단하게 보완합니다.

결과는 수신자의 퍼블릭 키로 암호화되어 UDP 데이터그램으로 내장되어 수신자의 알려진 IP 및 포트로 전송됩니다. UDP 데이터그램의 처음 256 비트는 수신자의 추상주소를 포함하기 때문에 수신자는 데이터그램의 나머지 부분을 해독하는데 사용해야 하는 프라이빗 키를 식별할 수 있습니다. 그 후에야 발신자의 신분이 게시됩니다.

3.1.4. 덜 안전한 방법, 발신자의 평문 주소 (Less secure way, with the sender's address in plaintext). 때로는 수신자와 발신자의 주소가 UDP 데이터그램에서 일반 텍스트로 유지되는 경우 덜 안전한 계획으로 충분할 수 있습니다; 발신자의 프라이빗 키와 수신자의 퍼블릭 키를 ECDH (타원곡선 디피-헬만)을 사용하여 결합한 이후에 사용되는 256-비트 공유암호와 암호화되지 않은 부분에 포함된 임의의 256-비트 논스를 생성하여 암호화에 사용되는 ASE 키를 가져옵니다. 무결성은 예를 들어 원래의 일반 텍스트 데이터의 해시를 암호화 전에 일반 텍스트에 연결함으로써 제공될 수 있습t니다.
이 방법은 두 주소 사이에 둘 이상의 데이터그램이 교환될 것으로 예상되는경우 공유 암호를 한 번만 계산하여 캐시 할 수 있다는 이점이 있습니다. 그러면 더 느린 타원곡선 연산은 다음 데이터 그램의 암호화 또는 암호 해독에 더 이상 필요하지 않습니다.

3.1.5. 채널 및 채널 ID (Channels and channel identifiers). 가장 간단한 경우, 내장된 TON ADNL 데이터그램을 전달하는 UDP 데이터그램의 처음 256 비트는 수신자의 주소와 같습니다. 그러나 일반적으로 채널 ID를 구성합니다. 다양한 유형의 채널이 있습니다. 그들 중 일부는 포인트-투-포인트 입니다; 채널 ID들은 장래에 많은 데이터를 교환하고 **3.1.3** 또는 **3.1.4**에서 설명한대로 암호화된 여러 개의 패킷을 교환하여 공유 암호를 생성하려는 두 당사자에 의해 생성됩니다. 이는 디피-헬만을 사용하여 클래식 또는 타원곡선을 실행하거나 (추가 보안이 요구되면), 또는 단순히 한 당사자가 임의의 공유 비밀을 생성하여 상대방에게 전송하는 방법이 있습니다.

그런 다음 채널 ID는 예를 들어 해싱을 통해서 같은 일부 추가데이터 (예. 발신자 및 수신자의 주소)와 결합된 공유 암호에서 파생되며 해당 ID는 공유 비밀로 암호화된 데이터를 전송하는 UDP 데이터그램의 처음 256-비트로 사용됩니다.

3.1.6. 터널 ID로서의 채널 (Channel as a tunnel identifier). 일반적으로, "채널" 또는 "채널 ID"는 단순히 수신자에게 알려진 인바운드 UDP 데이터그램을 처리하는 방법을 선택합니다. 만약 채널이 수신자의 추상주소인 경우 **3.1.3** 또는 **3.1.4** 에 설명된 대로 처리됩니다; 만약 채널이 **3.1.5** 에서 논의된 설립된 P2P 채널인 경우, 위의 인용문 등에서 설명된대로 공유 비밀을 사용하여 데이터그램을 해독하는 것으로 처리됩니다.

특히, 채널 ID는 직접적인 수신자)가 실제 수신된 메시지를 다른 사람—실제 수신자 또는 다른 프록시—에게 전달할 때 "터널"을 실제 선택할 수 있습니다. 일부 암호화 또는 암호해독 단계 ("양파 라우팅"[11] 또는 심지어 "마늘 라우팅"[34] 과 같은)가 그 과정에서 수행될 수 있으며 또 다른 채널 ID는 재암호화된 전달 패킷에 사용될 수 있습니다 (예를 들어, 피어-투-피어 채널은 경로상의 다음 수신자에게 패킷을전달하기 위해 사용될 수 있음).

이러한 방식으로 TOR 또는 *I²P* 프로젝트에서 제공하는 것과 유사한 "터널링" 및 "프록싱"에 대한 일부 지원은 TON 추상 데이터그램 네트워크 계층의 수준에서, 그러한 추가가 불가지론적인 모든 상위 TON 네트워크 프로토콜의 기능에 영향을 주지 않으면서 추가될 수 있습니다. 이 기회는 TON 프록시 서비스에 의해 활용됩니다 (cf. **4.1.11**).

3.1.7. 제로 채널과 부트스트랩 문제 (Zero channel and the bootstrap problem). 일반적으로 TON ADNL 노드는 추상 소와 프리이미지 (즉, 퍼블릭 키) 및 IP 주소 및 UDP 포트에 대한 정보가 포함된 "인접 테이블"을 갖습니다. 그런 다음 이러한 알려진 노드에서 얻은 정보를 특수쿼리에 대한 응답으로 사용하여 이 테이블을 점차 확장하고 때로는 오래된 레코드를 제거합니다.

그러나, TON ADNL 노드가 시작될 때 노드의 IP 주소와 UDP 포트만 알고, 추상주소는 알 수 없는 노드가 발생할 수 있습니다. 예를 들어, 간단한 클라이언트가 이전에 캐시된 노드와 소프트웨어에 하드 코딩된 노드에 액세스 할 수 없고DNS를 통해 해결할 노드의 IP 주소나 DNS 도메인을 입력하도록 사용자에게 요청해야 합니다.

이 경우 노드는 해당 노드의 특수 "제로 채널"로 패킷을 보냅니다. 이것은 받는 사람의 퍼블릭 키에 대한 지식이 필요하지 않지만 메시지는 (여전히 보낸 사람의 ID와 서명을 포함해야 하므로) 암호화되지 않고 전송됩니다. 일반적으로 수신자의 ID (아마도 이 용도로 특별히 만들어진 일회성 ID)를 얻은 다음 더 안전한 방법으로 의사소통을 시작해야 합니다. 적어도 하나의 노드가 알려지면 "인접 테이블"과 "라우팅 테이블"을 더 많은 항목으로 채우고 이미 알려진 노드로 전송된 특수쿼리에 대한 응답에서 이를 학습합니다.

모든 노드가 제로 채널로 전송된 데이터그램을 처리해야 하는 것은 아니지만, 라이트 클라이언트를 부트스트랩 하는데 사용되는 노드는 이 기능을 지원해야 합니다.

[34] https://geti2p.net/en/docs/how/garlic-routing

3.1.8. ADNL을 통한 TCP 같은 스트림 프로토콜 (TCP-like stream protocol over ADNL). ADNL은 256-비트 추상주소를 기반으로 하는 신뢰할 수 없는 (작은 크기) 데이터그램 프로토콜로서 보다 정교한 네트워크 프로토콜의 기반으로 사용할 수 있습니다. 예를 들어, IP와 유사한 대체 프로토콜로 ADNL을 사용하여 TCP와 유사한 스트림 프로토콜을 구축할 수 있습니다. 그러나, TON 프로젝트의 대부분의 구성요소에는 이러한 스트림 프로토콜이 필요하지 않습니다.

3.1.9. RLDP 또는 ADNL을 통한 신뢰할 수 있는 대형 데이터그램 프로토콜 (RLDP, or Reliable Large Datagram Protocol over ADNL). RLDP 라 불리는 ADNL 기반의 신뢰할 수 있는 임의 크기 데이터 그램 프로토콜은 TCP와 같은 프로토콜 대신에 사용됩니다. 이 신뢰할 수 있는 데이터그램 프로토콜은 원격 호스트에 RPC 쿼리를 보내고 응답을 수신하는데 사용될 수 있습니다 (cf. **4.1.5**).

3.2 TON DHT: 카뎀리아 같은 분산형 해시 테이블 (TON DHT: Kademlia-like Distributed Hash Table)

TON분산형 해시 테이블 *(DHT)*는 TON 프로젝트의 네트워킹 부분에서 중요한 역할을 담당하며 네트워크의 다른 노드를 찾는데 사용됩니다. 예를 들어, 샤드체인에 트랜잭션을 커밋하고자 하는 클라이언트는 해당 샤드체인의 밸리데이터 또는 콜레이터, 혹은 클라이언트의 트랜잭션을 콜레이터로 중계할 수 있는 적어도 일부 노드를 찾고 싶을 수 있습니다. TON DHT에서 특수 키를 찾아보면 됩니다. TON DHT의 또 다른 중요한 응용은 임의의 키나 새 노드의 주소를 찾는 것만으로 새로운 노드의 인접 테이블 (cf. **3.1.7**)을 빠르게 채우는데 사용할 수 있다는 것입니다. 만약 노드가 인바운드 데이터그램에 대해 프록싱 및 터널링을 사용하면 노드는 TON DHT에 터널ID 및 해당 진입점 (예. IP주소 및 UDP 포트)을 게시합니다; 그 노드에 데이터그램을 전송하고자 하는 모든 노드는 DHT로부터 이 연락처 정보를 먼저 얻을 것입니다. TON DHT는 카뎀리아 같은 분산형 해시 테이블의 구성원입니다 [16].

3.2.1. TON DHT의 키 (Keys of the TON DHT). TON DHT의 키는 단순히 256-비트 정수입니다. 대부분의 경우, 키의 프리이미지 또는 키 설명이라고 하는 TL-직렬화된 객체의 SHA256 (cf. **2.2.5**)으로 계산됩니다. 경우에 따라 TON 네트워크 노드의 추상주소 (cf. **3.1.1**)도 TON DHT의 키로 사용할 수 있습니다. 왜냐하면 이 키는 256-비트이기도 하고 TL-직렬화된 객체의 해시이기도 합니다. 예를 들어, 노드가자신의IP주소를 게시하는 것을 두려워하지 않으면 DHT의 키로 그 주소를 찾는 것만으로추상주소를 아는 모든 사람이 노드를 찾을 수 있습니다.

3.2.2. DHT 값 (Values of the DHT). 이러한 256-비트 키에 지정된 값은 본질적으로 제한된 길이의 임의 바이트 문자열입니다. 그러한 바이트 문자열의 해석은 대응하는 키의 프리이미지에 의해 결정됩니다; 일반적으로 키를 검색하는 노드와 키를 저장하는 노드에게 모두 알려져 있습니다.

3.2.3. DHT의 노드. 반영구적 네트워크 신분 (Nodes of the DHT. Semi-permanent network identities). TON DHT의 키-값 매핑은 DHT의 노드 - 본질적으로 TON 네트워크의 모든 구성원에 의해 유지됩니다. 이를 위해 TON 네트워크의 모든 노드 (아마도 일부 매우 가벼운 노드를 제외하면)에는 **3.1.1** 에서 설명된 임시 및 영구 추상주소의 수를 제외하고 TON DHT 구성원임을 나타내는 "반영구적 주소"가 적어도 하나 있습니다. 이 반영구 또는 DHT 주소는 너무 자주 변경하면 안됩니다. 그렇지 않으면 다른 노드가 검색하는 키를 찾을 수 없습니다. 노드가 "참" 신분을 밝히고 싶지 않은 경우 DHT에 참여할 목적으로만 사용되는 별도의 추상주소를 생성합니다. 그러나 이 추상 주소는 노드의 IP주소 및 포트와 연관될 것이므로 공개되어야 합니다.

3.2.4. 카뎀리아 거리 (Kademlia distance). 이제는 256-비트 키와 256-비트 (반영구적) 노드주소가 모두 있습니다. 우리는 소위 XOR 거리 또는 카뎀리아 거리 d_K 를 256-비트 배열 집합에 소개합니다.

$$d_k(x, y) := (x \oplus y) \text{ 부호없는 256-비트 정수로 해석됩니다.}$$

여기 $x \oplus y$ 는 동일한 길이의 두 비트 배열의 비트별 eXclusive OR (XOR)을 나타냅니다. 카뎀리아 거리는 모든 256-비트 배열의 집합 2^{256} 에 대한 미터법을 도입합니다. 특히, $x = y$, $d_K(x, y) = d_K(y, x)$ 및 $d_K(x, z) \leq d_K(x, y) + d_K(y, z)$ 와 같은 경우에만 $d_K(x, y) = 0$ 입니다. 또 다른 중요한 속성은 x 로부터 $(x : d_K(x, y) = d_K(x, y')$는 $y = y'$ 을 의미) 임의의 주어진 거리에 단 하나의 점이 있다는 것입니다.

3.2.5. 카뎀리아 같은 DHT 및 TON DHT (Kademlia-like DHTs and the TON DHT). 256-비트 키와 256-비트 노드주소를 가진 DHT는 s 카뎀리아에서 가까운 노드의 키 K 값을 K로 유지할 것으로 예상되는 경우 카뎀리아 같은 DHT입니다(즉, 그들의 주소에서 K 까지 가장 작은 카뎀리아 거리를 갖는 s 노드).

여기서 s 는 DHT의 신뢰성을 향상시키는데 필요한 $s = 7$과 같은 작은 매개변수입니다 (우리가 K 와 가장 가까운 키를 하나의 노드에서만 유지한다면 그 유일한 노드가 오프라인 상태가 될 경우 그 키의 값이 손실될 것입니다).

TON DHT는 이 정의에 따라 카뎀리아 같은 DHT입니다. 이는 **3.1**에서 설명하는 ADNL 프로토콜을 통해 구현됩니다.

3.2.6. 카뎀리아 라우팅 테이블 (Kademlia routing table). 카뎀리아 같은 DHT에 참여하는 노드는 대개 카뎀리아 라우팅 테이블을 유지 관리합니다. TON DHT의 경우 0부터 n-1까지 번호가 매겨진 n=256 버킷으로 구성됩니다. i-번째 버킷에는 노드의 주소 a 에서 카뎀리아 거리 2^i부터 $2^{i+1} - 1$ 까지에 있는 일부 알려진 노드 (고정된 수 t 의 "최상" 노드와 어쩌면 일부 추가 후보자)에 대한 정보가 포함됩니다.[35] 이 정보에는 해당 (반영구적인) 주소, IP 주소 및 UDP 포트, 마지막 핑의 시간 및 지연과 같은 일부 가용성 정보가 포함됩니다.

카뎀리아 노드가 일부 쿼리의 결과로 다른 카뎀리아 노드에 대해 알게되면 이를 먼저 라우팅 테이블의 적합한 버킷에 후보 노드를 포함시킵니다. 그런 다음 해당 버킷의 "최상" 노드 중 일부가 실패하면 (예. 오랜 시간 동안 핑 쿼리에 응답하지 않으면) 일부 후보에 의해 대체될 수 있습니다. 이 방법으로 카뎀리아 라우팅 테이블이 채워집니다.

카뎀리아 라우팅 테이블의 새 노드는 **3.1.7** 에서 설명한 ADNL 인접 테이블에도 포함됩니다. 카뎀리아 라우팅 테이블의 버킷에서 "최상" 노드가 자주 사용되는 경우 **3.1.5** 에서 설명한 의미의 채널을 개설하여 데이터그램의 암호화를 촉진합니다.

TON DHT의 특별한 특징은 카뎀리아 라우팅 테이블의 버킷에 대한 "최상" 노드로서 가장 작은 왕복이동 지연을 가진 노드를 선택하려고 한다는 것입니다.

3.2.7. (카뎀리아 네트워크 쿼리) (Kademlia network queries.) 카뎀리아 노드는 대개 다음 네트워크 쿼리를 지원합니다.

- PING - 노드 가용성을 검사합니다.

[35] 버킷에 노드가 충분히 많으면 예를 들어, 카뎀리아 거리의 상위 4-비트에 따라 8개의 하위 버킷으로 세분화할 수 있습니다. 이렇게하면 DHT 조회 속도가 빨라집니다.

- STORE(*key, value*) - 노드가 *value* 를 키*key* 의 값으로 유지하도록 요청합니다. TON DHT의 경우 STORE 쿼리가 약간 더 복잡합니다 (cf. **3.2.9**).
- FIND_NODE(*key, l*) - 카뎀리아에 가장 가까운 (카뎀리아 라우팅 테이블에서) 알려진 노드를 *key* 로 반환하도록 노드에 요청합니다.
- FIND_VALUE(*key, l*) - 위와 같지만 노드가 키 *key* 에 해당하는 값을 알고있으면 값을 반환합니다.

임의의 노드가 키 K 의 값을 검색하기를 원할 때, 모든 알려진 노드들 사이에서 카뎀리아 거리에 관하여 K 에 가장 가까운 (즉, 카뎀리아 라우팅 테이블에서 가져옴), s' 노드들의 세트 S 를 (s'의 일부 작은 값, 예를 들어, $s' = 5$) 먼저 생성합니다. 그런 다음 FIND-_VALUE 쿼리가 각각에 보내지고 응답에 언급된 노드가 S 에 포함됩니다. 그 이후 K 에서 가장 가까운 S 의 s' 노드가 이전에 완료되지 않은 경우 FIND_VALUE 쿼리를 보냅니다. 이 과정은 값이 발견되거나 세트 S 의 성장이 멈출 때까지 계속됩니다. 이것은 카뎀리아 거리와 관련하여 K 에 가장 가까운 노드의 일종의 "빔 검색"입니다.

일부 키 K 의 값을 설정하려면 FIND_VALUE 대신 FIND_NODE 쿼리를 사용하여 $s' \geq s$ 에 대해 동일한 절차를 실행하여 K 에 가장 가까운 노드를 찾습니다. 이후에 저장소 쿼리가 모든 키에 전송됩니다.

카뎀리아 같은 DHT 구현에는 덜 중요한 세부사항이 있습니다 (예. 모든 노드는 예를 들어 매 시간 한 번씩 가장 가까운 노드를 검색해야 하며 STORE 쿼리를 사용하여 저장된 모든 키를 다시 게시해야 합니다). 우리는 당분간 이것을 무시할 것입니다.

3.2.8. 카뎀리아 노드 부팅 (Bookting a Kademlia node). 카뎀리아 노드가 온라인 상태가 되면 먼저 자체 주소를 찾아 카뎀리아 라우팅 테이블을 채웁니다. 이 과정에서 자신에게 가장 가까운 노드 s 를 식별합니다. DHT의 일부분을 채우기 위해 알려진 모든 (*key, value*) 쌍을 그들로부터 다운로드 할 수 있습니다.

3.2.9. TON DHT에 값 저장 (Storing values in TON DHT). TON DHT에 값을 저장하는 것은 일반적인 카뎀리아 같은 DHT와는 약간 다릅니다. 누군가가 값을 저장하고자 할 때, 키 K 자체를 STORE 쿼리뿐만 아니라 이의 프리이미지 — 즉, 키의 “설명"이 포함된 TL-직렬화된 문자열 (처음에 몇 개의 미리 정의된 TL-생성자 중 하나를 포함)을 제공해야 합니다. 이 키 설명은 나중에 키와 값과 함께 노드에 보관됩니다.

키 설명은 향후 업데이트를 대비해서 저장되는 객체의 "유형", "소유자" 및 "업데이트 규칙"을 설명합니다. 소유자는 대개 키 설명에 포함된 퍼블릭 키로 식별됩니다. 이것이 포함되어 있는 경우, 통상 해당 프라이빗 키에 의해 서명된 업데이트만이 받아 들여집니다. 저장된 객체의 "유형"은 일반적으로 바이트 문자열 입니다. 그러나 경우에 따라 더욱 정교해질 수 있습니다. __예를들어, 입력터널 설명 (cf. **3.1.6**) 또는 노드 주소 모음.

"업데이트 규칙"도 다를 수 있습니다. 경우에 따라 새 값이 소유자에 의해 서명된 경우 이전 값을 새 값으로 대체할 수 있습니다 (서명은 값의 일부로 유지되어야 하며 이 키의 값을 얻은 후에는 다른 노드에 의해 나중에 확인해야 합니다). 다른 경우에는 이전 값이 어떻게든 새 값에 영향을 줍니다. 예를 들어 배열번호가 포함될 수 있으며 새 배열번호가 큰 경우에만 이전 값을 덮어 씁니다 (재연 공격을 방지하기 위해).

3.2.10. TON DHT의 분산형 "토렌트 트래커" 및 "네트워크 관심 그룹" (Distributed "torrent track-ers" and "network interest groups" in TON DHT). 또 다른 흥미로운 경우는 값에 노드 목록이 포함될 때 - 그들의 IP 주소와 포트, 또는 그들의 추상 주소만 함께 - 및 "업데이트 규칙"이 요청자를 (정체를 확인할 수 있는 경우) 이 목록에 포함시키는 것으로 구성되어 있는 경우입니다.

이 메커니즘은 특정 "토렌트" (즉, 특정 파일)에 관심이 있는 모든 노드가 동일한 토렌트에 관심이 있거나 이미 사본이 있는 다른 노드를 찾을 수 있는 경우 분산형 "토렌트 추적기"를 만드는데 사용될 수 있습니다.

TON 저장소 (cf. **4.1.8**)는 이 기술을 사용하여 필요한 파일의 사본이 있는 노드 (예. 샤드체인 상태의 스냅샷 또는 예전 블록)를 찾습니다. 그러나 더 중요한 용도는 "오버레이 멀티캐스트 서브 네트워크"와 "네트워크 관심 그룹" (cf. **3.3**)을 만드는 것입니다. 이 발상은 오로지 일부 노드만 특정 샤드체인의 업데이트에 관심이 있다는 것입니다. 만약 샤드체인의 수가 매우 많아지면 동일한 샤드에 관심이 있는 하나의 노드조차 찾는 것이 복잡해 질 수 있습니다. 이 "분산형 토렌트 추적기"는 이러한 노드 중 일부를찾는 편리한 방법을 제공합니다. 또 다른 옵션은 밸리데이터에 요청하는 것이지만 이것은 확장성있는 접근법이 아니며 밸리데이터는 임의의 모르는 노드에서 오는 쿼리에응답하지 않을 수도 있습니다.

3.2.11. 폴-백 키 (Fall-back keys). 지금까지 설명한 "키 유형"의 대부분은 일반적으로 0 과 동일한 TL 설명에 추가 32-비트 정수 필드가 있습니다. 그러나, 해당 설명을 해싱하여 얻은 키를 TON DHT에서 검색하거나 업데이트 할 수 없는 경우 이 필드의 값이 증가하고 새로운 시도가 이루어집니다. 이러한 방식으로, 공격을 받고있는 키 근처에 있는 많은 추상주소를 생성하고 해당 DHT 노드를 제어함으로써 키를 "포착"하고 "검열" (즉, 키 보유 공격을 수행) 할 수 없습니다.

3.2.12. 서비스 찾기 (Locating services). TON 네트워크에 위치하고 **3.1** 에 설명된 TON ADNL (위에 구축된 상위 프로토콜)을 통해 제공되는 일부 서비스는 클라이언트가 어디에서 찾을 수 있는지 알 수 있도록 추상주소를 어딘가에 게시하려고 할 수 있습니다.

그러나 TON 블록체인에 서비스의 추상주소를 게시하는 것이 가장 좋은 방법은 아닐 수 있습니다. 왜냐하면 추상주소를 자주 변경해야 할 수도 있고 안정성 또는 부하균형 목적을 위해 여러 주소를 제공하는 것이 합리적 일 수 있기 때문입니다.

대안으로 퍼블릭 키를 TON 블록체인에 게시하고 TL 설명 문자열 (cf. **2.2.5**)에서 "소유자"로 퍼블릭 키를 나타내는 특수 DHT 키를 사용하여 서비스의 추상주소에 대한 최신 목록을 게시할 수 있습니다 이것이 TON 서비스가 이용하는 방법중 하나입니다.

3.2.13. TON 블록체인 계정 소유자 찾기 (Locating owners of TON blockchain accounts). 대부분의 경우, TON 블록체인 계정 소유자는 자신의 개인정보를 침해할 수 있으므로 추상 네트워크 주소 및 특히 IP 주소와 관련되는 것을 좋아하지 않습니다. 그러나, 어떤 경우에는 TON 블록체인 계정의 소유자가 연락할 수 있는 곳에 하나 또는 여러 개의 추상주소를 게시하고자 할 수 있습니다.

전형적인 경우는 TON 페이먼트의 "라이트닝 네트워크” (cf. **5.2**)의 실시간 암호화폐 전송을 위한 플랫폼에 있는 노드의 경우입니다. 퍼블릭 TON 페이먼트 노드는 다른 피어와 결제채널을 설정하는 것뿐만 아니라 이미 설정한 채널을 따라 대금을 이체하기 위해 나중에 연락할 수 있는 추상 네트워크 주소를 게시하기를 원할 수도 있습니다.

하나의 옵션은 결제채널을 생성하는 스마트 컨트랙트에 추상 네트워크 주소를 포함시키는 것입니다. 보다 융통성있는 옵션은 스마트 컨트랙트에 퍼블릭 키를 포함시킨 다음 **3.2.12**에서 설명한대로 DHT를 사용하는 것입니다. 가장 자연스러운 방법은 TON 블로체인의 계정을 제어하는 동일한 프라이빗 키를 사용하여 해당 계정과 관련된 추상주소에 대한 TON DHT의 업데이트에 서명하고 게시하는 것입니다. 이는 **3.2.12**에서 설명한 것과 거의 같은 방식으로 수행됩니다. 그러나 사용된 DHT 키에는 계정의 퍼블릭 키가 포함하는 "계정 설명"의 SHA256과 동일한 *account_id* 자체만 포함하는 특별한 키 설명이 필요합니다. 이 DHT 키의 값에 포함된 서명에는 계정 설명도 포함됩니다.

이런 방식으로, TON 블록체인 계정의 일부 소유자의 추상 네트워크 주소를 찾는 메커니즘을 사용할 수 있습니다.

3.2.14. 추상주소 찾기 (Locating abstract addresses). TON DHT는 TON ADNL을 통해 구현되는 동안 TON ADNL에 의해 여러 용도로 사용됩니다. 가장 중요한 것은 256-비트 추상주소에서 시작하여 노드 또는 그 연락처 데이터를 찾는 것입니다. 이는 추가 정보가 제공되지 않더라도 TON ADNL 임의의 256-비트 추상주소로 데이터그램을 전송할 수 있어야 하기 때문에 필요합니다.

이를 위해 256-비트 추상주소는 단순히 DHT에서 키로 조회됩니다. 이 주소를 가진 노드 (즉, 이 주소를 공개적으로 반영구적인 DHT 주소로 사용)가 발견되는 경우, IP 주소와 포트를 알 수 있습니다; 또는, 입력 터널 설명은 올바른 프라이빗 키로 서명된 해당 키의 값으로 검색될 수 있으며, 이 경우 이 터널 설명은 의도된 수신자에게 ADNL 데이터그램을 보내는데 사용됩니다.

추상적인 주소를 "퍼블릭" (네트워크상의 어떤 노드에서도 접근 가능)하게 만들기 위해, 그것의 소유자는 그것을 반영구적인 DHT 주소로 사용하거나 (고려중인 추상주소와 동일한 DHT 키에) 터널의 진입점으로 다른 공개 추상주소 (예컨대, 반영구적 주소)와 함께 입력 터널 설명을 게시해야 합니다. 또 다른 옵션은 단순히 IP 주소와 UDP 포트를 게시하는 것입니다.

3.3 오버레이 네트워크 및 멀티캐스팅 메시지 (Overlay Networks and Multicasting Messages)

TON 블록체인과 같은 멀티-블록체인 시스템에서, 풀-노드조차도 일반적으로 일부 샤드체인에 대해서만 업데이트 (즉, 새로운 블록)를 얻는데 관심이 있습니다.
이를 위해 샤드체인마다 하나씩, **3.1**에서 논의된 ADNL 프로토콜 위에 TON 네트워크 내부에 특수 오버레이 (서브) 네트워크를 구축해야 합니다.

따라서 참여하고자 하는 어떤 노드에게 개방된 임의의 오버레이 서브네트워크를 구축해야 할 필요성이 발생합니다. ADNL을 기반으로 만들어진 특별한 가십 프로토콜이 이러한 오버레이 네트워크에서 실행됩니다. 특히, 이러한 가십 프로토콜은 서브 네트워크 내에서 임의의 데이터를 전파 (브로드캐스트)하는데 사용될 수 있습니다.

3.3.1. 오버레이 네트워크 (Overlay networks). 오버레이 (서브)네트워크 는 단순히 더 큰 네트워크 내에 구현된 (가상) 네트워크입니다. 일반적으로 큰 네트워크의 일부 노드만이 오버레이 서브네트워크에 참여하며 이러한 노드 사이의 일부 "링크" (물리적 또는 가상)가 오버레이 서브네트워크의 일부입니다.

이와 같이 포괄적인 네트워크가 그래프로 표현되면 (예. ADNL과 같은 데이터그램 네트워크의 경우 모든 노드에서 다른 노드와 쉽게 통신할 수 있는 전체 그래프) 오버레이 서브네트워크는 이 그래프의 서브그래프입니다.

대부분의 경우, 오버레이 네트워크는 큰 네트워크의 네트워크 프로토콜을 기반으로 구축된 일부 프로토콜을 사용하여 구현됩니다. 큰 네트워크와 동일한 주소를 사용하거나 사용자 정의 주소를 사용할 수 있습니다.

3.3.2. TON의 네트워크 오버레이 (Overlay networks in TON). TON의 오버레이 네트워크는 **3.1**에서 논의된 ADNL 프로토콜을 기반으로 구축됩니다; 그들은 256-비트 ADNL 추상주소를 오버레이 네트워크의 주소로 사용합니다. 각 노드는 일반적으로 오버레이 네트워크의 주소 중 하나를 선택하여 이를 추상주소만큼 두 배로 만듭니다.

ADNL과 달리 TON 오버레이 네트워크는 일반적으로 임의의 다른 노드에 데이터그램을 보내는 것을 지원하지 않습니다. 그 대신, 일부 "반영구 링크"가 일부 노드간에 설정되고 (일반적으로 고려 대상인 오버레이 네트워크에 대해 "이웃"이라고 함) 메시지는 일반적으로 이러한 링크를 따라 전달됩니다 (즉, 한 노드에서 이웃 노드 중 하나까지). 이러한 방식으로 TON 오버레이 네트워크는 ADNL 네트워크의 (전체) 그래프 안에 있는 (보통 전체가 아닌) 서브그래프입니다.

TON 오버레이 네트워크의 이웃들과의 링크는 전용 피어-투-피어 ADNL 채널을 사용하여 구현할 수 있습니다 (cf. **3.1.5**).

오버레이 네트워크의 각 노드는 자신의 추상주소 (오버레이 네트워크에서 식별하기 위해 사용하는)와 일부 링크 데이터 (예. 그들과 통신하는데 사용되는 ADNL 채널)를 포함하는 이웃 목록 (오버레이 네트워크 관련)을 유지합니다.

3.3.3. 프라이빗 및 퍼블릭 오버레이 네트워크 (Private and public overlay networks). 일부 오버레이 네트워크는 퍼블릭(공개)이므로, 모든 노드가 자유롭게 가입할 수 있습니다. 다른것은 프라이빗(비공개)이며, 특정 노드만 허용될 수 있습니다 (예. 밸리데이터로서의 ID를 증명할 수 있는 노드). 일부 프라이빗 오버레이 네트워크는 "일반 대중"에게 알려지지 않을 수도 있습니다. 이러한 오버레이 네트워크에 대한 정보는 특정 신뢰할 수 있는 노드에서만 사용할 수 있습니다; 예를들어 프라이빗 키로 암호화 될 수 있으며 해당 프라이빗 키의 사본이 있는 노드만이 정보를 암호 해독할 수 있습니다.

3.3.4. 중앙에서 제어되는 오버레이 네트워크 (Centrally controlled overlay networks). 일부 오버레이 네트워크는 하나 또는 여러 개의 노드에 의해 중앙에서 제어되거나 널리 알려진 프라이빗 키의 소유자에 의해 제어됩니다. 다른 것들은 분산화되어 있으며, 이것은 그들을 책임지는 특정한 노드가 존재하지 않는다는 것을 의미합니다.

3.3.5. 오버레이 네트워크 연결 (Joining an overlay network). 노드가 오버레이 네트워크에 참여하기를 원할 때, 먼저 오버레이 네트워크에 대한 설명의 SHA256과 같은 256-비트 네트워크 식별자, 즉 TL 직렬화된 객체 (예. cf. **2.2.5**)를 알아야합니다. 이 객체는 예를 들어 오버레이 네트워크의 중앙권한 (즉, 공개 키와 추상 주소[36]), 오버레이 네트워크의 이름이 포함된 문자열, 해당 샤드와 관련된 오버레이 네트워크인 경우 TON 블록체인 샤드 식별자 등이 포함됩니다.

때로는 TON DHT에서 네트워크 식별자를 조회하는 것만으로 네트워크 식별자에서 시작하는 오버레이 네트워크 설명을 복구할 수 있습니다. 다른 경우 (예. 프라이빗 오버레이 네트워크의 경우) 네트워크 식별자와 함께 네트워크 기술을 획득해야 합니다.

3.3.6. 오버레이 네트워크의 한 구성원 찾기 (Locating one member of the overlay network). 노드가 네트워크 식별자와 결합하려는 오버레이 네트워크의 네트워크 설명을 학습한 후에는 해당 네트워크에 속한 하나 이상의 노드를 찾아야 합니다.

이것은 오버레이 네트워크에 참여하기를 원하지 않지만 노드와 통신하기를 원하는 노드에도 필요합니다; 예를 들어 특정 샤드체인에 대한 트랜잭션 후보를 수집하고 전파하는데 사용되는 오버레이 네트워크가 있을 수 있으며 클라이언트는 이 네트워크의 노드에 연결하여 트랜잭션을 제안하려고 할 수 있습니다.오버레이 네트워크의 멤버를 찾는데 사용되는 방법은 해당 네트워크의 설명에 정의되어 있습니다. 때로는 (특히 프라이빗 네트워크의 경우) 멤버 노드를 이미알고 있어야만 가입할 수 있습니다. 다른 경우에는 일부 노드의 추상주소가 네트워크 설명에 포함됩니다. 좀 더 유연한 접근 방식은 네트워크 설명에서 네트워크에대한 책임이 있는 중앙기관만 표시한 다음 해당 중앙권한에 의해 서명된 특정 DHT 키의 값을 통해 추상주소를 사용할 수 있게 하는 것입니다.

마지막으로 진정한 분산형 퍼블릭 오버레이 네트워크는 TON DHT의 도움으로 구현된 **3.2.10**에 설명된 "분산형 토렌트 추적기" 메커 니즘을 사용할 수 있습니다.

3.3.7. 오버레이 네트워크의 멤버 더 찾기. 링크 만들기 (Locating more members of the overlay network. Creating links). 오버레이 네트워크의 하나의 노드가 발견되면, 다른 멤버의 리스트, 예를 들어, 질의되는 노드의 이웃, 또는 그것의 무작위 선택과 같은 요청을 그 노드로 전송할 수 있습니다.

이를 통해 가입 회원은 일부 새로 알게된 일부 네트워크 노드를 선택하고 해당 네트워크에 대한 링크를 설정하여 오버레이 네트워크와 관련하여 자신의 "인접성" 또는 "이웃목록"을 채울 수 있습니다 (즉, ADNL 포인트-투-포인트 전용 채널. cf. **3.3.2**). 그 후, 새로운 멤버가 오버레이 네트워크에서 작업할 준비가 되었음을 알리는 특별한 메시지가 모든 이웃에게 전송됩니다. 이웃에는 이웃목록에 있는 새 회원에 대한 링크가 포함됩니다.

3.3.8. 이웃리스트 유지 (Maintaining the neighbor list). 오버레이 네트워크 노드는 이웃리스트를 수시로 업데이트해야 합니다. 일부 이웃 또는 적어도 그들에 대한 링크 (채널) 가 응답하지 않을 수 있습니다. 이 경우, 이러한 링크는 "중지됨"으로 표시되어야 하며, 그러한 이웃과의 재접속을 시도해야 하며, 이러한 시도가 실패하면 링크를 파괴해야 합니다.

다른 한편, 모든 노드는 때로는 임의로 선택된 이웃으로부터 자신의 이웃 목록 (혹은 약간의 무작위 선택)을 요청하고, 일부 새롭게 발견된 노드를 추가하고 이전의 일부 노드를 무작위로 또는 응답 시간과 데이터그램 손실 통계에 따라 그 중 일부를 제거함으로써 자체 이웃 목록을 부분적으로 업데이트합니다.

3.3.9. 오버레이 네트워크는 임의의 서브그래프입니다 (The overlay network is a random subgraph). 이러한 방식으로 오버레이 네트워크는 ADNL 네트워크 내에서 임의의 서브그래프가 됩니다. 각 꼭지점의 차수가 적어도 3 이상이면 (즉, 각 노드가 적어도 3 개의 이웃 노드에 연결되어 있는 경우), 이 임의의 그래프는 거의 1과 동일한 확률로 연결되는 것으로 알려져 있습니다. 더 정확하게 말하면, n 개의 꼭지점이 끊어진 임의의 그래프의 확률은 기하 급수적으로 작으며 만약 $n \geq 20$ 이면 이 확률은 완전히 무시할 수 있습니다 (물론, 글로벌 네트워크 구획의 경우 파티션의 다른 측면에 있는 노드가 서로에 대해 알 수 있는 기회가 없습니다). 반면에, n 이 20 보다 작다면, 각 꼭지점에는 적어도 10개의 이웃을 갖도록 요구하면 충분할 것입니 다.

3.3.10. 낮은 대기시간에 최적화된 TON 오버레이 네트워크 (TON overlay networks are optimized for lower latency). TON 오버레이 네트워크는 다음과 같이 이전 방법으로 생성된 "임의의"네트워크 그래프를 다음과 같이 최적화 합니다. 모든 노드는 "빠른 이웃"목록을 거의 변경하지 않고 최소한의 왕복시간으로 최소 3명의 이웃을 유지하려고 시도합니다. 동시에는 오버레이 네트워크 그래프에 항상 임의의 서브그래프가 포함될 수 있도록 완전히 다른 임의의 최소 3개의 다른 “느린 이웃"이 선택됩니다. 이는 연결을 유지하고 오버레이 네트워크가 여러개의 연결되지 않은 지역 서브네트워크로 분할되는 것을 방지하는데 필요합니다. 특정상수 (실제로는 빠르고 느린 이웃의 왕복시간의 함수)로 묶인 중간 왕복시간을 갖는 적어도 3개의 "중간 이웃"도 선택되어 유지됩니다.

이런 식으로, 오버레이 네트워크의 그래프는 여전히 연결될 수 있는 임의성을 유지하면서 대기시간과 처리량을 높이기 위해 최적화되어 있습니다.

3.3.11. 오버레이 네트워크의 가십 프로토콜 (Gossip protocols in an overlay network). 오버레이 네트워크는 소위, 가십 프로토콜 중 하나를 실행하는데 종종 사용되며 모든 글로벌 목표를 달성하면서 모든 노드가 이웃 노드와만 상호 작용하도록 합니다. 예를 들어, 각 노드에서 제한된 양의 메모리만 사용하여 (너무 크지 않은) 오버레이 네트워크의 모든 구성원의 대략적인 목록을 구성하거나, 또는 (임의로 큰) 오버레이 네트워크의 구성원 수를 계산하는데 사용되는 가십 프로토콜이 있습니다 (자세한 것은 cf. [**22, 4.4.3**], [**3**]).

3.3.12. 오버레이 네트워크를 브로드캐스트 도메인으로 (Overlay network as a broadcast domain). 오버레이 네트워크에서 실행되는 가장 중요한 가십 프로토콜은 브로드캐스트 프로토콜로, 네트워크의 노드 또는 지정된 발신자자 노드 중 하나에 의해 생성된 브로드캐스트 메시지를 다른 모든 노드로 전파하도록 되어 있습니다.

사실 여러가지 사용예에 맞게 최적화된 몇 가지 브로드캐스트 프로토콜이 있습니다. 그중 가장 간단한 프로토콜은 새로운 브로드캐스트 메시지를 수신하고 자신이 보낸 메시지 사본을 아직 보내지 않은 모든 이웃에게 전달합니다.

3.3.13. 더 정교한 브로드캐스트 프로토콜(More sophisticated broadcast protocols). 일부 애플리케이션은 보다 정교한 브로드캐스트 프로토콜을 보증할 수 있습니다. 예를 들어, 상당한 크기의 브로드캐스팅 메세지는 새롭게 수신된 메시지 자체가 아니라 이 메시지의 해시 (또는 새 메시지의 해시 모음)를 이웃에게 보내는 것이 좋습니다.

이웃 노드는 이전에 보이지 않는 메시지 해시를 학습한 후에 메시지 자체를 요청할 수 있으며, 예를 들어, **3.1.9**에서 논의된 신뢰할 수 있는 대형 데이터그램 프로토콜 (RLDP)을 사용하여 전송할 수 있습니다. 이 경우, 새 메시지는 하나의 이웃에서만 다운로드됩니다.

3.3.14. 오버레이 네트워크의 연결 확인 (Checking the connectivity of an overlay network). 오버레이 네트워크의 연결은 오버레이 네트워크 내에 있어야하는 알려진 노드 (예를 들어, 오버레이 네트워크의 "소유자" 또는 "생성자")가 존재 하는지로 점검할 수 있습니다. 그런 다음 문제의 노드는 현재시간, 배열번호 및 서명을 포함하는 짧은 메시지를 때때로 브로드캐스트합니다. 다른 노드는 얼마전에 그러한 브로드캐스트를 수신했다면 여전히 오버레이 네트워크에 연결되어 있는지 확인할 수 있습니다. 이 프로토콜은 여러 잘 알려진 노드의 경우까지 확장될 수 있습니다; 예를 들어, 그들이 모두 그러한 브로드캐스트를 보내면 다른 모든 노드는 잘 알려진 노드의 절반 이상으로부터 브로드캐스트를 수신할 것으로 예상됩니다.

특정 샤드체인의 새 블록 (또는 오직 새 블록 헤더)을 전파하는데 사용되는 오버레이 네트워크의 경우 노드가 연결을 확인하는 좋은 방법은 지금까지 받은 가장 최근의 블록을 추적하는 것입니다. 블록은 일반적으로 5초마다 생성되기 때문에 예를 들어 30초를 초과하였는데도 새로운 블록이 수신되지 않으면 노드가 아마도 오버레이 네트워크에서 연결이 끊어진 것입니다.

3.3.15. 브로드캐스트 프로토콜 스트리밍 (Streaming broadcast protocol). 마지막으로, TON 오버레이 네트워크를 위한 스트리밍 브로드캐스트 프로토콜이 있습니다. 예를 들어 일부 샤드체인 ("샤드체인 작업그룹")의 밸리데이터 중에서 블록 후보를 전파하는데 사용되며, 물론 그 목적을 위해 프라이빗 오버레이 네트워크를 만듭니다. 동일한 프로토콜을 사용하여 샤드체인에 대한 모든 풀-노드에 새 샤드체인 블록을 전파할 수 있습니다.

이 프로토콜은 이미 **2.6.10**에 요약되어 있습니다. 새로운 (큰) 브로드캐스트 메시지는, 예를 들어 N개의 1 킬로바이트 청크로 나뉩니다. 이러한 청크의 배열은 리드-솔로몬이나 파운틴 코드와 같은 삭제 코드를 사용하여 $M \geq N$ 청크로 증가되며 (예. RaptorQ 코드 [15] [21] 또는 온라인 코드 [17]), M 개의 청크는 모든 이웃들에게 오름차순의 청크번호 순서로 스트리밍 됩니다. 참여 노드는 본래 큰 메시지를 복구할 수 있을 때까지 이 청크를 수집하고 (이것을 위해 적어도 N개의 청크를 성공적으로 수신해야 합니다) 이웃 노드에게 스트림의 새로운 청크 전송을 중지하도록 지시합니다. 이제는 이러한 노드가 원래 메시지의 복사본을 가지고 다음 청크를 스스로 생성할 수 있기 때문입니다. 이러한 노드는 스트림의 후속 청크를 계속 생성하여 이웃 노드가 더 이상 필요하지 않다는 것을 표시하지 않는 한 이웃 노드로 전송합니다.

이런 방식으로, 노드는 더 큰 메시지를 전파하기 전에 전체 메시지를 다운로드 할 필요가 없습니다. 이는 특히 **3.3.10**에 설명된 최적화와 결합할 때 브로드캐스트 대기 시간을 최소화합니다.

3.3.16. 기존 오버레이 네트워크를 기반으로 새로운 오버레이 네트워크 구축 (Constructing new overlay networks based on existing ones). 가끔은 처음부터 오버레이 네트워크를 구축하기를 원하지 않습니다. 그 대신 하나 또는 여러개의 기존 오버레이 네트워크가 알려져 있으며 새로운 오버레이 네트워크의 멤버십은 이 오버레이 네트워크의 결합된 멤버십과 크게 겹칠것으로 예상됩니다.

중요한 예는 TON 샤드체인이 두 개로 나뉘거나, 혹은 두 개의 형제 샤드체인이 하나로 합쳐지면 발생합니다 (cf. **2.7**). 첫 번째 경우에 새로운 블록을 풀-노드에전파하는데 사용되는 오버레이 네트워크는 새로운 각 샤드체인을 위해 구성되어야 합니다; 그러나 이러한 새로운 각 오버레이 네트워크는 원래 샤드체인의 블록 전파 네트워크에 포함될 것으로 예상될 수 있습니다. 두 번째 경우에는 병합된 샤드체인의 새 블록을 전파하기 위한 오버레이 네트워크가 대략 병합되는 두 형제 샤드체인과 관련된 두 개의 오버레이 네트워크 멤버의 조합으로 구성됩니다.

그러한 경우 새로운 오버레이 네트워크에 대한 설명은 관련된 기존의 오버레이 네트워크 목록에 대한 명시적 또는 암시적 참조를 포함할 수 있습니다. 새로운오버레이 네트워크에 참여하기를 원하는 노드는 그것이 이미 기존의 네트워크 중하나의 멤버인지 여부를 검사하고 새로운 네트워크에도 관심이 있는지를 이 네트워크의 이웃에게 질문할 수 있습니다. 긍정 응답의 경우, 새로운 이웃에 대해 새로운 피어-투-피어 채널이 설정될 수 있고, 이들은 새로운 오버레이 네트워크에 대한이웃 목록에 포함될 수 있습니다.
이 메커니즘은 **3.3.6**및 **3.3.7**에 설명된 일반적인 메커니즘을 완전히 대체하지는 못합니다. 오히려 둘 다 병렬로 실행되고 이웃목록을 채우는데 사용됩니다. 이는 새로운 오버레이 네트워크를 실수로 여러개 의 연결되지 않은 서브네트워크로 분할하는 것을 방지하기 위해 필요합니다.

3.3.17. 오버레이 네트워크 내의 오버레이 네트워크 (Overlay networks within overlay networks). 또 다른 흥미로운 사례는 *TON* 페이먼트 (즉각적인 오프-체인 값 전송을 위한 "라이트닝 네트워크"; cf. **5.2**) 구현에서 발생합니다. 이 경우, 먼저 "라이트닝 네트워크"의 모든 중계 노드를 포함하는 오버레이 네트워크가 구축됩니다.

그러나, 이러한 노드 중 일부는 블록체인에 결제채널을 설정했습니다; **3.3.6, 3.3.7** 및 **3.3.8**에 설명된 일반적인 오버레이 네트워크 알고리즘에 의해 선택된 "임의의" 이웃뿐만 아니라 항상 이 오버레이 네트워크의 이웃이어야 합니다. 확립된 결제채널을 가진 이웃들에 대한 이러한 "영구링크"는 특정 라이트닝 네트워크 프로토콜을 실행하는데 사용되므로 포괄적인 (거의 항상 연결된) 오버레이 네트워크 내에 효과적으로 오버레이 서브네트워크 (일이 잘못되면 반드시 연결되어 있지 않아도 됩니다) 를 생성합니다.

4 TON 서비스 및 애플리케이션 (TON Services and Applications)

우리는 TON 블록체인 및 TON 네트워킹 기술에 대해 어느 정도 논의했습니다. 지금부터는 다양한 서비스와 애플리케이션을 만들기 위해 이들을 결합할 수 있는 몇 가지 방법을 설명하고 TON 프로젝트 자체에서 처음부터 또는 나중에 지원될 서비스의 일부 또는 전체에 대해서 논의합니다.

4.1 TON 서비스 구현 전략 (TON Service Implementation Strategies)

TON생태계 내에서 블록체인 및 네트워크 관련 애플리케이션 및 서비스가 어떻게 구현될 수 있는지에 대한 논의부터 시작합니다. 우선, 간단한 분류가 순서대로 이루어집니다:

4.1.1. 애플리케이션 및 서비스 (Applications and services). 우리는 "애플리케이션"과 "서비스"라는 단어를 같은 의미로 사용합니다. 그러나 미묘하고 다소 모호한 구별이 있습니다. 애플리케이션은 일반적으로 인간 사용자에게 직접 서비스를 제공하지만, 서비스는 일반적으로 다른 애플리케이션 및 서비스에 의해 활용됩니다. 예를 들어, TON 저장소는 사람이 직접 애플리케이션 및 서비스를 사용할 수 있는 경우에도 파일을 다른 애플리케이션 및 서비스 대신 보관하도록 설계 되었기 때문에 "서비스"입니다. TON 네트워크를 통해 사용 가능하다면 (즉, "톤 서비스"로 구현, cf. **4.1.6**) 가상의 "블록 체인 내의 페이스북" (cf. **2.9.13**) 또는 텔레그램 메신저는, 오히려 애플리케이션이 될 것이며, 일부 "봇(bots)"이 사람의 개입없이 자동으로 액세스할 수 있습니다.

4.1.2. 애플리케이션의 위치: 온-체인, 오프-체인 또는 혼합 (Location of the application: on-chain, off-chain or mixed). TON 생태계를 위해 고안된 서비스나 애플리케이션은 데이터를 보관하고 해당 데이터를 어딘가에서 처리해야 합니다. 이는 다음과 같은 애플리케이션 (및 서비스)의 분류로 이어집니다.

- 온-체인 (*On-chain*)애플리케이션 (cf.**4.1.4**): 모든 데이터와 처리는 TON블록체인에있습니다.
- 오프-체인 (*Off-chain*) 애플리케이션 (cf.**4.1.5**): 모든 데이터 및 처리는TON네트워크를통해 사용 가능한 서버의 TON 블록체인 외부에 있습니다.
- 혼합 (*Mixed*) 애플리케이션 (cf. **4.1.7**): 다는 아니지만 일부 데이터와 처리는 TON 블록체인에 있습니다; 나머지는 TON 네트워크를 통해 사용할 수있는 오프-체인 서버에있습니다

4.1.3. 중앙 집중화: 중앙형 및 탈중앙형 또는 분산형 애플리케이션 (Centralization: centralized and decentralized, or distributed applications). 또 다른 분류 기준은 애플리케이션 (또는 서비스)이 중앙 집중식 서버 클러스터에 의존하는지 아니면 실제로 "분산"되어 있는지 (cf. **4.1.9**)입니다. 모든 온-체인 애플리케이션은 자동으로 분산되어 배포됩니다. 오프-체인 및 혼합 애플리케이션은 다른 정도의 중앙 집중화를 나타낼 수 있습니다.

이제 위의 가능성을 보다 자세히 고려해 보겠습니다.

4.1.4. 순수 "온-체인" 애플리케이션 : 블록체인에 있는 분산된 애플리케이션 또는 "댑" (Pure “on-chain” applications: distributed applications, or “dapps”, residing in the block-chain). **4.1.2**에서 언급된 가능한 접근 방법 중 하나는, "분산 애플리케이션" (일반적으로 "댑"으로 약칭함)을 TON 블록체인에 완전히 하나의 스마트 컨트랙트 또는 스마트 컨트랙트 모음으로 배포하는 것입니다. 모든 데이터는 이러한 스마트 컨트랙트의 영구적인 상태의 일부로 유지되며 프로젝트와의 모든 상호작용은 이러한 스마트 컨트랙트와 주고받는 메시지 (TON 블록체인)를 통해 수행됩니다.

우리는 이미 **2.9.13** 에서 이 접근법이 단점과 한계점을 가지고 있다고 논의했습니다. 이는 장점도 있습니다: 예를 들어, 이러한 분산 애플리케이션은 데이터를 실행하거나 저장하는데 서버가 필요하지 않고 ("블록체인"으로 실행됨 — 즉, 밸리데이터의 하드웨어) 블록체인의 매우 높은 (비잔틴) 신뢰성과 접근성을 누릴 수 있습니다. 이러한 분산 애플리케이션의 개발자는 하드웨어를 구입하거나 임대할 필요가 없습니다. 그가 해야할 일은 소프트웨어 (즉, 스마트 컨트랙트를 위한 코드)를 개발하는 것 뿐입니다. 그 후에, 그는 밸리데이터로부터 효과적으로 컴퓨팅 파워를 임대할 것이고, 그 비용을 그램으로 스스로 또는 사용자들의 부담으로 지불할 것입니다.

4.1.5. 순수 네트워크 서비스: "톤-사이트" 및 "톤-서비스" (Pure network services: “ton-sites” and “ton-services”). 또 다른 극단적인 옵션은 일부 서버에 서비스를배포하고 **3.1**에서 설명한 ADNL 프로토콜을 통해 사용자가 사용할 수 있도록 하는것입니다. 또한 혹은 **3.1.9**에서 논의된 RLDP와 같은 더 높은 수준의 프로토콜을 사용하여 RPC 쿼리를 사용자 지정 형식으로 서비스에 보내고 이러한 쿼리에 대한 응답을 얻는데 사용할 수 있습니다. 이런 식으로, 이 서비스는 완전히 오프-체인이고 거의 TON 블록체인을 사용하지 않고 TON 네트워크 내에 상주하게 될 것입니다.

TON 블록체인은 아마도 도메인 같이 사람이 읽을 수 있는 문자열을 추상주소로 쉽게 변환할 수 있도록 TON DNS (cf. **4.3.1**)와 같은 서비스의 도움을 받아 **3.2.12**에 요약된 것처럼 서비스의 추상주소 또는 주소를 찾기 위해서만 사용될 수 있습니다.

ADNL 네트워크 (예. TON 네트워크)가 I^2P 와 비슷한 정도로 순수한 네트워크 서비스는 소위 "eep-services"와 유사합니다 (즉, I^2P 주소를 진입점으로하며 I^2P 네트워크를 통해 클라이언트에서 사용할 수 있습니다). 우리는 TON 네트워크에 거주하는 순수한 네트워크 서비스를 "톤 서비스"라고 말할 것입니다.

"eep-service"는 HTTP를 클라이언트-서버 프로토콜로 구현할 수 있습니다. TON 네트워크 컨텍스트에서 "톤 서비스"는 HTTP 쿼리와 응답을 전송하기 위해 단순히 RLDP (cf. **3.1.9**) 데이터그램을 사용할 수 있습니다. 사람이 읽을 수있는 도메인 이름으로 추상주소를 검색할 수 있도록 TON DNS를 사용하는 경우 웹 사이트에 대한 비유가 거의 완벽해집니다. 심지어 사용자의 컴퓨터에서 로컬로 실행되는 특수 브라우저 또는 특수 프록시 ("톤-프록시")를 작성하고 사용자가 사용하는 일반 웹 브라우저에서 임의의 HTTP 쿼리를 허용하여 (로컬 IP 주소와 프록시의 TCP 포트가 브라우저의 구성에 입력되면) 이러한 쿼리를 TON 네트워크를 통해 서비스의 추상주소로 전달합니다. 그러면 사용자는 월드 와이드 웹(WWW)과 유사한 브라우징 경험을 갖게 됩니다.

I^2P 생태계에서는 이러한 "eep-services"를 "eep-sites"라고 부릅니다. TON 생태계에서도 쉽게 "톤-사이트"를 만들 수 있습니다. 이것은 TON 블록체인과 TON DHT를 이용하여 도메인 이름을 추상주소로 변환하는 TON DNS와 같은 서비스의 존재로 인해 약간 촉진됩니다.

4.1.6. 텔레그램 메신저로 톤-서비스; RLDP를 통한 MTProto (Telegram Messenger as a ton-service; MTProto over RLDP). 텔레그램 메신저가 클라이언트-서버 상호 작용을 위해 사용하는 MTProto 프로토콜[37]이 **3.1.9**에서 논의된 RLDP 프로토콜에 쉽게 임베드되어 텔레그램을[38] 톤-서비스로 효과적으로 변환할 수 있다는 점을 언급하고자 합니다. TON 프록시 기술은 RLDP 및 ADNL 프로토콜 (cf. **3.1.6**)보다 낮은 수준에서 구현되는 톤-사이트 또는 톤-서비스의 최종 사용자를 위해 투명하게 전환될 수 있으므로 이는 텔레그램을 효과적으로 차단해제할 수 있습니다. 물론 다른 메시징 및 소셜 네트워킹 서비스도 이 기술의 이점을 누릴 수 있습니다.

[37] https://core.telegram.org/mtproto
[38] https://telegram.org/

4.1.7. 혼합 서비스: 부분적으로 오프-체인, 부분적으로 온-체인 (Mixed services: partly off-chain, partly on-chain). 일부 서비스는 혼합된 접근 방법을 사용할 수 있습니다. 대부분의 처리를 오프-체인으로 처리하지만 일부 온-체인 부분 또한 포함합니다 (예를 들어, 사용자에 대한 의무를 등록하고, 그 반대의 경우도 마찬가지입니다). 이런 방식으로, 상태의 일부는 여전히 TON 블록체인 (즉, 불변의 퍼블릭 장부)에 보관될 것이고, 서비스 또는 사용자의 부정행위는 스마트 컨트랙트에 의해 처벌될 수 있습니다.

4.1.8. 예: 파일을 오프-체인에 보관; TON 저장소 (Example: keeping files off-chain; TON Storage). 이러한 서비스의 예로 TON 저장소가 있습니다. 가장 간단한 형태로, 사용자는 저장된 파일의 해시만 온-체인을 유지함으로써 파일을 오프-체인으로 저장할 수 있으며, 아마도 사전 협상된 비용으로 특정기간 동안 파일을 계속 유지하기로 하는 다른 당사자들의 동의가 있는 스마트 컨트랙트 또한 보관할 수 있습니다. 실제로, 파일은 약간의 작은 크기 (예를 들어, 1 킬로바이트)의 청크로 분할되거나, 리드-솔로몬 또는 파운틴 코드와 같은 삭제코드에 의해 증가될 수 있고, 머클트리 해시는 증가된 청크 배열을 위해 구성될 수 있습니다. 이 머클트리 해시는 파일의 일반적인 해시 대신 또는 스마트 컨트랙트에 게시될 수 있습니다. 이것은 파일이 토렌트에 저장되는 방식을 연상케합니다.

파일을 저장하는 훨씬 더 단순한 형태는 완전한 오프-체인입니다: 새로운 파일을 위해 "토렌트"를 만들고 TON DHT를 이 토렌트의 "분산형 토렌트 트래커"로 사용하는 것입니다 (cf. **3.2.10**). 이것은 실제로 대중적인 파일에 꽤 잘 작동할 것입니다. 그러나 가용성을 보장 받지 못합니다. 예를 들어, 사용자의 프로필 사진을 이러한 "토렌트"에서 완전히 오프-체인 상태로 유지하려는 가상의 "블록체인 페이스북" (cf. **2.9.13**)은 일반 사용자 (별로 인기가 없는)의 사진을 잃을 위험이 있거나 또는 적어도 이러한 사진을 오랜 기간 동안 제공하지 못할 위험이 있습니다. TON 저장소 기술은 대부분 오프-체인이지만 저장된 파일의 가용성을 강화하기 위해 온-체인 스마트 컨트랙트를 사용하는 것이 이 작업에 더 적합할 수 있습니다.

4.1.9. 분산형 혼합 서비스 또는 "안개서비스" (Decentralized mixed services, or “fog services”). 지금까지 우리는 중앙형 혼합 서비스 및 애플리케이션에 대해 논의했습니다. 온-체인 구성 요소가 탈중앙형과 분산형 방식으로 처리되는 반면 블록체인에 있기 때문에 오프-체인 구성 요소는 일반적인 중앙형 방식으로 서비스 공급자가 통제하는 일부 서버에 의존합니다. 일부 전용서버를 사용하는 대신 대형기업 중 하나에서 제공하는 클라우드 컴퓨팅 서비스에서 컴퓨팅 파워를 빌릴 수 있습니다. 그러나 이것은 서비스의 오프-체인 구성 요소의 분산화로 이어지지 않을 것입니다.

서비스의 오프-체인 구성요소를 구현하는 분산형 접근 방식은 필요한 하드웨어를 소지하고 컴퓨팅 성능이나 디스크 공간을 임대하려는 사람에게 필요한 서비스를 제공하는 시장을 만드는 것입니다.

예를 들어, 다른 사용자의 파일을 보관하려는 모든 노드가 사용 가능한 저장 용량, 가용성 정책, 가격 및 연락처 정보를 게시하는 레지스트리 ("시장" 또는 "거래소"라고도 함)가 있을 수 있습니다. 이러한 서비스를 필요로하는 사용자는 해당 사이트를 검색하고 상대방이 동의하면 블록체인에 스마트 컨트랙트를 만들고 오프-체인 저장소에 파일을 업로드 할 수 있습니다. 이러한 방식으로, TON 저장소와 같은 서비스는 파일 저장을 위한 중앙형 서버 클러스터에 의존할 필요가 없으므로 진정으로 분산화 됩니다.

4.1.10. 예: 분산형 혼합 서비스로서의 "안개계산" 플랫폼 (Example: “fog computing” platforms as decentralized mixed services). 이러한 분산형 혼합 애플리케이션의 또 다른 예는 특정한 계산 (예. 3D 렌더링 또는 트레이닝 뉴런 네트워크)을 수행하기를 원할 때 발생하며, 종종 특정한 고가의 하드웨어를 필요로 합니다. 이런 장비를 가진 사람들은 유사한 "거래소"를 통해 그들의 서비스를 제공할 수 있으며 그러한 서비스를 필요로 하는 사람들은 스마트 컨트랙트로 등록된 측의 의무와 함께 이것을 빌릴 것입니다. 이것은 골렘 (https://golem.network/) 또는 SONM (https://sonm.io/)과 같은 "안개계산" 플랫폼이 제공할 것으로 약속한 것과 유사합니다.

4.1.11. 예: 안개서비스 TON 프록시 (Example: TON Proxy is a fog service). TON 프록시 는 ADNL 네트워크 트래픽을 위한 터널처럼 서비스를 제공하고자 하는 노드 (보상 유무와 상관없이)가 등록할 수 있는 안개서비스의 또 다른 예를 제공합니다. 그(것)들을 필요로하는 사람들은 제안된 가격, 대기시간 및 대역폭에 따라 이러한 노드 중 하나를 선택할 수 있습니다. 이후에, *TON* 페이먼트가 제공한 결제채널을 사용하여, 예를 들어, 128 KiB가 양도될 때마다 결제된 금액과 함께 해당 프록시의 서비스에 대한 소액지불을 처리할 수 있습니다.

4.1.12. 예: TON 페이먼트는 안개서비스 (Example: TON Payments is a fog service). TON 페이먼트 플랫폼 (cf. **5**)은 분산형 혼합 애플리케이션의 예이기도 합니다.

4.2 사용자와 서비스 제공자 연결 (Connecting Users and Service Providers)

4.1.9에서 볼 수 있듯이 "안개서비스” (즉, 혼합 분산형 서비스)에는 특정 서비스가 필요한 시장, 거래소 또는 레지스트리 가 필요합니다.

이러한 시장은 온-체인, 오프-체인 또는 혼합 서비스 자체, 중앙형 또는 분산형으로 구현될 가능성이 큽니다.

4.2.1. 예: TON 페이먼트에 연결 (Example: connecting to TON Payments). 예를 들어, TON 페이먼트 (cf. **5**)를 사용하려면, 첫 번째 단계는 "라이트닝 네트워크" (cf. **5.2**)의 기존 중계노드를 찾아 내고 그들이 원하는 경우 결제채널을 수립하는 것입니다. 일부 노드는 모든 중계 라이트닝 네트워크 노드를 포함하는 것으로 간주되는 "포괄적인" 오버레이 네트워크를 사용하여 찾을 수 있습니다 (cf. **3.3.17**). 그러나 이러한 노드가 새로운 결제채널을 기꺼이 만들 것인지 여부는 명확하지 않습니다. 따라서, 새로운 링크를 생성할 준비가 된 노드가 자신의 연락처 정보 (예를 들어, 그들의 추상 주소)를 공개할 수 있는 레지스트리가 필요합니다.

4.2.2. 예: TON 저장소에 파일 업로드 (Example: uploading a file into TON Storage). 마찬가지로, 파일을 TON 저장소에 업로드 하려면 스마트 컨트랙트에 서명하여 해당 파일 (또는 특정 크기 제한 이하의 파일)의 사본을 보관할 노드를 찾아야 합니다. 따라서, 파일 저장을 위한 서비스를 제공하는 노드의 레지스트리가 필요합니다.

4.2.3. 온-체인, 혼합 및 오프-체인 레지스트리 (On-chain, mixed and off-chain registries). 이러한 서비스 공급자의 레지스트리는 영구저장소에 레지스트리를 보관할 수 있는 스마트 컨트랙트의 도움으로 완전히 온-체인으로 구현될 수 있습니다. 그러나 이것은 매우 느리고 비용이 많이 듭니다. 상대적으로 작고 거의 변경되지 않은 온-체인 레지스트리는 오프-체인 (중앙형) 레지스트리 서비스를 제공하는 일부 노드를 지적하는데만 사용되는 (그들의 추상주소에 의해 또는 **3.2.12**에서 설명된 바와 같이 실제 추상주소를 찾는데 사용될 수 있는 퍼블릭 키에 의해) 혼합된 접근 방법이더 효율적입니다.

마지막으로, 분권화되고 순전히 오프-체인 접근 방식은 퍼블릭 오버레이 네트워크 (cf. **3.3**)로 구성될 수 있으며, 서비스를 제공하고자 하는 사람들 또는 누군가의 서비스를 구매하고자 하는 사람들은 프라이빗 키로 서명된 자신들의 오퍼를 단순히 브로드캐스트 합니다. 제공할 서비스가 매우 단순한 경우, 오퍼를 브로드캐스팅하는 것조차 필요하지 않을 수도 있습니다. 오버레이 네트워크의 대략적인 멤버십 자체가 특정 서비스를 제공하고자 하는 사람들의 "레지스트리"로 사용될 수 있습니다. 그런 다음 이 서비스를 요구하는 클라이언트는 이 오버레이 네트워크의 일부 노드를 찾은 다음 (**3.3.7** 참조) 이미 알려진 노드가 필요를 충족할 준비가 되지않은 경우 이웃 라우터를 쿼리 할 수 있습니다.

4.2.4. 레지스트리 또는 사이드-체인의 교환 (Registry or exchange in a side-chain). 분산형 혼합 레지스트리를 구현하기 위한 또 다른 접근법은 독립된 특수 블록체인("사이드-체인")을 만드는 것입니다. 이 사이드-체인은 온-체인 스마트 컨트랙트에 자신의 신분을 게시하고 이 특수 블록체인에 관심있는 모든 사람들에게 네트워크 접근을 제공하여 전용 오버레이 네트워크를 통해 트랜잭션 후보를 수집하고 블록 업데이트를 브로드캐스트 하는 자체 선언된 밸리데이터 집합에 의해 유지관리됩니다 (cf. **3.3**). 그런 다음 이 사이드-체인에 대한 모든 풀-노드는 공유 레지스트리 (본질적으로 사이드체인의 글로벌 상태와 동일함)의 자체 복사본을 유지관리할 수 있으며 이 레지스트리와 관련된 임의의 쿼리를 처리할 수 있습니다.

4.2.5. 워크체인의 레지스트리 또는 거래소 (Registry or exchange in a workchain). 또 다른 옵션은 레지스트리, 시장 및 거래소 생성을 전문으로하는 TON 내에 전용 워크체인을 만드는 것입니다. 이것은 기본 워크체인에 있는 스마트 컨트랙트를 사용하는 것보다 효율적이고 저렴할 수 있습니다 (cf. **2.1.11**). 그러나, 이것은 여전히 사이드--체인에서 레지스트리를 유지하는 것보다 비용이 많이 듭니다 (cf. **4.2.4**).

4.3 TON 서비스 액세스 (Accessing TON Services)

우리는 **4.1**에서 TON 생태계에 상주하는 새로운 서비스와 애플리케이션을 만드는데 사용할 수 있는 다양한 접근 방식에 대해 논의했습니다. 이제는 이러한 서비스에 액세스하는 방법과 *TON DNS* 및 TON 저장소를 포함하여 TON에서 제공하는 "도우미 서비스"에 대해 설명합니다.

4.3.1. TON DNS: 주로 온-체인의 계층적 도메인 이름 서비스 **(TON DNS: a mostly onchain hierarchical domain name service).** *TON DNS* 는 스마트 컨트랙트를 사용하여 사람이 읽을 수 있는 도메인 이름의 맵을 ADNL 네트워크 노드 및 TON 블록체인 계정 및 스마트 컨트랙트의 (256-비트) 주소로 유지하는 사전 정의된 서비스입니다.

누구나 원칙적으로 TON 블록체인을 사용하여 이러한 서비스를 구현할 수 있지만 애플리케이션이나 서비스가 사람이 읽을 수 있는 식별자를 주소로 변환하려고 할 때마다 기본적으로 잘 알려진 인터페이스가 있는 미리 정의된 서비스를 갖는 것이 유용합니다.

4.3.2. TON DNS 사용 사례 (TON DNS use cases). 예를 들어, 일부 암호화폐를 다른 사용자 또는 판매자에게 전송하려는 사용자는 그들의 256-비트 계정 식별자를 유지하고 라이트 월렛 클라이언드의 수신자 필드에 복사하여 붙여 넣는 대신 해당 사용자 또는 판매자의 계정의 TON DNS 도메인 이름을 기억하는 것을 선호할 수 있습니다.

마찬가지로, TON DNS는 스마트 컨트랙트의 계정 식별자 또는 톤 서비스 및 톤-사이트의 진입점 (cf. **4.1.5**)을 찾거나, 전문 클라이언트 ("톤-브라우저") 또는 독립실행형 애플리케이션과 결합된 일반 인터넷 브라우저를 사용하여 WWW 같은 브라우징 경험을 사용자에게 제공하는데 사용될 수 있습니다.

4.3.3. TON DNS 스마트 컨트랙트 (TON DNS smart contracts). TON DNS는 특별한 (DNS) 스마트 컨트랙트 트리를 통해 구현됩니다. 각 DNS 스마트 컨트랙트는 일부 고정도메인의 서브도메인을 등록하는 것을 책임집니다. TON DNS 시스템의 레벨-1 도메인(이 유지되는 "루트" DNS 스마트 컨트랙트는 마스터체인에 있습니다. 이 계정 ID는 TON DNS 데이터베이스에 직접 액세스하려는 모든 소프트웨어에 반드시 하드-코딩되어야 합니다.

DNS 스마트 컨트랙트에는 가변 길이의 널(null)로 끝나는 UTF-8 문자열을 "값"에 매핑하는 해시맵이 들어 있습니다. 이 해시맵은 **2.3.7**에서 설명한 것과 유사하지만 가변-길이 비트스트링을 키로 지원하는 바이너리 패트리샤 트리로 구현됩니다.

4.3.4. DNS 해시맵 또는 TON DNS 레코드의 값 (Values of the DNS hashmap, or TON DNS records). 값은 TL-체계에 의해 기술된 "TON DNS레코드"입니다 (cf. **2.2.5**). 이는 "매직 넘버"로 구성되어 지원되는 옵션 중 하나를 선택한, 다음 계정 식별자 혹은 스마트 컨트랙트 식별자, 추상 네트워크 주소(cf. **3.1**), 서비스의추상주소를 찾는데 사용되는 퍼블릭 (cf. **3.2.12**) 키 및 오버레이 네트워크에 대한설명, 등을 포함할 수 있습니다. 중요한 경우는 다른 DNS 스마트 컨트랙트의 경우입니다. 이러한 경우 스마트 컨트랙트는 도메인의 서브도메인을 해결하는데 사용됩니다. 이 방법으로, 도메인의 소유자가 통제하는 다른 도메인에 대한 별도의 레지스트리를 만들 수 있습니다.

이러한 레코드에는 만료시간, 캐싱시간 (블록체인에서 값을 업데이트하는 것이 너무 비싸기 때문에 일반적으로 매우 큽니다) 및 대부분의 경우 문제의 서브도메인 소유자에 대한 참조가 포함될 수 있습니다. 소유자는 이 레코드를 변경하고 (특히 소유자 필드이므로 도메인을 다른 사람의 통제로 넘겨줌) 연장할 수 있는 권리가 있습니다.

4.3.5. 기존 도메인의 새 서브도메인 등록. 기존 도메인의 새 서브도메인을 등록하려면 등록할 서브도메인 (즉, 키), 몇가지 미리 정의된 형식 중 하나의 값, 소유자의 신분, 만료날짜 및 도메인 소유주가 결정한 일부 암호화폐가 포함된 스마트 컨트랙트 (즉, 해당 도메인의 레지스트러)에 간단히 메세지를 보냅니다.

서브도메인은 "선착순” 방식으로 등록됩니다.

4.3.6. DNS 스마트 컨트랙트에서 데이터 검색 (Retrieving data from a DNS smartcontract). 원칙적으로 DNS 스마트 컨트랙트를 포함하는 마스터체인 또는 샤드체인의 풀-노드는 스마트 컨트랙트의 영구저장소 내부에 있는 해시맵의 구조와 위치를 알고 있는 경우 해당 스마트 컨트랙트의 데이터베이스에 있는 서브도메인을 조회할 수 있습니다.

그러나, 이 방법은 특정 DNS스마트 컨트랙트에만 적용됩니다. 비표준 DNS 스마트 컨트랙트가 사용되면 비참하게 실패할 것입니다.
그 대신, 일반적인 스마트 컨트랙트 인터페이스와 *get*방식 (cf. **4.3.11**)에 기반한 접근 방식이 사용됩니다. DNS 스마트 컨트랙트는 키를 찾기 위해 호출되는 "알려진 서명"으로 반드시 get방식을 정의해야 합니다. 이 접근법은 다른 스마트 컨트랙트, 특히 온-체인 및 혼합 서비스를 제공하는 컨트랙트에 대해서도 의미가 있으므
로 **4.3.11**에서 자세히 설명됩니다.

4.3.7. TON DNS 도메인 변환 (Translating a TON DNS domain). 자체 또는 라이트 클라이언트를 대신하여 작동하는 풀-노드가 DNS 스마트 컨트랙트 데이터베이스의 항목을 조회할 수 있으면 임의의 TON DNS 도메인 이름을 잘 알려진 고정된 루트 DNS 스마트 컨트랙트 (계정) 식별자에서부터 역으로 변환될 수 있습니다.

예를 들어, A. B. C 를 변환하려는 경우 루트 도메인 데이터베이스에서 .C, .B. C 와 A. B. C 키를 찾습니다. 첫 번째가 발견되지 않고 두 번째가 발견되면, 또한 그 값이 다른 DNS스마트 컨트랙트에 대한 참조인 경우 해당 스마트 컨트랙트의 데이터베이스에서 A 가 조회되고 최종값이 검색됩니다.

4.3.8. 라이트 노드를 위한 TON DNS 도메인 변환 (Translating TON DNS domains for light nodes). 이 방식으로 마스터체인과 도메인 조회 과정에 관련된 모든 샤드체인의 풀-노드는 외부 도움없이 모든 도메인 이름을 현재값으로 변환할 수 있습니다. 라이트 노드는 풀-노드를 대신하여 이것을 요청하고 머클증명과 함께 값을 반환할 수 있습니다 (cf. **2.5.11**). 이 머클증명은 라이트 노드가 응답이 올바른지 확인할 수 있게하여 일반적인 DNS 프로토콜과 달리 악의적인 인터셉터에 의해 TON DNS 응답을 "위장"할 수 없습니다.

모든 샤드체인에 대해 완전한 노드가 될 것으로 예상할 수 있는 노드가 없으므로 실제 TON DNS 도메인 변환에는 이 두 가지 전략을 조합해야 합니다.

4.3.9. 전용 "TON DNS 서버" (Dedicated “TON DNS servers”). 간단한 "TON DNS 서버"를 제공하여 주어진 도메인을 변환하도록 요청하는 RPC "DNS" 쿼리를 수신하고(cf. **3.1**에서 설명된 ADNL 또는 RLDP 프로토콜을 통해), 필요한 경우 일부 서브쿼리를 다른 노드로 전달하여 이 쿼리를 처리한 다음, 필요한 경우 머클증명으로 보강된 원래 쿼리에 대한 응답을 반환합니다.

이러한 "DNS 서버"는 cf. **4.2**에 설명된 방법 중 하나를 사용하여 다른 노드 및 특히 라이트 클라이언트에게 (무료 또는 유료로) 서비스를 제공할 수 있습니다. 이러한 서버는 만약 TON DNS 서비스의 일부로 간주될 경우 배포된 온-체인 서비스에서 분산형 혼합형 서비스 (즉, "안개서비스")로 효과적으로 변환됩니다.

이것으로 TON 블록체인과 TON 네트워크 엔티티의 사람이 읽을 수 있는 도메인 이름을 위한 확장 가능한 온-체인 레지스트리인 TON DNS 서비스에 대한 간략한 개요를 마칩니다.

4.3.10. 스마트 컨트랙트에 보관된 데이터 액세스 (Accessing data kept in smart contracts). 우리는 상태를 변경하지 않고 스마트 컨트랙트에 저장된 데이터에 액세스해야 하는 경우가 있음을 이미 확인했습니다.

스마트 컨트랙트 구현의 세부사항을 알고 있는 경우 스마트 컨트랙트가 있는 샤드체인의 모든 풀-노드에서 사용할 수 있는 스마트 컨트랙트의 영구저장소에서 필요한 모든 정보를 추출할 수 있습니다. 그러나, 이것은 스마트 컨트랙드 구현에 따라 일을 하는 방식이 상당히 비현실적입니다.

4.3.11. 스마트 컨트랙트의 "get 방식" (“Get methods” of smart contracts). 더 나은 방법은 스마트 컨트랙트에서 일부 *get* 방식을 정의하는 것입니다. 이것은 즉, 일부 유형의 인바운드 메시지이지만 전달될 때 스마트 컨트랙트의 상태에 영향을 주지 않고 get 방식의 "결과"를 포함하는 하나 이상의 출력 메시지를 생성합니다.

이러한 방식으로, 알려진 서명이 있는 get 방식을 구현한다는 것을 알면서 스마트 컨트랙트로부터 데이터를 획득할 수 있습니다 (즉, 발신될 인바운드 메시지 및 결과적으로 수신되는 아웃바운드 메시지의 알려진 형식).

이 방법은 훨씬 더 우아하고 객체 지향 프로그래밍 (OOP)과 일치합니다. 하지만, 이것은 현재까지 명백한 결함이 있습니다. 실제로 트랜잭션을 블록체인에 위임해야 하며 (스마트 계약서에 get 메시지 보내기), 밸리데이터가 커밋하여 처리할 때까지 기다린 후 새 블록에서 응답을 추출하고 가스 비용을 지불해야 합니다 (즉, 밸리데이터의 하드웨어에서 get 방식 실행하기 위해). 이것은 자원 낭비입니다: get 방식은 스마트 컨트랙트의 상태를 어쨌든 변경하지 않으므로 블록체인에서 실행될 필요가 없습니다.

4.3.12. 스마트 컨트랙트의 get 방식 임시실행 (Tentative execution of get methods of smart contracts). 우리는 스마트 컨트랙트의 주어진 상태에서 시작하여 해당 트랜잭션을 실제로 커밋하지 않고 모든 풀-노드가 스마트 컨트랙트의 모든 방식을 잠정적으로 실행할 수 있다고 (즉, 모든 메시지를 스마트 컨트랙트로 전달) 이미 언급했습니다 (cf. **2.4.6**). 풀-노드는 고려중인 스마트 컨트랙트의 코드를 TON VM에 로드하고 샤드체인의 글로벌 상태 (샤드체인의 모든 풀-노드에게 알려짐)에서 영구저장소를 초기화 한 다음 인바운드 메시지가 있는 스마트 컨트랙트 코드를 입력 매개변수로 실행합니다. 생성된 출력 메시지는 이 계산의 결과를 가져올 것입니다.

이러한 방식으로, 풀-노드는 서명 (즉, 인바운드 및 아웃바운드 메시지의 형식)이 알려져 있으면, 임의의 스마트 컨트랙트의 임의의 get 방식을 평가할 수 있습니다. 노드는 이 평가 동안 접근된 샤드체인 상태의 셀을 추적할 수 있고, 풀-노드에 그렇게 하도록 요구할 수 있는 라이트 노드의 이익을 위해 수행된 계산의 유효성에 대한 머클증명을 생성할 수 있습니다 (cf. **2.5.11**).

4.3.13. TL-계획의 스마트 컨트랙트 인터페이스 (Smart-contract interfaces in TL schemes). 스마트 컨트랙트에 의해 구현된 방식들 (즉, 그것에 의해 받아 들여지는 입력 메시지)은 TL-계획 (cf. **2.2.5**)에 의해 기술될 수있는 TL-직렬화된 객체이다. 결과 출력 메시지는 동일한 TL-계획으로도 설명될 수 있습니다. 이러한 방식으로, 스마트 컨트랙트에 의해 다른 계정 및 스마트 컨트랙트로 제공되는 인터페이스는 TL-계획을 통해 공식화될 수 있습니다.

특히, 스마트 컨트랙트에 의해 지원되는 get 방식(의 서브세트)은 공식화된 스마트 컨트랙트 인터페이스로 설명될 수 있습니다.

4.3.14. 스마트 컨트랙트의 퍼블릭 인터페이스 (Public interfaces of a smart contract). 형식화된 스마트 컨트랙트 인터페이스는 TL-계획(TL 소스 파일로 표시; cf. **2.2.5**) 또는 직렬화된 형식[39] 으로 게시할 수 있습니다 (예. 블록체인에 저장된 스마트 컨트랙트 계정 설명의 특수 필드 또는 이 인터페이스가 여러번 참조될 경우 별도로). 후자의 경우 지원되는 퍼블릭 인터페이스의 해시가 인터페이스 설명 자체 대신 스마트 컨트랙트 설명에 통합될 수 있습니다.

이러한 퍼블릭 인터페이스의 예는 DNS 스마트 컨트랙트의 예이며 이는 서브도메인을 조회하는데 최소한 하나의 표준 get 방식을 구현해야 합니다 (cf. **4.3.6**). 새 서브도메인을 등록하기 위한 표준 방법은 DNS 스마트 컨트랙트의 퍼블릭 공개 인터페이스에도 포함될 수 있습니다.

[39] TL-체계는 자체적으로 TL-직렬화 될 수 있습니다; https://core.telegram.org/mtproto/TL-tl

4.3.15. 스마트 컨트랙트의 사용자 인터페이스 (User interface of a smart contract). 스마트 컨트랙트를위한 퍼블릭 인터페이스의 존재는 다른 이점 또한 가지고 있습니다. 예를 들어, 월렛 클라이언트 애플리케이션은 사용자의 요청에 대한 스마트 컨트랙트를 검토하면서 그러한 인터페이스를 다운로드할 수 있고, 공식 계약에서 제공되는 경우 사람이 읽을 수 있는 코멘트를 사용하여 스마트 컨트랙트에서 지원하는 퍼블릭 메소드의 목록 (즉, 이용 가능한 액션)을 보여줄 수도 있습니다. 사용자가 이 방법들 중 하나를 선택하면, TL-계획에 따라 자동으로 양식이 생성될 수 있으며 여기서 사용자는 선택된 방법에 필요한 모든 필드와 원하는 양의 암호화폐가 이 요청에 첨부될 것을 요구 받게됩니다. 이 양식을 제출하면 사용자의 블록체인 계정에서 전송되어 방금 작성한 메시지가 포함된 새로운 블록체인 트랜잭션이 생성됩니다.

이 방법으로 사용자는 스마트 컨트랙트가 인터페이스를 게시한 경우, 특정 양식을 작성하고 제출하여 사용자 친화적인 방식으로 월렛 클라이언트 애플리케이션의 임의의 스마트 컨트랙트와 상호작용할 수 있습니다.

4.3.16. "톤-서비스"의 사용자 인터페이스 (User interface of a “ton-service”). "톤-서비스” (즉, **TON** 네트워크에 상주하며 cf. **3** (TON 네트워킹)의 ADNL 및 RLDP 프로토콜을 통해 쿼리를 수락하는 서비스 cf. **4.1.5**)는 TL-계획으로 설명된 퍼블릭 인터페이스를 갖는 것이 이익이 될 수 있습니다 (cf. **2.2.5**). 라이트 월렛 또는 "톤-브라우저"와 같은 클라이언트 애플리케이션은 **4.3.15** 에서 설명한 것처럼 사용자가 방법 중 하나를 선택하고 인터페이스에서 정의된 매개변수로 양식을 채우도록 요청할 수 있습니다. 유일한 차이점은 TL-직렬화된 메시지가 블록체인에서 트랜잭션으로 제출되지 않는다는 것입니다. 대신에, 그것은 RPC 쿼리로서 문제의 "톤-서비스"의 추상주소로 보내지고, 이 쿼리에 대한 응답은 형식 인터페이스 (즉, TL-계획)에 따라 분석되고 표시됩니다.

4.3.17. TON DNS 통한 사용자 인터페이스 위치 (Locating user interfaces via TON DNS). 톤-서비스 또는 스마트 컨트랙트 계정 식별자의 추상주소가 포함된 TON DNS 레코드에는 해당 엔티티의 퍼블릭 (사용자) 인터페이스 또는 지원되는 여러 인터페이스를 설명하는 선택적 필드가 포함될 수도 있습니다. 그런 다음 클라이언트 애플리케이션 (월렛, 톤-브라우저 또는 톤-프록시가 될 수 있음)은 인터페이스를 다운로드하고 문제의 엔티티 (스마트 컨트랙트 또는 톤-서비스)와 일관된 방식으로 상호 작용할 수 있습니다 .

4.3.18. 온-체인과 오프-체인 서비스의 구분 흐리기 (Blurring the distinction between on-chain and off-chain services). 이러한 방식으로 최종 사용자에게 온-체인, 오프-체인 및 혼합 서비스 (cf. **4.1.2**) 간의 구분이 희미해집니다; 원하는 서비스의 도메인 이름을 자신의 톤-브라우저 또는 월렛의 주소 행에 입력만 하면 나머지는 클라이언트 애플리케이션에 의해 원활하게 처리됩니다.

4.3.19. 텔레그램 메신저 클라이언트에 라이트 월렛과 TON 엔티티 탐색기 내장 (A light wallet and TON entity explorer can be built into Telegram Messenger clients). 흥미로운 기회가 이 시점에서 발생합니다. 위의 기능을 구현하는 라이트 월렛과 TON 엔티티 탐색기를 텔레그램 메신저 스마트폰 클라이언트 애플리케이션에 내장하여 2억명이 넘는 사람들에게 기술을 제공할 수 있습니다. 사용자는 메시지에 TON URIs (cf. **4.3.22**)를 포함시켜 TON 엔터티 및 리소스에 하이퍼링크들을 보낼 수 있습니다; 그러한 하이퍼링크가 선택되면, 수신자의 텔레그램 클라이언트 애플리케이션에 의해 내부적으로 열리며, 선택한 개체와의 상호작용이 시작됩니다.

4.3.20. HTTP 인터페이스를 지원하는 톤-서비스로서의 "톤-사이트" (“ton-sites” as tonservices supporting an HTTP interface). 톤-사이트는 다른 인터페이스와 함께 단순히 HTTP 인터페이스를 지원하는 톤-서비스입니다. 이 지원은 해당 TON DNS 레코드에서 발표될 수 있습니다.

4.3.21. 하이퍼링크 (Hyperlinks). 톤-사이트에서 반환된 HTML 페이지에는 추상 네트워크 주소, 계정 식별자 또는 사람이 읽을 수있는 TON DNS 도메인을 포함한 톤-하이퍼링크가 포함될 수 있습니다 (즉, 특별히 제작된 URI 체계 (cf. **4.3.22**)를 통해 다른 톤-사이트, 스마트 컨트랙트 및 계정에 대한). 그런 다음 "톤-브라우저"는 **4.3.15** 및 **4.3.16**에 설명된대로 그러한 하이퍼링크를 사용자가 선택하고 사용할 인터페이스를 감지하고, 사용자 인터페이스 양식을 표시할 수 있습니다.

4.3.22. 하이퍼링크 URL은 일부 매개 변수를 지정할 수 있습니다 (Hyperlink URLs may specify some parameters). 하이퍼링크 URL에는 문제의 서비스의 (TON) DNS 도메인 또는 추상주소 뿐만 아니라 호출할 메서드의 이름과 일부 또는 모든 매개변수가 포함될 수 있습니다. 이것에 대한 가능한 URI계획은 다음과 같습니다.

ton://<domain>/<method>?<field1>=<value1>&<field2>=. . .

사용자가 톤-브라우저에서 이러한 링크를 선택하면, 작업이 즉시 수행되거나 (특히 스마트 컨트랙트의 get 방식이거나 익명으로 호출되는 경우) 또는 부분적으로 채워진 양식이 표시되어 사용자가 명시적으로 확인하고 제출해야 합니다 (결제양식의 경우 필요할 수 있음).

4.3.23. POST 작업 (POST actions). 톤-사이트는 HTML 페이지에 내장될 수 있으며 보통의 POST 양식을 반환하고 POST 작업은 적절한 (TON) URL을 통해 톤-사이트, 톤-서비스 또는 스마트 컨트랙트를 참조합니다. 이 경우 사용자가 해당 사용자정의 양식을 작성하고 제출하면 즉시 또는 명시적 확인 후 해당 조치가 취해집니다.

4.3.24. TON WWW. 위의 모든 사항은 톤-브라우저를 통해 최종 사용자가 액세스 할 수 있는 TON 네트워크에 상주하는 상호참조 엔티티 전체 웹을 생성하여 사용자에게 WWW 같은 탐색 경험을 제공합니다. 최종 사용자의 경우, 이것은 마침내 블록체인 애플리케이션을 익숙한 웹사이트와 근본적으로 유사하게 만듭니다.

4.3.25. TON WWW의 장점 (Advantages of TON WWW). 온-체인 및 오프-체인 서비스의 이 "TON WWW"는 기존의 것보다 몇 가지 장점이 있습니다. 예를 들어,

- 지불은 본질적으로 시스템에 통합되어 있습니다.
- 사용자 신분은 항상 서비스에 제공될 수 있거나 (생성된 트랜잭션 및 RPC 요청에 대해 자동으로 생성된 서명을 통해) 또는 숨길 수 있습니다.
- 서비스는 사용자 자격 증명을 확인하고 다시 확인할 필요가 없습니다; 이러한 자격 증명은 블록체인에 한 번에 최종적으로 게시될 수 있습니다.
- 사용자 네트워크 익명성은 TON- 프록시를 통해 쉽게 보존될 수 있으며 모든 서비스는 효과적으로 차단 해제됩니다.
- 톤-브라우저를 TON 페이먼트시스템과 통합할 수 있기 때문에 소액결제가 가능하고 쉽습니다.

5 TON결제 (TON Payments)

이 텍스트에서 간략하게 논의할 TON 프로젝트의 마지막 구성요소는 (소액) 결제채널 및 "라이트닝 네트워크" 가치전송을 위한 플랫폼인 TON 페이먼트입니다. 모든 트랜잭션을 블록체인에 투입(커밋)할 필요없이 관련 트랜잭션 수수료를 지불하고 문제의 트랜잭션을 포함하는 블록이 확인될 때까지 5초 동안 기다릴 필요없이 "즉시" 결제를 가능하게 합니다.

이러한 즉시 결제의 전반적인 간접비는 아주 작기 때문에 소액결제에 사용할 수 있습니다. 예를 들어, TON 파일저장서비스는 128KiB의 다운로드 데이터마다 요금을 부과하거나, 또는 유료 TON 프록시는 중계된 128KiB 트래픽마다 약간의 소액결제를 요구할 수 있습니다.

TON 페이먼트는 TON 프로젝트의 핵심 구성요소보다 늦게 발표될 것이지만 초기에 몇 가지 고려 사항을 작성해야 합니다. 예를 들어, TON 블록체인 스마트 컨트랙트의 코드를 실행하는데 사용되는 TON 가상머신 (TON VM, cf. **2.1.20**)은 머클증명과 함께 일부 특수작업을 지원해야 합니다. 그러한 지원이 원래 설계에 없다면 나중에 추가하는 것이 문제가 될 수 있습니다 (cf. **2.8.16**). 그러나 우리는 TON VM이 "스마트" 결제채널 (cf. **5.1.9**)을 자연스럽게 지원할 것으로 봅니다.

5.1 결제채널 (Payment Channels)

우선 포인트-투-포인트 결제채널에 대한 논의부터 시작하여 TON 블록체인에서 구현할 수 있는 방법에 대해 설명합니다.

5.1.1. 결제채널의 아이디어 (The idea of a payment channel). 두 당사자인 A 와 B 가 앞으로 서로에게 많은 돈을 지불해야 한다는 것을 알고 있다고 가정해보십시오. 블록체인에서 각 지불을 트랜잭션으로 커밋하는 대신 공유된 "자금 풀" (또는 정확히 두 개의 계정이 있는 작은 개인은행)을 만들고 일부 금액을 기부합니다. A 는 a 코인을 제공하고 B 는 b 코인을 제공합니다. 이것은 블록체인에 특별한 스마트 컨트랙트를 만들어 돈을 보내면 성취됩니다.

"자금 풀"을 만들기 전에 양측은 특정 프로토콜에 동의합니다. 그들은 풀의 상태, 즉 공유 풀의 잔금을 추적합니다. 원래 상태는 (a, b) 이며 이는 a 코인이 실제로 A 에 속하고 b 코인이 B 에 속한 것을 의미합니다. 그러면 A 가 B 에게 d 코인을 지불하기를 원하면 새로운 상태가 $(a', b') = (a-d\ ,\ b+d)$ 라고 간단히 동의할 수 있습니다. 그 후에 B 가 A 에게 d' 코인을 지불하기를 원하면 상태는 $(a'', b'') = (a'+d', b'-d')$ 등이 될 것입니다.

풀 (자금-풀) 내부의 이러한 잔액의 업데이트는 완전히 오프-체인으로 수행됩니다. 두 당사자가 풀에서 기금을 인출하기로 결정하면 풀의 최종 상태에 따라 수행합니다. 이것은 A 와 B 의 서명과 함께 합의된 최종 상태 (a^*, b^*) 를 포함하는 스마트 컨트랙트에게 특별한 메시지를 보내면 이루어집니다. 그런 다음 스마트 컨트랙트는 a^* 코인을A , b^* 코인을 B 에게 전송하고 스스로 파괴합니다.

이 스마트 컨트랙트는 A 와 B 가 풀의 상태를 업데이트하는데 사용하는 네트워크 프로토콜과 함께 A 와 B 간의 간단한 결제채널입니다. **4.1.2** 에 설명된 분류에 따르면 이는 혼합 서비스입니다. 그 상태의 일부분은 블록체인 (스마트 컨트랙트)에 있지만 대부분의 상태 업데이트는 (네트워크 프로토콜에 의해) 오프-체인으로 수행됩니다. 모든 것이 잘 수행되면 양 당사자는 서로에게 원하는대로 많은 결제를 할 수 있게 되고 (유일한 제한은 채널의 "수용력"을 초과하지 않는 것입니다 - 즉, 결제채널의 잔고가 모두 비음수입니다), 블록체인에 두 개의 트랜잭션만 커밋합니다: 하나는 결제채널 (스마트 컨트랙트)을 여는 (생성) 것이고 다른 하나는그것을 닫는 (파괴) 것입니다.

5.1.2. 신용없는 결제채널 (Trustless payment channels). 이전 예제는 다소 비현실적이었습니다. 왜냐하면 양 당사자가 기꺼이 협조하고 어떤 이점을 얻기 위해 절대로 속이지 않을 것이라고 가정했기 때문입니다. 예를 들어, A 가 최종 잔액 (a', b') 을 $a' < a$ 로 서명하지 않기로 결정했다고 상상해 보십시오. 이것은 B 를 어려운 상황에 넣을 것입니다.

이러한 시나리오를 방지하기 위해, 일반적으로 당사자가 서로 신뢰하도록 요구하지 않는 무신용 결제채널 프로토콜을 개발하려고 시도하고 속임수를 쓰려는 당사자를 처벌하는 조항을 만들어야 합니다.

이것은 대개 서명을 통해 이루어집니다. 결제채널 스마트 컨트랙트는 A 와 B 의 퍼블릭 키를 알고 필요에 따라 서명을 확인할 수 있습니다. 결제채널 프로토콜은 당사자가 중간 상태에 서명하고 서로 서명을 전송하도록 요구합니다. 그런 다음, 당사자 중 한 사람이 속임수를 쓴 경우 - 예를 들어, 결제채널의 일부 상태가 존재하지 않은 것처럼 가장하는 경우 - 그 행동은 해당 상태에 대한 서명을 보여줌으로써 입증될 수 있습니다. 결제채널 스마트 컨트랙트는 양 당사자의 불만 사항을 서로 처리할 수 있고 모든 돈을 압수하여 상대방에게 수여함으로써 유죄 당사자를 처벌할 수있는 "온-체인 중재자(on-chain arbiter)"의 역할을 합니다.

5.1.3. 간단한 양방향 동기식 무신용 결제채널 (Simple bidirectional synchronous trustless payment channel). 다음과 같은 좀 더 현실적인 예를 생각해 봅시다: 결제채널의 상태를 트리플 (δ_i, i, o_i) 로 가정하십시오. 여기서 i 는 상태의 배열번호입니다 (원래 0 이고 그 다음에 후속 상태가 나타나면 1씩 증가합니다). δ_i 는 채널불균형 (A 와 B 는 각각 $a+\delta_i$ 와 $b-\delta_i$ 코인을 소유함을 의미), 그리고 o_i는 다음 상태 (A또는 B)를 생성할 수 있는 당사자를 나타냅니다. 각 상태는 앞으로 더 진행이 있기 전에 A 와 B 모두의 서명을 받아야 합니다.

이제 A 가 결제채널을 통해 d 코인을 B 로 전송하기를 원하고 현재 상태가 $o_i = A$ 로 $\delta_i = (\delta_i, i, o_i)$ 이면, 이는 간단히 새로운 상태 $S_{i+1}=(\delta_{i-d}, i+1, o_{i+1})$ 를 생성하고, 서명한 후 서명과 함께 B 로 보냅니다. 그런 다음 B 는 서명하고 A 에 서명 사본을 보냄으로써 서명을 확인합니다. 그 후 양 당사자는 양쪽의 서명이 있는 새로운 상태의 사본을 갖고 새로운 전송이 발생할 수 있습니다.

만약 A 가 B 에게 $o_i = B$ 로 상태 δ_i 인 코인을 전송하기를 원한다면, 먼저 B 에게 같은 불균형 $S_{i+1} = \delta_i$ 를 가졌지만 $o_i + 1 = A$ 인 후속 상태S_{i+1}에게 커밋하도록 요구합니다. 그 후에, A 는 전송을 할 수 있습니다.

두 당사자가 결제채널을 닫는데 동의하면 둘 다 자신의 특별한 최종 서명을 최종 상태라고 생각하는 상태 S_k 에 두고 두 당사자의 최종 서명과 함께 최종 상태를 전송함으로써 지불채널 스마트 컨트랙트의 깨끗한 또는 양면확정방법을 사용합니다.

상대방이 최종 서명을 제공하지 않거나 단순히 응답하지 않는 경우 채널을 일방적으로 닫을 수 있습니다. 이를 위해 당사자는 일방적인 최종방법을 사용하여 최종 컨트랙트 상태, 최종 서명 및 상대방의 서명이 있는 최신 상태를 스마트 컨트랙트에 보냅니다. 그 후에 스마트 컨트랙트는 받은 최종 상태에 즉각적으로 행동하지 않습니다. 그 대신, 상대방이 최종 상태의 버전을 제시하기 위해 특정 기간 (예를 들어, 1일)을 기다립니다. 상대방이 그의 버전을 제출하고 이미 제출된 버전과 호환되는 것으로 밝혀지면 "참" 최종 상태가 스마트 컨트랙트에 의해 계산되어 그에 따라 돈을 분배하는데 사용됩니다. 상대방이 최종 상태의 버전을 스마트 컨트랙트에 제시하지 못하면 제시된 최종 상태의 유일한 사본에 따라 돈이 재분배됩니다.

두 당사자 중 한 사람이 속임수를 쓰는 경우 - 예를 들어, 두 가지 다른 상태를 최종으로 서명하거나, 두 가지 다른 후속 상태 S_{i+1} 와 S'_{i+1}, 또는 무효한 새로운 상태 S_{i+1} (예. 불균형 $S_{i+1} < -a$ 또는 $> b$ 와) 를 서명하면 - 상대방은 이 잘못된 행동의 증거를 스마트 컨트랙트의 세 번째 방법에 제출할 수 있습니다. 유죄 판결을 받은 당사자는 즉시 결제채널에서 지분을 잃어 처벌됩니다.

이 단순 결제채널 프로토콜은 상대방의 협조 여부에 관계없이 언제든지 정당한대가를 받을 수 있고, 속임수를 쓰려고하면 결제채널에 투입된 모든 자금을 잃을 가능성이 있다는 의미에서 공정 합니다.

5.1.4. 두 밸리데이터가 있는 간단한 가상 블록체인으로서의 동기식 지불채널 (Synchronous payment channel as a simple virtual blockchain with two validators). 간단한 동기식 결제채널의 위 예는 다음과 같이 재구성 할 수 있습니다. 상태의 배열 $S_0, S_1, \ldots, S_n$ 이 실제로 매우 간단한 블록체인의 블록 배열이라고 상상해보십시오. 이 블록체인의 각 블록은 본질적으로 오직 블록체인의 현재 상태와 혹은 이전 블록 (즉, 해시)에 대한 참조만 포함합니다. 두 당사자 A 와 B 는 이 블록체인의 밸리데이터 역할을 하므로 모든 블록은 두 가지 서명을 모두 수집해야 합니다. 블록체인의 상태 S_i 는 다음 블록에 대해 지정된 생성자 oi 를 정의하므로 다음 블록을 생성하기 위해 A 와 B 사이에 경쟁이 없습니다. 생산자 A 는 A 에서 B 로 자금을 이체하는 (즉, 불균형을 줄입니다: $\delta_{i+1} \leq \delta_i$) 블록을 생성할 수 있으며, B 는 B 에서 A 로만 자금을 이체(즉, δ 를 증가)할 수 있습니다.

두 명의 밸리데이터가 블록체인의 최종 블록 (및 최종 상태)에 동의하면 두 당사자의 특별 "최종" 서명을 수집하고 최종 블록과 함께 결제채널 스마트 컨트랙트에 제출하여 처리하고 그에 따라 돈을 다시 배분함으로써 확정됩니다.

밸리데이터가 유효하지 않은 블록에 서명하거나, 포크를 만들거나, 또는 두 개의 다른 최종 블록에 서명하는 경우 두명의 밸리데이터의 "온-체인 중재자" 역할을 하는 스마트 컨트랙트가 잘못된 행동의 증거를 제시하여 처벌 받을 수 있습니다. 그런 다음 문제가 되는 당사자는 결제채널에 보관된 모든 돈을 잃게됩니다. 이는 지분을 잃는 밸리데이터와 유사합니다.

5.1.5. 두 개의 워크체인을 가진 가상 블록체인으로서 비동기 결제채널 (Asynchronous payment channel as a virtual blockchain with two workchains). 5.1.3 에서 논의된 동기 결제채널에는 특정 단점이 있습니다: 이전 트랜잭션이 상대방에 의해 검증되기 전에 다음 트랜잭션 (결제채널 내부의 송금)를 시작할 수 없습니다. 이것은 **5.1.4** 에서 논의된 싱글 가상블록체인을 두 개의 상호작용 가상 워크체인 (오히려 샤드체인)으로 대체함으로써 해결될 수 있습니다.

이 워크체인 중 첫 번째 트랜잭션은 A 에 의한 트랜잭션만 포함하고 해당 블록은 A 에 의해서만 생성될 수 있습니다: 이의 상태는 $S_i=(i, \phi_i, j, \psi_j)$ 입니다. 여기서 i 는 블록 배열번호 (즉, 트랜잭션의 수 또는 A 가 송금한 돈)이고, ϕ_i는 A 에서 B 로여태까지 송금한 총금액입니다.

j 는 A 가 알고 있는 B 의 블록체인에서 가장 최근에 유효한 블록의 배열 번호이며, ψ_j 는 j 트랜잭션에서 B 에서 A 로 전송된 금액입니다. j-번째 블록에 놓인 B 의 서명도 이 상태의 일부여야 합니다. 이 워크체인의 이전 블록과 다른 워크체인의 j-번째 블록의 해시도 포함될 수 있습니다. S_i 에 대한 유효조건은 $i > 0, \psi_j \geq 0$ 및 $-a \leq \psi_j - \phi_i$ 일 때 $\phi_i \geq 0,\ \phi_i \geq \phi_{i-1}$ 을 포함합니다.

마찬가지로, 두 번째 워크체인에는 B 에 의한 트랜잭션만 포함되며 해당 블록은 B 에 의해서만 생성될 수 있습니다: 이의 상태는 유사한 유효조건을 가진 $T_j = (j, \psi_j, i, \phi_i)$

이제 A 가 B 에게 돈을 이체하기를 원한다면 이제는 워크체인에 새로운 블록을 만들고 서명한 다음 승인을 기다리지 않고 B 로 보냅니다.

결제채널은 A 가 (이의 해당 버전의) 블록체인의 최종 상태에 서명하고 (특별한 " 최종 서명"과 함께), B 가 블록체인의 최종 상태에 서명하여 이 두 가지 최종 상태를 결제채널 스마트 컨트랙트의 깨끗한 확정방법을 제공합니다. 일방적인 마무리도 가능하지만, 이 경우 스마트 컨트랙트는 상대방이 적어도 일부 유예기간 동안 최종 상태의 버전을 제공할 때까지 기다려야합니다.

5.1.6. 단방향 결제채널 (Unidirectional payment channels). A 만이 B 에게 지불해야 하는 경우 (예. B 가 서비스 제공자이고 A 가 클라이언트인 경우), 일방적인 결 제채널을 생성할 수 있습니다. 본질적으로, 두 번째 워크체인이 없는 **5.1.5** 에서 설명한 첫 번째 워크체인일 뿐입니다. 반대로, **5.1.5** 에 설명된 비동기 결제채널은 동일한 스마트 컨트랙트에 의해 관리되는 두 개의 단방향 결제채널 또는 "하프-채널"로 구성되어 있다고 말할 수 있습니다.

5.1.7. 보다 정교한 결제채널. 약속 (More sophisticated payment channels. Promises). **5.2.4** 의 후반부에서 여러가지 결제채널의 체인을 통해 실시간 자금이체를 가능하게 하는 "라이트닝 네트워크" (cf. **5.2**)가 관련된 결제채널에서 높은 수준의 정교함이 요구됩니다.

특히, 우리는 "약속" 또는 "조건부 자금 송금"에 전념할 수 있기를 원합니다: A 는 B 에게 c 코인을 보내기로 동의하지만, B 는 특정 조건이 충족되는 경우에만 B코인을 받습니다 (예. B 가 v 의 알려진 값에 대해 $\mathrm{HASH}(u) = v$ 를 사용하여 문자열 u 를 제시할 경우). 그렇지 않으면 A 는 일정 기간 후에 돈을 돌려받을 수 있습니다.

이러한 약속은 단순한 스마트 컨트랙트으로 온-체인상에서 쉽게 구현될 수 있습니다. 그러나, 우리는 결제채널에서 약속 및 기타 종류의 조건부 자금 송금을 오프-체인으로 가능하게 하고 싶습니다. 왜냐하면 이것들은 "라이트닝 네트워크"에 존재하는 결제채널의 체인을 따라 송금하는 것을 상당히 단순화하기 때문입니다. (cf. **5.2.4**)

5.1.4 및 **5.1.5**에 설명된 "간단한 블록체인으로서의 결제채널" 그림은 여기에서 편리하게 됩니다. 이제 우리는 못이룬 “약속" 세트의 상태와 그러한 약속에 잠겨져 있는 금액이 포함된 보다 복잡한 가상 블록체인을 고려해 봅니다. 이 블록체인, 또는 비동기 두 워크체인은, 해시를 통해 이전 블록을 명시적으로 참조해야 합니다. 그럼에도 불구하고 일반적인 메커니즘은 동일하게 유지됩니다.

5.1.8. 정교한 결제채널 스마트 컨트랙트의 도전과제 (Challenges for the sophisticated payment channel smart contracts). 주목할 것은, 정교한 결제채널의 최종 상태는 여전히 작으며 "깨끗한" 최종 결정은 간단하지만 (만약 양측이 금액 지불에 동의하고 양측이 합의에 서명한 경우 나머지는 수행할 필요가 없음), 일방적인 최종 결정 방법과 사기 행위를 처벌하는 방법은 더 까다로울 필요가 있습니다. 실제로, 그들은 잘못된 행동의 머클증명을 수락하고 결제채널 블록체인의 보다 정교한 트랜잭션이 올바르게 처리되었는지 여부를 확인해야 합니다.

다시 말해, 결제채널 스마트 컨트랙트는 머클증명을 사용하여 "해시 유효성"을 검사할 수 있어야 하며 결제채널 (가상) 블록체인에 대한 *ev_trans* 및 *ev_block* 함수의 구현 (cf. **2.2.6**)을 포함해야 합니다.

5.1.9. "스마트" 결제채널을 지원을 위한 TON VM (TON VM support for “smart” payment channels). TON 블록체인 스마트 컨트랙트의 코드를 실행하는 데 사용되는 TON VM은 "스마트"한 또는 정교한 결제채널 (cf. **5.1.8**)에 필요한 스마트 컨트랙트를 실행하는 것에 어려움을 겪고 있습니다.

이 시점에서 "모든 것은 셀백이다" 패러다임 (cf. **2.5.14**)은 매우 편리해집니다.모든 블록 (임시 결제채널 블록체인의 블록 포함)은 셀백으로 표현되고 (대수 데이터 유형으로 설명됨) 이는 메시지 및 머클증명에도 동일하게 적용되므로 머클증명은 결제채널 스마트 컨트랙트에 전송된 인바운드 메시지에 쉽게 임베드 될 수 있습니다. 머클증명의 "해시 조건"은 자동으로 확인되며, 스마트 컨트랙트가 제시된 "머클증명"에 액세스하면, (불완전 하지만) 트리의 일부 서브트리가 생략된 서브트리의 머클 해시를 포함하는 특수 노드로 대체된 채로 해당 대수데이터 유형의 값인 것처럼 작동합니다.

스마트 컨트랙트는 예를 들어 결제채널 (가상) 블록체인의 블록과 그상태를 그 나타내는 해당 값으로 작동하며 이 블록과 이전 상태에서 그 블록체인의 *ev_block* 함수 (cf. **2.2.6**)를 평가한 가치와 함께 작동합니다. 그런 다음 계산이 끝나고 블록에서 주장된 것과 최종 상태를 비교하거나 머클증명이 유효하지 않음을 나타내는 부재가 있는 서브트리에 액세스하려고 시도하는 동안 "부재 노드" 예외가 발생합니다.

이러한 방식으로 스마트 결제채널 블록체인에 대한 인증 코드의 구현은 TON 블록체인 스마트 컨트랙트를 사용하여 매우 간단합니다. 어떤 사람은 TON 가상머신에는 다른 간단한 블록체인의 유효성을 검사할 수 있는 지원 기능이 내장되어 있다고 말할 수 있습니다. 유일한 제한 요소는 스마트 컨트랙트에 (즉, 거래) 대한 인바운드 메시지에 통합될 머클증명의 크기입니다.

5.1.10. 스마트 결제채널 내 간단한 결제채널 (Simple payment channel within a smart payment channel). 기존 결제채널에 간단한 (동기식 또는 비동기식) 결제채널을 만들 가능성에 대해 논의하고자 합니다.

이것은 다소 복잡하게 보일 수 있지만, **5.1.7**에서 논의된 "약속"보다 이해하고 구현하는 것이 그리 어렵지 않습니다. 본질적으로 일부 해시 문제에 대한 해결책이 제시되면 상대방에게 코인c 를 지불하겠다는 약속 대신 A 는 다른 (가상) 결제채널 블록체인의 최종 합의에 따라 코인c 를 B 에게 지불할 것을 약속합니다. 일반적으로, 이 다른 결제채널 블록체인은 A 와 B 사이에 있을 필요가 없습니다. 예를 들어 코인c 및 코인 d 를 기꺼이 간단한 결제채널에 투입하려는 C 및 D 와 같은 다른 당사자가 참여할 수도 있습니다. (이 가능성은 나중에 **5.2.5** 에서 활용됩니다.)

만약 포괄적인 결제채널이 비대칭이라면, 두 가지 약속을 두 개의 워크체인에 커밋해야 합니다: A 는 "내부" 단순 결제채널의 최종 결제가 $0 \leq -\delta \leq c$; B 는 δ 가 양수이면 A 를 δ 로 지불할 것을 약속해야 합니다. 반면, 포괄 결제채널이 대칭이면, 매개 변수 (c, d) 가 있는 싱글 "간단한 결제채널 생성" 트랜잭션을 A 에 의한싱글 결제채널 블록체인에 위탁하여 수행할 수 있습니다. (A에 속하는 코인 c를 동결시킵니다).

그리고 나서 B 에 의해 특별한 "승인 트랜잭션"을 커밋합니다 (B 의 코인d 를 동결시킵니다). 제출하려는 머클증명의 크기를 최소화하기 위해 내부 결제채널이 매우 간단할 것으로 기대합니다 (예. **5.1.3** 에서 설명한 간단한 동기 결제채널). 외부 결제채널은 **5.1.7** 에서 설명한 의미에서 "스마트" 해야합니다.

5.2 결제채널 네트워크 또는 "라이트닝 네트워크" (Payment Channel Network, or “Lightning Network”)

이제 우리는 참여한 두 노드간에 즉시 돈을 송금 할 수 있는 TON 페이먼트의 "라이트닝 네트워크"에 대해 토론할 준비가 되었습니다.

5.2.1. 결제채널의 한계 (Limitations of payment channels). 결제채널은 그들 사이에 많은 돈이 송금될 것으로 예상되는 사람들에게 유용합니다. 그러나, 특정 수령인에게 한두 번만 돈을 송금해야 하는 경우 결제채널을 만드는 것은 실용적이지 않습니다. 무엇보다도, 그것은 결제채널에서 상당량의 돈이 동결되고 어쨌든 적어도 두 개의 블록체인 트랜잭션이 필요합니다.

5.2.2. 결제채널 네트워크 또는 "라이트닝 네트워크" (Payment channel networks, or “lightning networks”). 결제채널 네트워크는 결제채널 체인을 따라 송금을 가능하게 함으로써 결제채널의 한계를 극복합니다. A 가 E 에게 돈을 송금하기를 원한다면, E 와의 결제채널을 개설할 필요가 없습니다. 예를 들어 A 에서 B , B 에서 C , C 에서 D , D 에서 E 까지 네 개의 결제채널을 통해 A 와 E 를 연결하는 일련의 결제채널을 보유하는 것만으로 충분합니다.

5.2.3. 결제채널 네트워크 개요 (Overview of payment channel networks). "라이트닝 네트워크"라고도 알려진 결제채널 네트워크는 참여노드 모음으로 구성되며, 그 중 몇몇은 그들 사이에 장수하는 결제채널을 설정했습니다. 우리는 이러한 결제 채널이 **5.1.7**의 의미에서 "스마트"해야 한다는 것을 잠시 볼 것입니다. 참여노드 A가 다른 참여노드 E 로 돈을 전송하고자 할 때, 그는 결제채널 네트워크 내에서 A에서 E 를 연결하는 경로를 찾으려고 합니다. 그러한 경로가 발견되면, (A는) 이 경로를 따라 "체인송금"을 수행합니다.

5.2.4. 체인송금 (Chain money transfers). *A* 에서 *B* 로, *B* 에서 *C* 로, *C* 에서 *D* 로, *D* 에서 E 로 일련의 결제채널이 있다고 가정 해 봅시다. 이어서, *A* 가 *x* 개의 코인을 *E* 로 이전하려고 한다고 가정하십시오.

단순한 접근법은 *x* 개의 코인을 기존 결제채널을 따라 *B* 로 이관하고 *C* 에게 돈을 더 전달하도록 요청하는 것입니다. 그러나, 이것은 *B* 가 왜 단순히 스스로를 위해 돈을 가져가는 것이 아니라는 것은 분명하지 않습니다. 따라서, 관련 당사자가 서로 신뢰하지 않아도 되는 보다 정교한 접근법을 사용해야 합니다.

이것은 다음과 같이 달성될 수 있습니다. *A* 는 큰 난수 *u* 를 생성하고 해쉬v = HASH(u) 를 계산합니다. 그런 다음 그는 *B* 와 함께 그의 결제채널 안에 해쉬 *v* 가 있는 숫자 *u* 가 제시되면 (cf. **5.1.7**) *x* 개의 코인을 B 에게 지불하겠다는 약속을 만듭니다. 이 약속에는 *v* 가 포함되어 있지만 여전히 비밀이 유지되는 *u* 는 포함되어 있지 않습니다.

그 후에, *B* 는 결제채널에서 C 와 유사한 약속을 합니다. 그는 *A* 가 약속한 것과 비슷한 약속의 존재를 알고 있기 때문에 그런 약속을 하는 것을 두려워하지 않습니다. *C* 가 *B* 가 약속 한 *x* 개의 코인을 모으기 위해 해시 문제의 해답을 제시하면 *B* 는 즉시 이 솔루션을 *A* 에 제출하여 *A* 에서 *x* 개의 코인을 수집합니다.

그 다음 *C* 와 *D, D* 와 *E* 의 유사한 약속이 만들어집니다. 약속이 모두 갖추어 졌을 때, *A* 는 솔루션 *u* 를 관련 당사자 또는 *E* 에게만 전달함으로써 전송을 촉발시킵니다.

이 설명에서는 일부 사소한 세부사항이 생략되었습니다. 예를 들어, 이 약속은반드시 다른 만료시간을 가져야 하고, 약속된 금액은 체인을 따라 약간 다를 수 있습니다 (*B* 는 $x - \in$ 코인을 *C* 로 약속할 수 있습니다. 여기서 $\in$ 는 사전합의된 작은 운송료입니다). 우리는 당분간 그러한 세부 사항을 무시합니다. 왜냐하면 결제채널의 작동방식과 TON에서 구현할 수 있는 방법을 이해하기에는 적절하지 않기 때문입니다.

5.2.5. 일련의 결제채널 내 가상 결제채널 (Virtual payment channels inside a chain of payment channels). 이제 *A* 와 *E* 가 서로에게 많은 돈을 지불할 것으로 예상한다고 가정해보십시오. 그들은 블록체인에서 그들 사이에 새로운 결제채널을 만들 수 있지만, 일부 자금이 이 결제채널에 묶여있기 때문에 이것은 여전히 비쌉니다. 또 다른 옵션은 각 지불에 대해 **5.2.4**에 설명된 체인송금을 사용하는 것입니다. 그러나 이것은 관련된 모든 결제채널의 가상 블록체인에서 많은 네트워크 활동과 많은 트랜잭션을 필요로 합니다.

대안은 결제채널 네트워크에서 A 에서 E 를 연결하는 체인 내부에 가상 결제채널을 만드는 것입니다. 이를 위해, A 와 E 는 블록체인에서 결제채널을 만드는 것처럼 결제에 대한 (가상) 블록체인을 만듭니다. 그러나 블록체인에 결제채널 스마트 컨트랙트를 생성하는 대신 A 에서 B 로, B 에서 C 로, 등을 연결하는 모든 중간 결제채널에 A 와 E 로 만든 가상 블록체인에 바인딩된 간단한 결제채널을 만듭니다 . (cf. **5.1.10**). 다시 말해, A 와 E 사이의 최종 합의에 따라 돈을 이체 한다는 약속은 모든 중간 결제채널에 존재합니다.

가상 결제채널이 단방향이라면, 최종 δ 불균형은 양성이 아니므로, **5.2.4**에서 설명한 것과 동일한 순서로 중간 결제채널에서 간단한 결제채널을 생성할 수 있기 때문에 그러한 약속을 아주 쉽게 구현할 수 있습니다. 만료 시간도 같은 방식으로 설정할 수 있습니다.

가상 결제채널이 양방향인 경우 상황은 약간 더 복잡합니다. 이 경우, 최종 합의에 따라 δ 개의 코인을 전송하겠다는 약속은 **5.1.10**에서 설명한 바와 같이 두 개의 반 약속(순방향으로 $\delta - = max(0, -\delta)$ 코인을 전달하고, 역방향으로 $\delta + = max(0, \delta)$ 를 전달하는)을 약속해야 합니다. 이 반-약속(half-promises)은 중간 결제채널에서 독립적으로 생성될 수 있으며, 한 반-약속의 체인은 A 에서 E 방향으로, 그리고 다른 체인은 그 반대 방향으로 생성될 수 있습니다. 이러한 반-약속은 중간 결제채널에서 독립적으로 생성될 수 있으며, 한 약속은 A 에서 E 방향으로, 그리고 다른 체인은 반대 방향으로 생성될 수 있습니다.

5.2.6. 라이트닝 네트워크에서 경로 찾기 (Finding paths in the lightning network). 여태까지 설명되지 않은 한 핵심이 있습니다: A 와 E 가 결제 네트워크에서 그들을 연결하는 경로를 어떻게 찾을 것인가? 결제 네트워크가 너무 크지 않은 경우, OSPF 같은 프로토콜을 사용할 수 있습니다: 결제 네트워크의 모든 노드가 오버레이 네트워크 (cf. **3.3.17**)를 생성한 다음, 모든 노드가 사용 가능한 모든 링크 (즉, 참여하는 결제채널) 정보를 가십 프로토콜을 통해 이웃에게 제공합니다.궁극적으로, 모든 노드는 결제 네트워크에 참여하는 모든 결제채널의 전체 목록을갖게 될 것이며, 그들 자체로 최단 경로를 찾을 수 있게 됩니다 - 예를 들어, 관련된 결제채널의 "수용력"을 고려하여 수정된 Dijkstra의 알고리즘을 적용함으로써 (즉, 그들과 함께 전송될 수 있는 최대 금액) 이루어질 수 있습니다. 후보 경로가 발견되면 전체 경로가 포함된 특수 ADNL 데이터그램을 조사하고 각 중간 노드에게 해당 지불 경로의 존재를 확인하고 그 경로로 이 데이터그램을 추가로 전달할 수 있습니다. 그 후 체인을 구성할 수 있고 체인 전송 (cf. **5.2.4**) 또는 결제채널 체인 (cf. **5.2.5**) 내부에 가상 결제채널을 만드는 프로토콜을 실행할 수 있습니다.

5.2.7. 최적화 (Optimizations). 여기에서 몇가지 최적화가 이루어질 수 있습니다. 예를 들어, 라이트닝 네트워크의 중계 노드 만이 **5.2.6**에서 논의된 OSPF와 같은 프로토콜에 참여해야 합니다. 라이트닝 네트워크를 통해 연결하고자 하는 두 개의 "리프"노드는 자신이 연결되어 있는 중계노드 목록 (즉, 결재 네트워크에 참여하는 결제채널을 설정한 곳)을 서로에게 통신합니다. 그런 다음, 하나의 목록에서 다른목록의 중계 노드로 연결되는 경로는 위에서 **5.2.6**에서 설명한대로 검사할 수 있습니다.

5.2.8. 결론 (Conclusion). 우리는 TON 프로젝트의 블록체인 및 네트워크 기술이 오프-체인 즉시 송금 및 소액결제를 위한 플랫폼인 TON 페이먼트를 작성하는 작업에 적합한지를 설명했습니다. 이 플랫폼은 TON 생태계에 상주하는 서비스에 매우 유용할 수 있으므로 필요할 때 언제 어디서나 소액결제를 쉽게 수집할 수 있습니다.

결론

우리는 사용자 친화적인 인터페이스를 통해 대용량 암호화폐 및 분산형 애플리케이션을 지원할 수 있는 확장 가능한 멀티-블록체인 아키텍처를 제안했습니다.필요한 확장성을 얻기위해 샤딩에 대한 상향식 접근 방식 (cf. **2.8.12** 및 **2.1.2**)이 있는 "단단히 결합된" 멀티-블록체인 시스템 (cf. **2.8.14**)인 TON 블록체인을 제안했습니다.

잠재적인 성능을 더욱 높이기 위해 유효하지 않은 블록(cf. **2.1.17**)을 대체하기 위한 2-블록체인 메커니즘과 샤드간의 신속한 커뮤니케이션 (cf. **2.4.20**)을 위한 인스턴트 하이퍼큐브 라우팅을 소개했습니다.

TON 블록체인과 기존 및 제안된 블록체인 프로젝트 (cf. **2.8, 2.9**)에 대한 간략한 비교는 초당 수백만 건의 트랜잭션을 처리하려는 시스템에 대한 이 접근 방식의 이점을 강조합니다.

3 장에서 설명한 TON 네트워크는 제안된 멀티-블록체인 인프라 스트럭처의 네트워킹 요구를 다루고 있습니다. 이 네트워크 구성요소는 블록체인과 함께 사용되어 광범위한 애플리케이션 및 서비스를 생성할 수 있으며 블록체인만으로는 불가능합니다 (cf. **2.9.13**). 4 장에서 논의된 이러한 서비스에는 사람이 읽을 수 있는 객체 식별자를 주소로 번역하는 서비스인 TON DNS 와; 임의의 파일을 저장하기 위해 사용되는 분산형 플랫폼인 TON 저장소와; 네트워크 액세스를 익명화하고 TON 서비스에 액세스하는 서비스인 TON 프록시와; TON 생태계 전반에 애플리케이션이 소액결제에 사용할 수 있는 실시간 오프-체인 송금을 위한 플랫폼인 TON 페이먼트 (cf. **5**)이 포함됩니다.

TON 인프라는 최종사용자 (cf. **4.3.24**)에게 브라우저 같은 경험을 가능하게 하는 특수한 라이트 클라이언트 월렛과 "톤 브라우저" 데스크톱 및 스마트폰 애플리케이션을 TON 플랫폼에서 암호화폐 결제 및 스마트 컨트랙트와 기타 서비스와의 상호작용을 대량 사용자가 액세스할 수 있게 합니다. 이러한 라이트 클라이언트는 텔레그램 메신저 클라이언트 (cf. **4.3.19**)에 통합될 수 있으므로 결국 수 많은 블록체인 기반 애플리케이션이 수억명의 사용자에게 제공됩니다.

참고문헌

[1] L. Baird, The Swirlds Hashgraph consensus algorithm: fair, fast, Byzan-tine fault tolerance, Swirlds tech report SWIRLDS-TR-2016-01, https: //www.swirlds.com/down-loads/SWIRLDS-TR-2016-01.pdf, 2016.

[2] P. Barreto, M. Naehrig, Pairing-Friendly Elliptic Curves of Prime Order, in: Preneel, B., Tavares, S. (Eds.) SAC 2005. LNCS vol. 3897, p. 319–
331. Springer, 2006. Available at https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/pfcpo.pdf.

[3] K. Birman, Reliable Distributed Systems: Technologies, Web Services and Applications, Springer, 2005.

[4] A. Boldyreva, Threshold signatures, multisignatures and blind signa-tures based on the Gap-Diÿe-Hellman-Group signature scheme, in Inter-national Workshop on Theory and Practice in Public Key Cryptography (PKC) 2003 Proceedings, LNCS vol. 2567, p. 31–46, Springer, 2003.

[5] V. Buterin, Ethereum: A next-generation smart contract and de-centralized application platform, https://github.com/ethereum/wiki/wiki/White-Paper, 2013.

[6] M. Ben-Or, B. Kelmer, T. Rabin, Asynchronous secure computa-tions with optimal resilience, in Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing, p. 183–192. ACM, 1994.

[7] M. Castro, B. Liskov, et al., Practical byzantine fault tolerance, Proceedings of the Third Symposium on Operating Systems Design and Implementation (1999), p. 173–186, available at http://pmg.csail.mit. edu/papers/osdi99.pdf.

[8] EOS.IO, EOS.IO technical white paper, https://github.com/EOSIO/Documentation/blob/-master/TechnicalWhitePaper.md, 2017.

[9] R. Gennaro, C. Gentry, B. Parno, M. Raykova, Quadratic span programs and succinct nizks without pcps. In Advances in Cryptology –Eurocrypt 2013, p. 626–645, 2013.

참고문헌

[10] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, Secure dis-tributed key generation for discrete-log based cryptosystems, Eurocrypt 99, 1999.

[11] D. Goldschlag, M. Reed, P. Syverson, Onion Routing for Anony-mous and Private Internet Connections, Communications of the ACM, 42, num. 2 (1999), http://www.onion-router.net/Publications/CACM-1999.pdf.

[12] F. Hess, N. Smart, F. Vercauteren, The Eta pairing revisited, available at https://eprint.iacr.org/2006/110.pdf.

[13] L. Lamport, R. Shostak, M. Pease, The byzantine generals prob-lem, ACM Transactions on Programming Languages and Systems, 4/3 (1982),
p.382–401.

[14] S. Larimer, The history of BitShares, https://docs.bitshares.org/bitshares/history.html, 2013.

[15] M. Luby, A. Shokrollahi, et al., RaptorQ forward error correction scheme for object delivery, IETF RFC 6330, https://tools.ietf.org/html/rfc6330, 2011.

[16] P. Maymounkov, D. Mazières, Kademlia: A peer-to-peer infor-mation system based on the XOR metric, in IPTPS ’01 revised pa-pers from the First International Workshop on Peer-to-Peer Systems,
p.53–65, available at http://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf, 2002.

[17] P. Maymounkov, Online codes (extended abstract), New York Uni-versity Computer Science, TR2002-833, https://cs.nyu.edu/media/publications/TR2002-833.pdf, 2002.

[18] A. Miller, Yu Xia, et al., The honey badger of BFT protocols, Cryptology e-print archive 2016/99, https://eprint.iacr.org/2016/
199.pdf, 2016.

[19] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, https: //bitcoin.org/bitcoin.pdf, 2008.

참고문헌

[20] S. Peyton Jones, Implementing lazy functional languages on stock hardware: the Spineless Tagless G-machine, Journal of Functional Pro-gramming 2 (2), p. 127–202, 1992.

[21] A. Shokrollahi, M. Luby, Raptor Codes, IEEE Transactions on Information Theory 6, no. 3–4 (2006), p. 212–322.

[22] M. van Steen, A. Tanenbaum, Distributed Systems, 3rd ed., 2017.

[23] The Univalent Foundations Program, Homotopy Type Theory: Univalent Foundations of Mathematics, Institute for Advanced Study, 2013, available at https://homotopytypetheory.org/book.

[24] G. Wood, PolkaDot: vision for a heterogeneous multi-chain frame-work, draft 1, https://github.com/w3f/polkadot-white-paper/raw/master/PolkaDotPaper.pdf, 2016.

컨트A. TON 코인 또는 그램 (The TON Coin, or the Gram)

TON 블록체인, 특히 마스터체인과 기본 워크체인의 핵심 암호화폐는 그램 (*Gram,* GRM)이라고도하는 *TON* 코인 입니다. 이는 밸리데이터가 되기 위해 요구되는 보증금을 만드는데 사용됩니다. 거래 수수료, 가스 지불 (즉, 스마트 컨트랙트 메시지 처리 수수료) 및 영구 저장소 지불도 보통 그램에서 수집됩니다.

A.1. 세분 및 용어 (Subdivision and terminology). 그램은 나노그램, *ngram* 또는단순히 나노라고 불리는 10억 (10^9) 개의 작은 단위로 세분됩니다. 모든 트랜잭션과 계정 잔액은 나노의 음수가 아닌 정수의 배수로 표시됩니다. 다른 단위는 다음과 같습니다.

- 나노 (*nano*), *ngram* 또는 나노그램 (*nanogram*)은 가장 작은 단위이며 10^{-9}그램과 동일합니다.
- 마이크로(*micro*) 또는 마이크로그램(*microgram*)은 천 (10^{-3}) 나노와 같습니다.
- 밀리 (*milli*) 는 백만 (10^6) 나노 또는 1/1,000 (10^{-3})입니다.
- 그램 (*gram*) 은 10억 (10^9) 나노와 같습니다.
- 킬로그램 또는 *kGram*은 천 (10^{-3}) 그램입니다.
- 메가그램 (*megagram*) 또는 *MGram*은 백만 (10^6) 그램 또는 나노와 같습니다.
- 마지막으로, 기가그램 (*gigram*) 또는 *GGram*은 10억 (10^9) 그램 또는 나노와 같습니다.

그램의 초기 공급($5 \cdot 10^9$) 그램 (즉, 5 기가그램)으로 제한되기 때문에 더 큰 단위에 대한 필요성은 없을 것입니다.

A.2. 가스가격을 표현을 위한 더 작은 단위 (Smaller units for expressing gas prices). 더 작은 단위에 대한 필요성이 생기면, 2^{-16} 나노그램에 해당하는 "스펙(speck)"이 사용됩니다. 예를 들어, 가스가격은 스펙으로 표시될 수 있습니다. 그러
나 가스가격과 소비된 가스량의 곱으로 계산된 실제 지불 비용은 항상 가장 가까운 2^{16} 스펙의 반올림되며 나노의 정수로 표시됩니다.

A.3. 초기공급, 채굴보상 및 인플레이션 (Original supply, mining rewards and inflation). 그램의 총 공급량은 원래 5기가그램 (즉, 50억 그램 또는 5·1018 나노)으로 제한됩니다.

이 공급은 새로운 마스터체인과 샤드체인 블록을 채굴하기 위한 밸리데이에 대한 보상이 누적되기 때문에 천천히 증가할 것입니다. 우리는 처음에 밸리데이터가 부지런히 업무를 수행하고 모든 블록에 서명하고, 오프라인 상태가 아니며 유효하지 않은 블록에 서명하지 않는 한, 이 보상이 연간 유효 기간의 20% (정확한 숫자는 향후 조정될 수 있음)가 될 것으로 예상합니다. 이렇게하면 밸리데이터는 증가하는 사용자 트랜잭션을 처리하는데 필요한 더 빠르고 더 나은 하드웨어에 투자할 만큼 충분한 이익을 얻게됩니다.

우리는 평균적으로 그램의 총 공급량의 최대 10%[40] 가 특정 순간에 밸리데이터 지분에 묶일 것으로 기대합니다. 이것은 연간 2%의 인플레이션율을 산출할 것이고, 그 결과로 35년 안에 그램의 총 공급량을 (10기가그램 까지) 두 배로 늘릴 것입니다. 본질적으로, 이 인플레이션은 커뮤니티의 모든 구성원이 시스템을 계속 실행하기 위해 밸리데이터에 지불한 금액을 나타냅니다.

반면에 밸리데이터의 부정행위가 포착되면 처벌로 그 스테이크의 일부 또는 전부를 가져갈 것이며, 그 중 상당 부분이 "태워져" 그램의 총 공급량이 감소합니다. 이는 디플레이션으로 이어질 수 있습니다. 벌금의 작은 부분은 밸리데이터의 부정행위를 증빙한 밸리데이터 또는 "어부"에게 재분배될 수 있습니다.

$$p\,(n\,) \approx 0.1 \cdot (1 + 10^{-9})^{n} \text{ USD}, \tag{26}$$

또는 대략 동등한 금액의 다른 (암호)화폐입니다.

A.4.1. 기하 급수적으로 책정된 암호화폐 가격 (Exponentially priced cryptocurrencies).우리는 그램이 기하 급수적으로 가격이 책정된 암호화폐라고 말합니다. 즉, n-번째 그램의 유통 가격은 적어도 공식에 의해 주어진 "참조 가격" $p\,(n\,)$ 은

[40] 밸리데이터 지분의 최대 총량은 블록체인의 구성 가능한 매개변수이므로 필요한 경우 프로토콜에 의해 이 제한을 적용할 수 있습니다.

$$p(n) = p_0 \bullet e^{\alpha n} \tag{27}$$

구체적인 값은 $p_0 = 0.1$ USD 와 $\alpha = 10^{-9}$ 입니다.

좀 더 정확하게 말하자면, n 개의 코인이 유통되면 새로운 코인의 작은 분수 dn은 (최소한) $p(n)dn$ 달러의 가치가 있습니다. (여기서 n 은 반드시 정수일 필요는 없습니다.)

이러한 암호화폐의 다른 중요한 매개변수에는 순환중인 코인의 총수 n , 존재할 수 있는 코인의 총 수 $N \geq n$ 을 포함합니다. 그램의 경우 초기 수량은 $N = 5 \cdot 10^9$ 입니다.

A.4.2. 첫 번째 n 개의 코인의 총 가격 (Total price of first n coins). 앞으로 유통될지수적으로 가격이 책정된 암호화폐 (예. 그램)의 첫 번째 n개의 코인의 전체 참조 가격 $\mathrm{T}(n) = \int_0^n p(n)\, dn \approx p(0) + p(1) + \ldots + p(n-1)$는 다음과 같이 계산 될 수 있습니다.

$$T(n) = p_0 \bullet \alpha^{-1}(e^{\alpha n} - 1) \tag{28}$$

A.4.3. 다음 개의 코인의 총 가격 (Total price of next coins). n개의 기존의 코인이후에 유통되는 Δn개의 코인의 총 참고 가격 $T(n + \Delta n)$ — $\mathrm{T}(n)$은 다음과 같이계산될 수 있습니다.

$$T(n + \Delta n) - T(n) = p_0 \bullet \alpha^{-1}(e^{\alpha(n+\Delta n)} - e^{\alpha n}) = p(n) \bullet \alpha^{-1}(e^{\alpha \Delta n} - 1) \tag{29}$$

A.4.4. 총 가치 T 를 가진 다음 코인 구매 (Buying next coins with total value T). n 개의 코인이 이미 유통되었고, 새로운 코인을 사는데 T (달러)를 쓰고 싶다고 가정합시다. 새로 획득한 코인 Δn 의 수량은 $T(n + \Delta n) - T(n) = T$ 를 수식 (29)에넣은 다음 계산할 수 있습니다.

$$\Delta n = \alpha^{-1} \log\left(1 + \frac{(T \bullet \alpha)}{p(n)}\right) \tag{30}$$

이는 모든 새로운 코인들이 정확히 그들의 긴준 가격으로 팔리리라는 가정하에 다음과 같이 산출됩니다.

물론, $T \lll p(n)\alpha^{-1}$, 이면 $\Delta n \approx T/p(n)$ 입니다.

A.4.5. 그램의 시장가격 (Market price of Grams). 물론, n개의 그램이 유통된후자유 시장 가격이 $p(n) = p_0 e^{\alpha n} \approx 0.1 \bullet (1 + 10^{-9})^n$ 아래로 떨어지면 TON 리져브에서 새로운 그램을 구입할 가능성이 줄어듭니다; 그들은 그램의 총량을 증가시키지 않으면서 자유시장에서 자신의 그램을 구입하기로 결정할 것입니다. 반면 TON리져브의 그램을 판매할 수 있는 능력은 그램 가격의 상승을 제한할 수 있습니다.

이것은 그램의 시장 가격이 급격한 상승 (및 하락)에 덜 취약하다는 것을 의미합니다. 지분 (밸리데이터의 보증금)이 적어도 한달 동안 동결되고 가스 가격이 너무 빠르게 변하지 않는다면 블록체인이 보다 원활하고 예측 가능하게 작동하기 때문에 이것은 중요합니다.

A.4.6. 그램 환매 (Buying back the Grams). 그램의 시장가격이 $0.5 \cdot p(n)$ 아래로 떨어지면 총 n 그램이 유통될 때 (즉, TON 리져브에 의해 관리되는 특별 계정에 보관되지 않은 경우), TON 리져브는 그램을 환매할 권리를 보유하고 그램은 유통 중인 그램의 총량을 n 으로 줄입니다. 이렇게 하면 그램 환율이 갑자기 떨어지는 것을 방지할 수 있습니다.

A.4.7. 더 높은 가격으로 새로운 그램 판매 (Selling new Grams at a higher price). TON리져브는 잔여 그램을 전혀 팔지 않거나, $p(n)$ 보다 높은 가격으로 판매할 권리를 보유하지만 절대로 더 저렴한 가격으로 판매할 수는 없습니다 (환율이 급변하는 불확실성을 고려하여).

A.5. 할당되지 않은 그램 사용 (Using unallocated Grams). TON 리져브는 할당되지 않은 그램 (대략 $5 \cdot 10^9 - n$ 그램)은 어떤 목적으로도 사용하지 (TON 리져브특별계정에 명시된 계정 및 연결된 일부 다른 계정) 않으며 **A.4, A.4.7** 및 **A.6**에서 설명한 대로 할당되지 않은 사전 정의된 그램은 **A.5.1** 및 **A.5.2**에서 설명한 대로 개발자 및 생태계 인센티브로 이전합니다.

A.5.1. 비할당 그램의 일부를 개발자들에게 제공 (Some unallocated Grams will be given to developers). 사전정의된 양의 할당되지 않은 그램 (예. 총 공급량의 4%에 해당하는 200메가그램)은 TON 블록체인을 배포하는 동안 오픈소스 TON소프트웨어의 개발자에게 지급되는 "보상"으로 이전되며 가득 기간은 4년입니다.

A.5.2. 생태계 인센티브를 위해 비할당 일부 그램을 예비 (Some unallocated Grams will be reserved for ecosystem incentives). 미리 정의된 양의 그램 (예. 초기 총 공급량의 10%에 해당하는 500 메가그램)은 삼자 밸리데이터 및 TON 저장소 및 TON 프록시노드의 설치를 장려하기 위해 "생태계 인센티브"용으로 예약됩니다 (예를 들어 TON블록체인의 이전 블록을 저장하거나 서비스의 선택된 서브세트의 네트워크 트래픽을 프록싱하는데 비용을 지불해야 합니다). 이러한 인센티브의 또 다른 예는 TON 월렛을 활성화한 사용자에게 보람을 줄 수 있습니다.

A.5.3. TON 리져브 기준가격의 증가 (Increase of the TON Reserve reference price). TON 생태계 (cf. **A.5.2**)와 TON 개발자-풀 (cf. **A.5.1**)에 대한 할당이 TON블록체인의출시와 함께 이루어진 후에 그램의 TON 리져브 기준 가격 $p(n)$은 미리 계산할 수 있는 일정 금액만큼 즉시 상승합니다.

나머지 할당되지 않은 그램은 **A.5**에서 설명한대로 TON 리져브에 사용됩니다.

A.6. 그램의 대량판매 (Bulk sales of Grams). 많은 사람들이 동시에 대량의 그램을 구매하기 원할 때 주문을 즉시 처리하지 않는 것이 좋습니다. 특정 주문 및 처리순서의 타이밍에 따라 결과가 크게 달라질 수 있기 때문입니다.

대신 그램 구매 주문은 사전정의된 기간 (예를 들어 하루 또는 1개월) 동안 수집된 다음 한 번에 모두 처리될 수 있습니다. i - 번째 차액의 T_i 달러를 가진 k 주문이 도착하면 총 차액 $T = T_1 + T_2 + \ldots + T_k$를 사용하여 (30)에 따라 n개의 새로운 코인을 구매하고 i - 번째 보낸사람이 코인들의 $\Delta n \, T_i / T$ 가 할당됩니다. 이런식으로 모든 구매자는 그램 당 $T/\Delta n$ 달러의 동일한 평균 가격으로 그램을 얻습니다.

그 후 새로운 그램 구입을 위한 새로운 수집 주문이 시작됩니다. TON 출시와 그램의 초기배포 이후 이 대량판매 시스템은 식 (30)에 따라 TON 리져브에서 그램을 즉시 판매하는 시스템으로 대체될 것으로 예상됩니다.