

모나드(Monad)

해당 번역본은 이용자가 디지털자산 관련 정보를 편리하게 확인할 수 있도록
단순 참고용으로 번역된 것이며, 투자 권유를 목적으로 하지 않습니다.
그 내용이 정확하지 않을 수 있으므로 정확한 정보 습득을 위해서는
원문을 참고하시거나 원문 작성 주체 측에 문의하시기 바랍니다.
두나무는 해당 번역본 내용의 진실성 및 정확성을 보장하지 않고,
제공된 정보에 의한 투자 결과에 대하여 어떠한 책임도 지지 않으며,
투자 손실은 투자자에게 귀속됩니다.

MonadBFT: 빠르고, 반응성이 높으며, 포크 저항적인 간소화된 합의 알고리즘

Mohammad Mussadiq Jalalzai, Kushal Babel

Category Labs

(이전 명칭 Monad Labs)

초록

이 논문은 성능과 확장성을 모두 크게 향상시키도록 설계된 새로운 비잔틴 결함 허용(BFT) 합의 프로토콜인 MonadBFT를 소개한다. MonadBFT는 정상 경로에서 메시지 및 인증자 복잡도에 있어 선형성을 달성하여 탈중앙화를 향상시킨다. 이는 단일 라운드 내에서 추측적(finality) 확정을 달성하며, 낙관적으로 반응성을 보인다. 추측적 메커니즘은 블록 이중 제안(block equivocation)만이 추측적 실행을 되돌릴 수 있도록 정교하게 설계되어, 프로토콜이 악의적인 행위에 대한 책임성을 보장할 수 있게 한다. MonadBFT의 주목할 만한 혁신 중 하나는 *테일 포킹(tail-forking)*으로 알려진 특정 형태의 최대 추출 가능 가치(Maximal Extractable Value, MEV) 취약성에 대한 내재적 저항성이다. 테일 포킹은 악의적인 리더가 선행 리더의 블록으로부터 분기(fork)하여 해당 블록이 버려지고 선행 리더가 보상을 받지 못하게 하는 경우 발생한다. 이로 인해 악의적인 리더는 거래를 재정렬하거나, 훔치거나, 조작하여 MEV 착취를 악화시킬 수 있다. MonadBFT는 이러한 취약성을 효과적으로 완화하여 거래 처리의 공정성과 무결성을 보장한다. 우리의 지식에 따르면, 이러한 모든 특성을 결합한 파이프라인 기반 리더형 BFT 합의 프로토콜은 존재하지 않는다.

1 서론

비잔틴 결함 허용(Byzantine Fault Tolerant, BFT) 합의 프로토콜은 일부 참여자가 임의로 이탈하더라도 네트워크 참여자들이 합의에 도달할 수 있도록 한다. 블록체인 이전의 BFT 합의 알고리즘은 주로 하드웨어나 소프트웨어 오류(예: 항공기 제어 시스템 [32]) 및 분산 컴퓨터 시스템 내의 악의적 공격 [10]을 처리하기 위해 사용되었다. 블록체인 맥락에서의 합의 프로토콜 사용은 새로운 공정성 및 설계상의 과제를 도입한다. 이는 참여자가 자신의 행동에 따라 보상을 받거나 페널티를 부과받기 때문이며, 또한 수백 개의 노드(또는 검증자)가 프로토콜에 참여함에 따라 확장성 측면에서도 고려가 필요하다.

PBFT [10]와 같은 전통적인 BFT 프로토콜은 합의를 주도하는 고정된 리더를 가지지만, 블록체인에서 사용되는 프로토콜은 단일 느린 리더에 대한 공정성과 강건성을 확보하기 위해 리더를 회전시킨다 [6]. 이후 HotStuff [35]는 확장성을 위한 두 가지 핵심 발전을 개척하였다: (1) 회전하는 리더 간의 커밋 결정 파이프라이닝, (2) 선형 통신 복잡도. 파이프라이닝(또는 제안의 체이닝)은 처리량을 두 배 이상($2\times$ 이상) 증가시키며, 선형 통신 복잡도는 블록체인이 더 많은 노드로 확장될 수 있게 하여 탈중앙화라는 핵심 속성을 강화한다. HotStuff 계열의 초기 프로토콜들 [16, 22, 35]은 또 다른 바람직한 특성인 낙관적 반응성(optimistic responsiveness)을 가진다. 즉, 누락 결함(omission fault)을 감지할 때를 제외하고는 타임아웃을 기다리지 않고 네트워크 속도로 진행할 수 있는 능력이다. 정확한 타임아웃 설정에 대한 의존도를 줄이는 것은 실제 구현에서 매우 중요하다. 지나치게 큰 타임아웃은 확정(finalization) 지연(latency)을 증가시킬 수 있으며, 특히 반응성이 없는 프로토콜에 큰 영향을 미친다. 반대로 너무 짧은 타임아웃은 모델 가정을 위반할 위험이 있다. 예를 들어, 인증서의 적시 수신에 의존하는 프로토콜은 타임아웃이 너무 낮게 설정되면 인증서가 누락되어 리더의 제안이 다수의 검증자에게 거부될 수 있다. 이러한 상황은 일부 검증자만이 더 엄격한 시간 제약을 충족시킬 수 있게 되어 중앙화 효과를 초래할 수 있다. 반면 반응형 프로토콜은 주로 누락 결함을 감지하기 위해 타임아웃을 사용하므로 더 큰 타임아웃 값을 설정할 수 있으며, 이러한 결함은 일반적으로 드물다고 가정한다.

테일 포킹(Tail-forking). 최근 연구 [17]는 *테일 포킹*이라 불리는 파이프라인 BFT 프로토콜의 공정성과 생존성(liveness) 취약성을 드러냈다. 테일 포킹은 그림 1에 나타난 것처럼 리더 L3과 같은 악의적인 리더가 선행 리더(L2)가 제안한 블록으로부터 의도적으로 포크하여 해당 블록이 버려지고 L2가 블록체인 보상을 받지 못하게 하는 경우 발생한다. 또한 L3은 L2의 블록에 포함된 거래를 훔치거나 재정렬하며, 자신의 거래를 삽입하여 L2의 거래를 이용할 수 있다. 이러한 현상은 일반적으로 *최대(이전에는 채굴자) 추출 가능 가치(Maximal (formerly, miner) Extractable Value, MEV)* [2, 12]라고 불리며, 지금까지 수억 달러 규모의 착취적 이익을 초래하였다. 테일 포킹은 여러 블록 간의 거래 재정렬을 가능하게 하여 MEV 문제를 더욱 악화시킨다.

지연 시간(Latency). HotStuff 계열의 프로토콜은 선형 통신 복잡도로 인해 (예: PBFT와 비교하여) 더 높은 지연 시간을 감수한다. HotStuff의 경우 3.5 라운드 트립, Fast-HotStuff [22] 및 Jolteon [16]의 경우 2.5 라운드 트립으로, 이는 결함이 있는 리더로부터 회전하는 과정에서 더 높은 통신 비용을 수반한다. HotStuff-1 [24] 및 HotStuff-2 [28]와 같은 최근의 시도는 (HotStuff-1의 경우 추측적 접근을 통해) HotStuff의 지연 시간을 줄이지만, 불행히도 반응성을 상실한다.

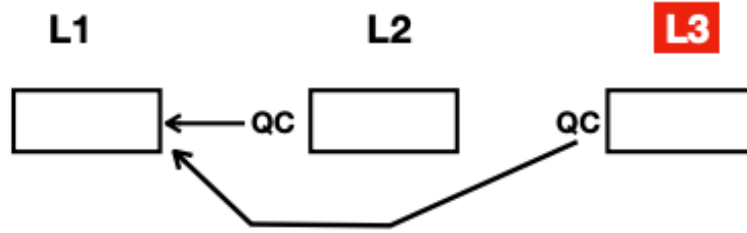


Fig. 1. Tail-Forking attack by the leader L3

그림 1. 리더 L3에 의한 테일 포킹 공격

1.1 MonadBFT

본 연구에서는 앞서 소개한 모든 바람직한 특성을 달성하는 MonadBFT를 제안한다. MonadBFT는 HotStuff 계열의 프로토콜과 유사한 파이프라인 구조의 프로토콜로, 정직한 리더가 있는 경우(*happy path*)에는 안정적인 동작 상태를 가지며, 결함 있는 리더로부터 복구하기 위한 *unhappy path*를 가진다. MonadBFT는 클라이언트가 단일 라운드 내에서 낙관적 조건하에 자신의 거래가 확정(*finalized*)되었는지를 검증할 수 있게 한다. 또한 MonadBFT는 정상 경로에서 선형 통신 복잡도를 유지하고, 결함 있는 리더로부터 복구하는 과정에서도 (낙관적) 반응성을 유지하며, 체인 재조직 공격 — 일명 테일 포킹(*tail-forking*) [17, 24] —에 대해 강력한 저항성을 제공한다. MonadBFT의 추측적 합의(*speculative agreement*)는 리더가 이중 제안(*equivocation*)을 하는 경우에만 되돌려지며, 이 경우 리더는 블록체인 프로토콜에 의해 확실히 페널티를 받는다.

우리가 아는 한, 이러한 특성들의 조합을 제공하는 BFT 합의 프로토콜은 지금까지 존재하지 않았다. 이제 MonadBFT가 이러한 네 가지 특성과 관련하여 어떻게 이를 달성하는지에 대한 세부 설명과 직관을 제시한다.

빠른 합의와 낙관적 선형 복잡도: MonadBFT는 기존 프로토콜이 설정한 지연 시간(*latency*) 지표를 크게 개선한다. 확정을 달성하는 데 필요한 라운드 수를 줄임으로써, 낙관적 실행 지연 시간은 HotStuff [34] 대비 60%, 더 빠른 변형들 [3, 22] 대비 40% 감소한다. HotStuff-1 [24]과 유사하게, MonadBFT는 클라이언트를 합의의 주요 구성원으로 취급하여 선형 통신 복잡도로 단일 라운드의 추측적 확정을 달성한다. 그러나 HotStuff-1은 상위 블록이 이미 커밋된 경우에만 블록을 추측적으로 확정할 수 있는 반면([24]에서 “*prefix speculation dilemma*”라 언급됨), MonadBFT는 상위 블록의 커밋을 기다리지 않고도 블록을 추측적으로 확정할 수 있다.

선형 복잡도는 합의 프로토콜에서 매우 중요한 속성으로, 더 큰 검증자 집합으로의 확장을 가능하게 하여 탈중앙화라는 블록체인의 핵심 가치를 강화한다. 또한 선형 복잡도는 검증자의 진입 장벽을 낮추고, 더 넓은 참여와 비용 절감을 촉진한다. Fast-HotStuff와 Jolteon과 마찬가지로, MonadBFT는 정상 경로에서 선형 복잡도를 우선시하며, 두 가지 주요 이유로 비정상 경로에서는 이차(*quadratic*) 통신 복잡도를 가진다. 첫째, 프로토콜이 대부분의 시간 동안 정상 경로에서 작동할 것으로 예상되므로, 이 단계에서 오버헤드를 최

소화하는 것이 중요하다. 특히 지분증명(proof-of-stake) 블록체인 맥락에서는 프로토콜이 부과하는 금전적 페널티로 인해 결함 있는 리더가 매우 드물다 [20, 30]. 둘째, 비정상 경로가 이차 복잡도를 보이더라도 향상된 반응성을 가지며, 글로벌 안정화 시간(Global Stabilization Time, GST) 이후 Bracha 메커니즘 [4, 5]을 이용하여 $O(\Delta)$ 시간 내에 검증자들이 수렴할 수 있게 되어, 빠른 복구가 가능하다 [3].

테일 포킹에 대한 강인성: 테일 포킹(그림 1)은 여러 선행 연구 [17, 22, 29]에서 설명되었듯, 파이프라인 합의 프로토콜 내의 심각한 취약성을 나타낸다.

이러한 취약성은 최근 분석 [13]에서 확인된 바와 같이, 낙관적 경로에서 선형 통신 복잡도를 사용하는 모든 파이프라인 합의 프로토콜(즉 MonadBFT 이전의 프로토콜)에 내재되어 있다. 근본적인 문제는 프로토콜 설계에서 후속 리더가 이전 리더의 블록에 대한 투표를 집계하는 책임을 지는 구조에 있다. 악의적인 리더는 이를 악용하여 필요한 투표를 고의로 숨김으로써 이전 블록이 체인에 추가되는 것을 방해할 수 있다. 이러한 전략적 생략은 일시적 검열을 초래할 뿐만 아니라, 악의적인 리더가 최대 추출 가능 가치(MEV) 공격을 수행할 수 있도록 하여 네트워크 중앙화 위험을 심화시키고 프로토콜의 무결성을 훼손한다. MonadBFT는 이러한 공격에 대해 강인한 선형 정상 경로를 가진 최초의 파이프라인 합의 프로토콜이다. BeeGees [17]는 파이프라인 프로토콜에서 테일 포킹을 처음으로 방지했지만, 일반적인 경우에서도 이차 복잡도를 초래한다. 본 연구에서는 BeeGees에서 제안된 AHL (Any Honest Leader) 속성이 테일 포킹 저항성을 달성하기에는 지나치게 엄격하며, 이를 완화하면 일반적인 경우에 선형 복잡도 설계를 가능하게 하여 실제 사용이 가능함을 관찰한다.

낙관적 반응성: 합의 프로토콜에서 낙관적 반응성이란 리더가 정직한 경우, 프로토콜이 최대 메시지 지연 시간($\Delta > \delta$)을 기다리는 대신 네트워크 속도(δ)로 진행할 수 있는 능력을 의미한다. 네트워크 지연에 근접하여 동작하는 고성능 프로토콜은 시간 의존성이 아닌 이벤트 발생에 의해 구동되므로 본질적으로 강건하다. 반응성이 없는 프로토콜은 짧은 대기 시간을 사용하여 일정 수준의 성능을 낼 수 있지만, 다수의 검증자가 효과적으로 동기화하고 라운드를 전환할 수 있도록 신중한 관리가 필요하다. 더 긴 대기 시간 Δ 는 신뢰성을 높이지만, 프로토콜을 느리게 만든다. MonadBFT는 정상 경로와 비정상 경로 모두에서 타임아웃에 의존하지 않으며, 대신 결함이 감지된 후 $2f + 1$ 명의 검증자가 응답하면 라운드가 즉시 진행된다.

이후 섹션에서는 MonadBFT가 단일 라운드 추측적 합의를 달성하는 방법을 자세히 설명하며, 빠른 합의를 위한 기초를 제시한다.

1.2 단일 라운드 추측 실행의 직관

Fast-HotStuff [22] 및 Diem-BFT [3]와 같은 2라운드 합의 프로토콜은 예비 잠금(locking) 단계를 거쳐 커밋 라운드를 수행해야 하는 반면, MonadBFT는 단일 라운드 내에서 추측적으로 합의를 달성한다. 전통적인 2라운드 프로토콜에서는 첫 번째 라운드가 제안 p 를 정족수 증명서(QC)와 함께 잠그는 역할을 하며, 그 다음 라운드에서 해당 잠긴 제안을 커밋한다. 제안 p 의 커밋은 최소 $2f + 1$ 명의 검증자에 의해 잠겼음을 보장하며, 그 중 적어도 $f + 1$ 명은 정직하다. 이 메커니즘은 일단 $f + 1$ 명의 정직한 검증자가 제안을 잠그면, 그 잠금이 생성된 모든 타임아웃 증명서(TC)에 포함되므로 잠금을 해제할 수 없게 하여, 다른 제안이 정족수를 획득하지 못하게 한다.

MonadBFT는 이러한 2단계 접근 방식을 지원하면서도, 검증자가 블록을 추측적으로 실행하고 단일 라운드 통신 후(블록에 대한 QC를 수신한 시점) 클라이언트에게 예비 확정(preliminary confirmation)을 전송할 수 있도록 한다.

리더로부터 블록을 수신한 후, 검증자는 리더가 이중 제안을 하지 않는다는 가정하에 제안에 낙관적으로 잠금(lock)을 건다. 이러한 조건에서 검증자가 이전 뷰의 블록에 대한 QC를 수신하면, 다수의 정직한 검증자가 해당 제안에 잠금되었고, 상충되는 제안이 존재하지 않음을 확신할 수 있다. 따라서 검증자는 제안을 추측적으로 실행하고 클라이언트에게 조기 확정(응답 메시지)을 보낸다. 클라이언트는 동일한 뷰(라운드 번호)에서 $2f + 1$ 개의 조기 확정 응답을 수신하면 해당 제안(자신의 거래를 포함한 제안)을 확정된 것으로 간주한다. 비정상 경로—즉 리더 실패가 발생할 경우—제안 헤더 또는 *tip*은 타임아웃 메시지에 포함된다. 이를 통해 후속 리더는 프로토콜을 재개할 때, 다수의 정직한 검증자가 마지막으로 투표한 제안을 반드시 인지하게 된다. 원래 제안자가 이중 제안을 하지 않은 경우, 해당 제안은 후속 리더에 의해 반드시 재제안되어야 한다. 결과적으로 MonadBFT는 테일 포킹 공격을 완화한다. 즉, 정직한 노드가 한 제안(특히 GST 이후)은 이후의 실패로 인해 버려지지 않는다(정리 1). MonadBFT의 테일 포킹 저항성은 제안에 대한 추측이 이중 제안의 경우에만 되돌려짐을 직접적으로 의미한다(따름정리 1).

비잔틴 리더가 동일한 뷰 번호와 시퀀스 번호(즉 블록체인 높이)에 대해 두 개의 상충되는 제안을 제시할 경우, $2f$ 이하의 검증자만 조기 확정을 제공한 경우 제안이 되돌려질 수 있다. 이러한 제안은 클라이언트에 의해 커밋되거나 확정된 것으로 간주되지 않음을 유의해야 한다. 클라이언트는 최소 $2f + 1$ 명의 검증자로부터 조기 확정을 받았을 때만 제안을 확정(또는 확정된 것으로 간주)하며, 이는 $f + 1$ 개의 서명이 필요한 비추측적 커밋 프로토콜과 달리 안전성 실패로 이어질 수 있는 부분적 승인으로부터 보호한다. 또한, 이중 제안의 사례는 쉽게 식별 가능하여, 위반자에 대한 스테이킹 슬래싱과 같은 신속한 제재 조치를 가능하게 한다.

2 관련 연구

MonadBFT는 정상 경로에서 선형 복잡도를 특징으로 하는 HotStuff 계열의 프로토콜에 속한다. 따라서 MonadBFT는 동일한 계열의 다른 프로토콜과 비교하는 것이 자연스럽다. 밀접하게 관련된 동시 연구로는 HotStuff-1 [24]이 있으며, 이는 연속된 라운드에서 올바른 리더가 존재하고 추측된 블록의 상위 블록이 커밋된 경우 추측적으로 빠른 합의를 달성한다. HotStuff-1은 정상 및 비정상 경로 모두에서 선형 통신 복잡도를 유지한다. 그러나 HotStuff-1은 파이프라인 방식으로 동작할 수는 있지만, 기본 변형(비파이프라인)으로 되돌아가 여러 슬롯 동안 동일 리더가 담당하는 슬롯팅(slotting)을 사용하지 않는 한, 테일 포킹 공격에 대해 본질적인 강인성을 제공할 수 없다. 테일 포킹 문제는 리더가 회전할 때 본질적으로 발생하므로, 슬롯팅 메커니즘은 근본적인 해결책이 아니라 단지 문제를 지연시키는 방법일 뿐이다. 또한 슬롯팅은 바람직하지 않은 부작용을 수반한다. 슬롯팅이 효과적이려면 타임아웃 기간이 충분히 길어야 하며, 리더는 뷰 내에서 마지막 제안(last-slot proposal)의 시점을 정확히 예측해야 한다. 그렇지 않으면 포크가 여전히 발생할 수 있다. 또한 더 긴 타임아웃 기간은 장기적인 검열 위험을 높이고, 리더 실패 시 지연을 증가시킨다. 슬롯팅은 계획된 리더 회전 상황에서도 낙관적 반응성을 제공하지 못한다. 마지막으로, HotStuff-1은 상위 블록이 커밋된 경우에만 블록을 추측할 수 있는 “prefix speculation dilemma” 문제를 겪는다.

BeeGees [17]는 빠른 합의와 테일 포킹에 대한 강인성을 모두 달성하지만, 일반적인 경우 높은 비트 복잡도로 인해 실질적 배포에 제약이 있다. BeeGees의 검증자들은 투표 중 전체 블록을 브로드캐스트하여 비효율성을 초래하며, 또한 실패 이후에는 반응성을 보이지 않는다.

표 1. 체인 기반 회전 리더 BFT SMR 프로토콜 비교

프로토콜	최소 커밋 지연	테일 포킹 내성	파이프라인	정상 경로 통신 복잡도	비정상 경로 통신 복잡도	낙관적 반응성	블록 복구
HotStuff [28]	7δ	$\times$	$\checkmark$	$O(n)$	$O(n)$	$\checkmark$	N/A
Fast-HotStuff [22]	5δ	$\times$	$\checkmark$	$O(n)$	$O(n^2)$	$\checkmark$	N/A
Jolteon [16]	5δ	$\times$	$\checkmark$	$O(n^2)$	$O(n^2)$	$\checkmark$	N/A
HotStuff-2 [28]	5δ	부분적 (1)	$\checkmark$	$O(n)$	$O(n)$	$\times$	N/A
PaLa [11]	4δ	부분적 (1)	$\checkmark$	$O(n)$	$O(n^2)$	$\times$	N/A
Banyan [31]	3δ	$\times$	$\times$	$O(n^2)$	$O(n^2)$	$\times$	N/A
BeeGees [17]	3δ	$\checkmark$	$\checkmark$	$O(n^2)$ (2)	$O(n^2)$	$\times$	$O(n^2)$ (3)
VBFT [21]	2δ	$\checkmark$	$\checkmark$	$O(n^2)$	$O(n^2)$	$\checkmark$	$O(n)$
Moonshot [13]	3δ	부분적 (1)	$\checkmark$	$O(n^2)$	$O(n^2)$	$\checkmark$	N/A
HotStuff-1 (파이프라인) [24]	3δ (4)	$\times$ (5)	$\checkmark$	$O(n)$	$O(n)$	$\times$	N/A
본 논문	3δ (4)	$\checkmark$	$\checkmark$	$O(n)$	$O(n^2)$	$\checkmark$	$O(1)$

- (1) 블록에 대한 QC를 $f + 1$ 명의 올바른 검증자가 보유한 경우에만 보장된다.
- (2) 각 검증자가 블록을 브로드캐스트하여 매우 높은 통신 복잡도를 초래한다.
- (3) 블록이 브로드캐스트되는 투표 단계 동안 페이로드 복구가 암묵적으로 포함된다.
- (4) 추측 실행으로 인해 지연이 최소화된다.
- (5) HotStuff-1의 파이프라인 변형은 동일 리더의 연속 블록 제안(“슬롯팅”)으로 테일 포킹을 회피하지만, 리더 회전 시 취약해진다.

Moonshot 프로토콜 [13] 역시 단일 라운드 지연을 달성한다. 그러나 Moonshot은 정상 경로에서 $O(n^2)$ 메시지 복잡도에 의존하고, 테일 포킹 공격에 대해서는 부분적 보호만 제공하여 완전한 내성을 결여한다. 이에 반해, MonadBFT는 한 블록이 다수의 검증자에 의해 지지되면 QC가 형성되지 않았더라도 테일 포킹에 대해 견고함을 보장한다.

VBFT는 또 다른 프로토콜로, 두 번의 통신 단계에서의 추측 실행을 통해 빠른 합의를 달성한다. 그러나 정상 경로에서 $O(n^2)$ 메시지 복잡도를 가지며, 페이로드 복구에는 $O(n)$ 복잡도를 가진다. 반면 MonadBFT는 정상 경로에서 $O(n)$ 메시지 복잡도를 유지하고, $O(1)$ 페이로드 복구를 달성하며, 테일 포킹 내성에 대한 형식적 증명을 제공한다.

HotStuff 계열의 다른 프로토콜들—HotStuff [28], Fast-HotStuff [22], Jolteon [16], HotStuff-2 [28], PaLa [11]—은 더 높은 커밋 지연을 보이며, 테일 포킹에 대한 포괄적인 강인성을 결여한다.

실용적 비잔틴 결함 허용(PBFT) 프로토콜 중 PBFT with Tentative Execution(PBFT-TE) [9]는 패스트 패스 합의를 목표로 하지만 패스트 패스가 실패할 경우 전통적인 PBFT로 되돌아간다. PBFT-TE는 정상 경로에서 $O(n^2)$ 메시지 복잡도를 가지며 파이프라인이 아니다. 또한 블록에 대한 QC를 $f + 1$ 명의 올바른 검증자가 보유한 경우에만 테일 포킹을 방지한다. 유사하게 SBFT [18]와 Zyzzyva [25]는 모든 복제본이 올바른 경우에만 낙관적으로 동작하며, 실패가 발생하면 다중 라운드 프로토콜로 되돌아간다.

3 시스템 모델 및 정의

참여자

시스템은 검증자라고도 불리는 n 명의 참여자로 구성되며, 이 중 최대 f 명이 비잔틴일 수 있다. 비잔틴 검증자는 잘못되거나 상충하는 정보를 보내거나 침묵하는 등 임의로 행동할 수 있다. 그들은 또한 프로토콜을 깨기 위해 담합할 수 있다. 프로토콜이 최대 f 개의 비잔틴 실패를 허용하기 위해서는 $n \geq 3f + 1$ 이 가정된다 [26].

통신

검증자들은 부분 동기 네트워크에서 통신한다고 가정한다. 즉, 알 수 없는 전역 안정화 시간(GST) 이후에는 네트워크가 모든 메시지를 유한 지연 Δ 내에 전달한다 [14]. GST 이전에는 네트워크가 메시지를 임의로 지연시킬 수 있다.

암호학적 도구

- **디지털 서명:** 각 검증자는 자신의 공개-비공개 키 쌍과 모든 다른 검증자의 공개 키를 보유한다. 메시지는 비공개 키로 서명되고, 해당 공개 키로 서명이 검증된다.
- **서명 집계:** 프로토콜은 다수 검증자의 지지를 단일 검증 가능한 서명으로 결합하기 위해 집계 서명을 활용하여 통신 및 저장 오버헤드를 줄인다.

3.1 데이터 구조

블록과 BlockHeader

블록은 $\langle \text{seq}, \text{payload}, \text{payload_hash}, \text{qc}, \text{block_hash} \rangle$ 로 형식적으로 표현된다. block_hash 는 시퀀스 번호 seq , payload_hash , 그리고 블록에 포함된 정족수 증명서(QC)의 해시이다.

BlockHeader 는 payload 가 생략된 Block 으로 정의된다.

제안(Proposal)

블록 제안, 또는 간단히 *Proposal*은 하나의 블록을 캡슐화한다. 이는 $\langle \text{view}, \text{proposal_id}, \text{block}, \sigma, \text{tc}, \text{nec} \rangle$ 로 표기되며, 여기서 proposal_id 는 $\langle \text{block.block_hash}, \text{view} \rangle$ 의 해시이고 σ 는 제안 리더 L 의 proposal_id 에 대한 서명이다. 제안은 선택적으로 타임아웃 증명서(TC)를 가질 수 있으며, No-Endorsement 증명서(NEC)가 동반될 수 있다. TC는 반드시 직전 뷰에서 나와야 하고, NEC는 제안과 동일한 뷰에서 나와야 한다.

- **신규 제안(Fresh Proposal):** 이전의 실패한 제안들에 영향받지 않은 새로운 블록을 제시하는 제안. 신규 제안 p 는 자신이 보유한 QC와 연속된 뷰 번호를 가지거나 NEC를 동반한다(즉, $p.\text{view} = p.\text{block.qc.view} + 1$ 또는 $p.\text{nec} \neq \perp$).
- **재제안(Reproposal):** 이전 신규 제안의 블록을 싣고 이를 소생시키거나 최종화하려는 제안. 재제안 p 의 경우 $p.\text{view} > p.\text{block.qc.view} + 1$ 이고 $p.\text{nec} = \perp$ 이다.

투표와 정족수 증명서(QC)

투표는 $\langle \text{view}, \text{seq}, \text{proposal_id}, \text{state_hash}, \sigma \rangle$ 로 표현된다. 정족수 증명서(QC)는 최소 $2f + 1$ 명의 검증자로 이루어진 정족수의 투표 모음으로, 하나의 제안을 인증한다. $\langle \text{view}, \text{seq}, \text{proposal_id}, \text{state_hash}, \Sigma \rangle$ 로 표기되는 QC는 특정 라운드 view 동안 최소 $2f + 1$ 명의 검증자가 시퀀스 번호(즉 블록체인 높이) seq 에서 블록 b 를 포함하는 제안(proposal_id 로 식별됨)에 투표했음을(개별 서명 σ 에 대한 집계 서명 Σ 를 통해) 암호학적으로 증명한다. 우리는 QC가 제안 p 와 그 제안 내의 블록 $p.\text{block}$ 을 가리킨다고 말한다. 이 증명서는 제안의 실행이 state_hash 로 표시되는 해당 상태를 산출했음을 검증한다. p 의 부모는 $p.\text{block.qc}$ 가 가리키는 제안이다.

Tip

제안 p 의 tip(또는 헤더)은 $\langle \text{view}, \text{proposal_id}, \text{block_header}, \sigma, \text{nec} \rangle$ 로 표기되며, $p.\text{tc}$ 와 $p.\text{block.payload}$ 를 제외한 제안의 모든 필드를 포함한다. 각 검증자는 자신이 투표한 최신 신규 제안의 tip만 유지한다. 또한 검증자는 TC의 *high_tip*(아래 설명)을 사용하여 자신의 로컬 tip을 동기화할 수 있다.

High Tip

신규 제안들의 tip 집합 $\{tip_1, tip_2, \dots, tip_j\}$ 가 주어졌을 때, 그중 *high tip*은 $1 \leq i \leq j$ 에 대해 뷰 번호 $tip_i.view$ 가 가장 높은 tip_i 로 정의된다. 실패 이후에는 최초로 선출된 리더가 최소 $2f + 1$ 명의 검증자의 tip을 포함하는 TC를 구축하고, *high tip*에 해당하는 블록을 담은 재제안을 만드는 책임이 있다.

무지지 메시지와 무지지 증명서(NEC)

테일 포킹을 방지하기 위해, 이전 뷰가 실패했다면 리더는 제안을 복구할 수 없는 경우를 제외하고 *high tip* 제안을 재제안해야 한다. 이때 리더가 신규 제안으로 진행할 수 있도록 No-Endorsement 증명서(NEC)가 구성된다. 이를 위해 리더는 검증자들이 제안을 복구할 수 없을 때 보내는 No-Endorsement 메시지 $\langle v, highqc.view, \sigma \rangle$ (NE)를 수집한다. 이 메시지는 현재 뷰와 검증자가 수신한 최고 QC의 뷰, 그리고 이 두 필드에 대한 서명으로 구성된다. 이 메시지들로부터 NEC는 $\langle view, highqc.view, \Sigma \rangle$ 로 구성되며, 여기서 Σ 는 $(view, highqc.view)$ 필드에 대한 $f + 1$ 명의 검증자 집계 서명이다. 제안 p 가 NEC를 포함할 때, 이는 암호학적 서명 Σ 에 의해 $p.tc.high_tip$ 에 해당하는 제안이 정족수 증명서(QC)를 형성하기에 충분한 지지를 받지 못했음을 입증한다. 제안 p 는 동일한 뷰에서의 NEC를 포함해야 한다(즉, $p.view = p.nec.view$). NEC는 또한 신규 제안 p 가 연장해야 할 QC를 지시한다(즉, $p.block.qc.view = p.nec.highqc.view$).

타임아웃 메시지

타임아웃 메시지는 $\langle view, tip, tc, \sigma \rangle$ 로 정의되며, 검증자가 특정 뷰에서 타임아웃을 경험하거나 실패를 감지하면 생성되어 브로드캐스트된다. 검증자는 자신이 투표한 최신 신규 제안의 tip(위에서 논의)을 전송하며, $view, tip.view, tip.block.qc.view, tip.nec.view$ 등 모든 관련 뷰에 대한 서명 σ 를 함께 보낸다. 검증자가 이전 라운드($view - 1$)의 실패를 나타내는 타임아웃 증명서를 관측했다면, tc 는 해당 뷰에 대응하는 TC로 설정되고, 그렇지 않으면 $\perp$ 로 설정된다.

타임아웃 증명서(TC)

타임아웃 증명서(TC)는 동일한 뷰에서의 정족수($2f + 1$) 타임아웃 메시지에서 생성되며, 타임아웃 조건에 대한 광범위한 함의를 나타낸다.

$$TC = \langle view, tips, high_tip, \Sigma \rangle$$

여기서 $tips$ 는 각 타임아웃 메시지에서 얻은 tip들의 벡터이고, Σ 는 타임아웃 메시지들로부터의 집계 서명을 의미하며, $high_tip$ 은 $tips$ 에 대해 계산된다.

증명서 확장

서로 다른 블록을 가리키는 두 개의 정족수 증명서 qc_1 과 qc_2 를 고려하자. 이때 $qc_1.view = v_1$, $qc_2.view = v_2$ 이다. 만약 $v_1 < v_2$ 이고, $qc_2.proposal_id$ 가 나타내는 제안 p 가 qc_1 을 포함한다면 qc_1 은 qc_2 의 부모이다. 또한 p 는 qc_2 를 포함하는 모든 제안의 부모이다. 부모 제안(및 증명서)의 시퀀스 번호는 자식 제안(및 증명서)의 시퀀스 번호보다 1 작다. 우리는 자식 제안(및 증명서)이 부모 제안(및 증명서)을 확장한다고 말하며, 확장 관계는 추이적이다. 즉, qc_2 가 qc_1 을 확장하고 qc_3 가 qc_2 를 확장하면, qc_3 도 qc_1 을 확장한다.

상충하는 증명서

증명서 qc_1 과 qc_2 가 주어졌을 때, qc_1 이 qc_2 를 확장하지도 않고 qc_2 가 qc_1 을 확장하지도 않으면 qc_1 은 qc_2 와 상충한다. 유사하게, qc_1 과 qc_2 를 포함하는 제안들은 상충하는 제안이다.

3.2 합의 속성

프로토콜은 다음을 보장한다.

- **안전성(Safety):** 동일한 시퀀스 번호에 대해 두 올바른 검증자가 서로 다른 블록을 커밋하지 않는다.
- **생존성(Liveness):** GST 이후, 올바른 검증자의 모든 제안은 유한한 시간 내에 커밋된다.

AHL(Any Honest Leader) 속성은 매개변수 k 로 파라미터화되며, GST 이후 정직한 리더의 제안이 잠재적으로 비연속적인 k 명의 정직한 리더의 뷰 후에 커밋되어야 함을 의미한다. 이는 파이프라인 프로토콜의 일반적인 LSO(Leader-Speaks-Once) 동작 모드—각 리더가 즉시 다음 뷰로 회전되기 전에 단일 제안을 만들고 인증하는 모드—에 대해 BeeGees [17]에서 최초로 제안되었다.

본 연구에서는 AHL이 테일 포킹 방지에 충분하지만 지나치게 엄격하다는 점을 관찰한다. 따라서 우리는 테일 포킹 내성의 본질을 포착하기 위해 **No Tail-Forking (NTF)**라는 새로운 완화된 속성을 정의하고 보장한다.

- **No Tail-Forking (NTF):** 시퀀스 번호 s 에 대한 정직한 리더의 제안이 다수의 정직한 검증자에게 투표되었다면, 동일한 시퀀스 s 에서 다른 어떤 제안도 커밋될 수 없다.

모든 선행 파이프라인 프로토콜(BeeGees 제외)은 NTF를 만족하지 않으며(따라서 AHL도 만족하지 않는다).

Leader-Speaks-Once-for-Self(LSOS) 모델

BeeGees [17]에서 고려된 Leader-Speaks-Once(LSO) 모델에 더해, 우리는 LSOS라 불리는 파이프라인 프로토콜의 새로운 동작 모델을 고려한다. LSOS 모델에서는 리더가 회전되기 전에 단일 **신규 제안**을 만들고 인증해야 한다. 중요한 점은 LSOS 모델 하에서, 리더는 자신의 신규 제안을 제시하기 전에 이전 리더의 블록에 대한 재제안을 발행할 수 있다는 것이다. 이 LSOS 모델로 동작할 때, MonadBFT는 더 강한 AHL 속성을 달성한다(정리 5).

3.3 Protocol Setting

Protocol Phases

프로토콜에서, 각 라운드는 다른 파이프라인 합의 프로토콜과 유사하게 제안 검증과 새로운 제안 생성을 위해 필수적인 일련의 작업이다. 라운드는 검증자가 제안을 검증하고 자신의 투표를 던질 때 시작된다. 다음 리더는 이 투표들을 수집하고, 정족수 증명서(QC)를 형성하며, 이를 새로운 제안에 포함한다. 제안과 QC를 수신하면, 다음 라운드가 시작된다. 리더십이 각 라운드마다 회전하므로, *view*와 *round*라는 용어는 상호 교환적으로 사용된다.

리더가 자신의 역할을 수행하지 못할 때(또는 비동기 상태 동안) *failed view*가 발생하며, 이때 비정상 경로가 트리거되고, 프로토콜은 복구 메커니즘을 사용하여 새로운 리더와 함께 TC를 사용해 다음 view로 이동하고 제안 생성을 계속한다.

Client and Protocol Interaction

아래에서는 클라이언트-프로토콜 상호작용의 고전적 모델이 어떻게 동작하는지 설명한다.

Client Request

클라이언트 c 가 자신의 거래 T 가 처리되기를 원할 때, 모든 검증자에게 *Request* 메시지를 보낸다. 이 메시지에는 거래 T 와 클라이언트의 로컬 타임스탬프 또는 순차적 요청 번호가 포함된다. 검증자들이 오래되거나 중복된 요청을 걸러내기 위해 이러한 값들에 의존하므로, 클라이언트는 타임스탬프 값이 단조 증가하도록 보장해야 한다.

Client Response (Confirmation)

클라이언트는 검증자들로부터 T 에 관한 *Early-Confirmation* 메시지를 최소 $2f + 1$ 개 수신하면 자신의 거래 T 가 확정되었다고 간주한다. 충분한 *Early-Confirmation*이 없을 경우, 클라이언트는 최소 $f + 1$ 개의 전통적 *Confirmation* 메시지를 수신하면 거래를 확정한다.

만약 *Early-Confirmation* 메시지가 제안 p 에 대해 view v 에서 전송되었다면, 반드시 $v = p.view + 1$ 이어야 한다. 이러한 경우, 클라이언트는 $T \in p$ 를 확정하기 전에 동일한 view v 를 참조하는 *Early-Confirmation* 메시지 $2f + 1$ 개를 기다려야 한다. 그러나 전통적 확인은 모든 메시지가 동일한 view에서 기원하는 것을 요구하지 않는다. 필요한 확인들을 수집한 후, 클라이언트는 T 를 실행된 것으로 표시하고 거래의 결과를 수락한다.

4 MonadBFT

프로토콜은 두 가지 구별되는 운영 모드를 규정한다: 정상 경로와 비정상 경로. 정상 경로에서, 블록체인은 제안들이 순차적으로 체인에 추가되는 정상 동작 상태에 있으며, 따라서 블록체인의 성장을 촉진한다. 이 과정은 제안에 대한 투표 수집, 제안에 대한 정족수 증명서의 구성과 생성, 현재 리더에 의한 새로운 제안의 생성, 그리고 새로 생성된 블록 제안을 인준하기 위한 검증자들의 후속 투표를 포함한다. 정상 경로 동안 이루어진 모든 제안은 이전에 실패한 것들에 영향을 받지 않는 새로운 블록을 제시하므로 신규 제안임을 유의하라.

반대로, 비정상 경로는 실패가 감지될 때 시작된다. 타임아웃 기간 내에 검증자가 유효한 제안을 수신하지 못하면 실패가 감지된다. 실패가 감지되면(타임아웃 기간 내에 제안을 수신하지 못하는 경우), 검증자들은 인지된 실패의 지표로서 타임아웃 메시지를 생성한다. 그런 다음 이 메시지들은 타임아웃 증명서로 집계되어 올바른 검증자의 다수가 실패를 인식했음을 입증한다. 그 후, 프로토콜은 재제안될 수 있고 이후에 연장될 수 있는 가장 최근에 인증된 신규 제안을 식별한다. 그러나 재제안될 블록이 인증되지 않았다는 증거(NEC로 표시됨)가 있으면, 프로토콜은 대신 신규 제안으로 진행한다.

4.1 Happy path

리더의 역할에서, 검증자는 블록체인에 다음으로 추가될 블록을 위한 제안¹을 작성함으로써 합의 프로세스를 시작한다.

정상 경로 동안의 제안 p 는 직전 view로부터의 QC를 포함하며(즉, $p.view = p.block.qc.view + 1$), 어떠한 TC(Timeout Certificate)나 NEC(No-Endorsement Certificate)도 포함하지 않는다.

합의 프로토콜의 정상 경로 동안, 리더는 제안 메시지에 캡슐화된 블록을 제안함으로써 과정을 시작한다. 이후 후속 리더는 제안에 대해 검증자들로부터 투표를 수집하여 정족수 증명서(QC)를 구성한다. 이 QC는 다음 제안의 블록에 포함되며, 리더가 제안하고 있는 제안의 부모에 대한 참조 역할을 한다(알고리즘 2, 3행).

마찬가지로, 제안을 수신하면 각 검증자는 제안을 검증한다. 이 절차에는 리더의 서명 인증과, 제안이 직전 라운드의 제안을 직접 확장하면서 직전 라운드 제안의 정족수 증명서(QC)도 함께 보유함을 확인하는 것이 포함된다. 이 확장 검증은 수신된 제안에 대해 SAFETYCHECK 함수를 실행함으로써 달성된다(알고리즘 2, 38행)이며, 알고리즘 1, 13행에 개요가 제시되어 있다. 간결함을 위해 알고리즘 설명에서 서명과 해시 검증은 생략한다. 문맥상 명확할 때는 메시지의 서명도 생략한다.

¹ 앞서 언급했듯이, 정상 경로 동안의 모든 제안은 신규 제안이다.

Algorithm 1: Utilities for a validator

```
1 Func BUILDBLOCK(qc):
2   payload  $\leftarrow$  fill with mempool transactions;
3   payload_hash  $\leftarrow$  HASH(payload);
4   block_hash  $\leftarrow$  HASH(seq, payload_hash, qc);
5   return  $\langle$ qc.seq + 1, payload, payload_hash, qc, block_hash $\rangle$ ;

6 Func BUILDPROPOSAL(view, block, tc, nec):
7   proposal_id  $\leftarrow$  HASH(block.block_hash, view);
8    $\sigma \leftarrow$  SIGN(proposal_id);
9   return  $\langle$ view, proposal_id, block,  $\sigma$ , tc, nec $\rangle$ ;

10 Func SAFETYCHECK(p):
11   if p.block.seq  $\neq$  p.qc.seq + 1 then
12     return false;
13   if p.view = p.block.qc.view + 1 then
14     return true; // Happy path: p extends qc's proposal
15   if p.tc  $\neq \perp$  and p.view = p.tc.view + 1 then
16     if p.block.qc  $\neq$  p.tc.high_tip.block.qc or
17       p.tc.high_tip  $\neq$  Alg3.FINDHIGHTIP(p.tc) then
18       return false; // qc of p and high_tip must match
19     if p.nec  $\neq \perp$  then
20       return
21       p.view == p.nec.view  $\wedge$  p.block.qc.view == p.nec.highqc.view;
22     return p.block.block_hash == p.tc.high_tip.block.block_hash;
23     //Without nec, repropose the block of high_tip
24   return false;

25 Func UPDATETIP(p):
26   if p.view = p.block.qc.view + 1 or p.nec  $\neq \perp$  then
27     return tip of p;
28   return Alg3.FINDHIGHTIP(p.tc);

29 Func PARENTID(proposal):
30   return proposal.block.qc.proposal_id;
```

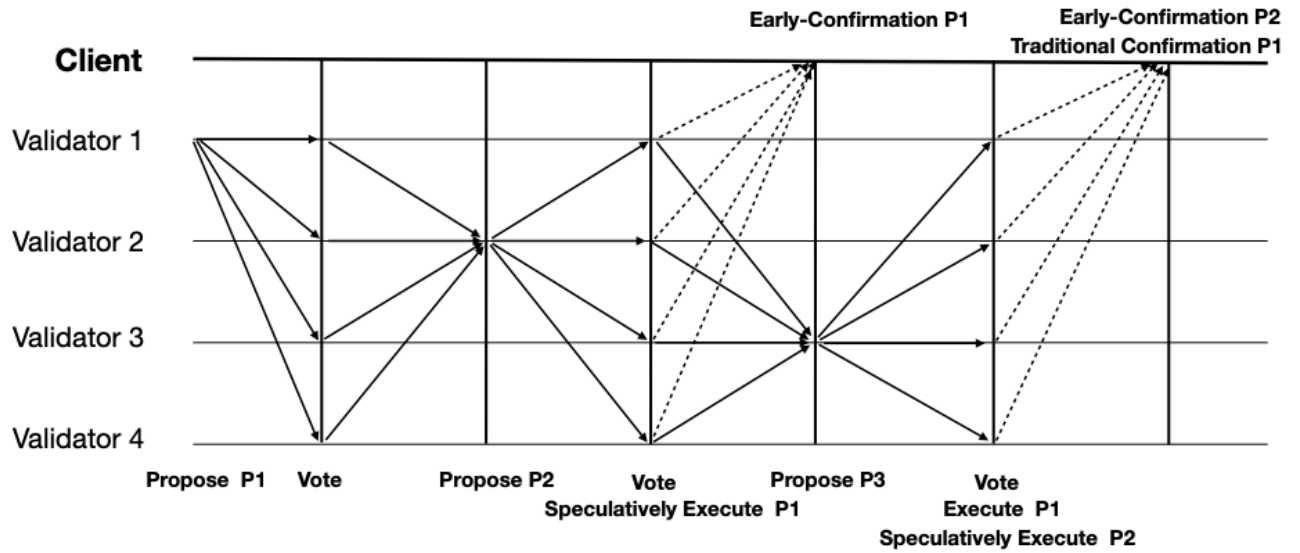


Fig. 2. Happy path workflow in MonadBFT

메시지 유형의 컨텍스트. 검증이 성공하면, 검증자는 부모 제안 내의 블록의 페이로드를 추측적으로 실행하고 이어서 자신의 투표 $\langle \text{view}, \text{seq}, \text{proposal_id}, \text{state_hash}, \sigma \rangle$ 를 다음 리더에게 전송한다. 각 투표에는 모든 필드에 대한 진정성을 보장하기 위해 검증자의 서명이 동반된다.

투표에 부모 제안의 추측 실행으로부터의 결과 상태 해시(머클 트리의 루트 해시)를 포함함으로써, 프로토콜은 검증자들이 추측 실행 이후 일관된 상태를 유지하도록 보장한다. 부모에 대한 정족수 증명서(QC)를 포함하는 제안을 수신하면, 검증자는 자신의 *view*를 증가시킨다. 부모와 조부모 제안이 연속된 *view*에서 온 경우, 검증자는 조부모 제안을 커밋한다. 그런 다음 전통적 *Confirmation* 메시지가 클라이언트에게 발행되어, 조부모 제안과 모든 미커밋 프리픽스 블록을 커밋한다(알고리즘 2, 41행). 현재 검증 중인 제안의 *view* 번호와 그 부모 제안의 *view* 번호가 연속적인 경우 ($p.\text{view} = p.\text{Block.QC.view} + 1$), 검증자는 클라이언트에게 *Early-Confirmation* 메시지를 보내어 부모 제안 내의 거래와 모든 미커밋 조상들을 인증한다(알고리즘 2, 46행). 투표를 수집하는 리더는 이를 정족수 증명서(QC)로 집계하고, 자신의 제안에 이를 포함하며, 이후 다음 제안을 제안한다. 그림 2는 정상 경로 동안의 MonadBFT의 워크플로를 보여준다.

정상 경로 동안, 검증자는 전통적인 2라운드 합의 메커니즘에서와 같이 정족수 증명서(QC)를 기준으로 잠그는 대신 자신이 투표한 유효한 제안에 대해 잠금을 설정한다. 이 잠금 메커니즘은 비정상 경로로부터의 복구 이후 다음 재제안이 잠금된 신규 제안의 블록을 포함하도록 보장한다. 이것은 검증자들이 잠금된 신규 제안이 다수의 정직한 검증자들로부터 투표를 받았는지 여부에 대한 지식이 없을 수 있으며, 그동안 후속 리더가 테일 포킹 전략을 시도할 수 있기 때문에 필수적이다. 정상 동작 하에서, 이 접근 방식은 추측 실행 또는 최종 확인을 실패하게 만들 수 있는 잠재적 결함을 최소화한다.

Algorithm 2: Consensus for Validator i

```
1  If Validator  $i$  is the leader
2  global variables: high_tip, last_tc, curView
3  upon Receipt of a vote  $v$  do
4     $qc \leftarrow \text{Alg4.HANDLEVOTE}(v)$ ; // Attempt to create  $qc$  from  $2f + 1$  votes
5    if  $qc = \perp$  then
6      break; // Not enough votes to form  $qc$ 
7     $\text{newblock} \leftarrow \text{Alg1.BUILDBLOCK}(qc)$ ; // Happy path, produce fresh proposal
8     $P \leftarrow \text{Alg1.BUILDPROPOSAL}(\text{curView}, \text{newblock}, \perp, \perp)$ ;
9    Broadcast  $P$ ;
10 upon Receipt of timeout message  $M$  do
11    $tc \leftarrow \text{Alg4.HANDLETIMEOUT}(M)$ ; // Attempt to create  $tc$  from  $2f + 1$  timeout messages
12   if  $tc = \perp$  then
13     break; // Not enough timeout messages to form  $tc$ 
14    $\text{last\_tc} \leftarrow tc$ ;  $\text{high\_tip} \leftarrow tc.\text{high\_tip}$ ;
15   if block and its payload (block.payload) for high_tip is present then
16      $P \leftarrow \text{Alg1.BUILDPROPOSAL}(\text{curView}, \text{block}, tc, \perp)$ ; // Reproposal
17     Broadcast  $P$ ;
18   else
19      $\text{Alg5.SENDRECOVERYREQUEST}(\text{curView}, \text{high\_tip}, tc)$ ; // Request block (c.f. Section 5)
20      $\text{recovering}^{\text{curView}} \leftarrow \text{true}$ ;
21 upon Receipt of block recovery response  $r$  and  $\text{recovering}^{\text{curView}}$  do
22    $\text{nec}, \text{block} \leftarrow \text{Alg5.LEADERRECOVERY}(r)$ ; //  $r$  is CR or NE (c.f. Section 5)
23   if  $\text{nec} \neq \perp$  then
24      $\text{newblock} \leftarrow \text{Alg1.BUILDBLOCK}(\text{high\_tip}.block.qc)$ ; // Fresh proposal
25      $P \leftarrow \text{Alg1.BUILDPROPOSAL}(\text{curView}, \text{newblock}, \text{last\_tc}, \text{nec})$ ;
26     Broadcast  $P$ ;
27   else if  $\text{block} \neq \perp$  then
28      $P \leftarrow \text{Alg1.BUILDPROPOSAL}(\text{curView}, \text{block}, \text{last\_tc}, \perp)$ ; // Reproposal
29     Broadcast  $P$ ;
30 All Validators
31 global variables: ledger, highest_voted_view, tip, curView
32 upon Receipt of recovery request  $R$  from leader  $L = \text{LEADER}(R.\text{view})$  do
33    $\text{Alg5.VALIDATORSYNC}(R)$ ;
34 upon Receipt of proposal  $P$  from leader  $L$  do
35    $\text{Alg4.INCREMENTVIEW}(P.tc)$ ;  $\text{Alg4.INCREMENTVIEW}(P.block.qc)$ 
36   if  $P.\text{view} \neq \text{curView}$  or  $L \neq \text{LEADER}(P.\text{view})$  then
37     break;
38   if  $\text{Alg1.SAFETYCHECK}(P)$  and  $P.\text{view} > \max(\text{highest\_voted\_view}, P.block.qc.\text{view})$  then
39      $\text{parent\_proposal} \leftarrow \text{get proposal PARENTID}(P)$ ;
40      $\text{grandparent\_proposal} \leftarrow \text{get proposal PARENTID}(\text{parent\_proposal})$ ;
41     if  $P.block.qc.\text{view} = \text{parent\_proposal}.block.qc.\text{view} + 1$  then
42       COMMIT( $\text{grandparent\_proposal}$ ); send Confirmation to clients;
43     if  $P.\text{view} = P.qc.\text{view} + 1$  then
44       If ledger conflicts with parent_proposal, revert to the common ancestor;
45        $\text{ledger} \leftarrow \text{EXECUTEBLOCK}(\text{parent\_proposal})$ ; // Speculate
46       Send Early-Confirmation of the parent_proposal to clients;
47       Send vote for  $P$  to the next leader;  $\text{highest\_voted\_view} \leftarrow P.\text{view}$ 
48        $\text{tip} \leftarrow \text{Alg1.UPDATE TIP}(P)$ ;
49 upon timeout event do
50   Broadcast Timeout message  $\langle \text{curView}, \text{tip}, \sigma \rangle$ ;
51    $\text{highest\_voted\_view} \leftarrow \max(\text{curView}, \text{highest\_voted\_view})$ ;
52 upon Receipt of timeout message  $M$  do
53    $\text{Alg4.HANDLETIMEOUT}(M)$ ;
```

Algorithm 3: Finding `high_tip` from a `tc` or set of timeout messages

```
1 Function FINDHIGHTIP(input):
2   high_tip  $\leftarrow \perp$ ;
3   if input is a TC then
4     | tips  $\leftarrow$  input.tips;
5   else
6     | tips  $\leftarrow$  [timeoutMsg.tip for timeoutMsg  $\in$  input];
7   for tip  $\in$  tips do
8     | if not VERIFYFRESHPROPOSAL(tip) then
9       | | return  $\perp$ ;
10  | return arg maxt $\in$ tips t.view;
11 Function VERIFYFRESHPROPOSAL(tip):
12  | if tip.view = tip.block.qc.view + 1 then
13    | return true;
14  | if tip.nec  $\neq \perp$  then
15    | return (tip.view == tip.nec.view)  $\wedge$  (tip.block.qc.view ==
16    |   tip.nec.highqc.view);
17  | return false;
```

신규 제안 P 에 대한 잠금은 “비정상 경로” 시나리오 동안 특정 조건들이 충족될 때까지 유지된다. 이러한 조건들은 P 에 대해 이중 제안이 없는 한, 최소 $f + 1$ 명의 올바른 검증자들에 의해 투표되지 않았다면에만 P 의 잠금이 상충 제안 P' 로 대체될 수 있도록 보장한다.

반면에, P 가 최소 $f + 1$ 명의 올바른 검증자들에 의해 투표되지 않았고, P 의 tip이 이후 TC의 `high_tip`인 경우, No-Endorsement 증명서(NEC)는 `high_tip`(P 에 대응하는)이 최소 $f + 1$ 명의 올바른 검증자들에 의해 투표되지 않았음을 입증하는 증거가 될 수 있다. 결과적으로, 잠금된 제안과 그 페이로드는 안전하게 교체될 수 있다. NEC가 없는 경우, 이후 리더는 `high_tip`의 블록을 재제안해야 하며, 이는 프로토콜의 **No Tail-Forking** 속성을 보장한다.

4.2 Unhappy path

리더가 유효한 블록을 제안하지 않거나, 리더의 제안이 최소 $2f + 1$ 명의 검증자들로부터 지지를 모으지 못하면, 프로토콜은 비정상 경로로 전환된다. 특히, 검증자들은 τ 라고 하는 타임아웃 기간 내에 유효한 제안을 수신하지 못하면 비정상 경로로 전환한다. 이 경우, 검증자는 그림 3에 표시된 바와 같은 타임아웃 메시지 $\langle \text{view}, \text{tip}, \text{tc}, \sigma \rangle$ 를 브로드캐스트한다. 다음 리더를 포함한 각 검증자는 타임아웃 메시지를 수신한다. 다음 리더는 그런 다음 서로 다른 $2f + 1$ 개의 타임아웃 메시지들로부터 타임아웃 증명서를 구축한다. 리더는 그런 다음 알고리즘 3을 사용하여 `high_tip`을 결정한다(모든 검증자들은 TC의 `high_tip`의 유효성을 검증하기 위해 동일한 알고리즘을 사용한다). `high_tip`을 찾는 목적은, 단일 검증자에 의해 인증된 (신규) 제안이 있거나 올바른 검증자들의 다수가 그 제안에 투표했다면, 반드시 그것이 복구되고 그 블록이 다시 제안되

어야 한다는 점이다. 이것은 다른 검증자들이 결국 동일한 높이에서 동일한 블록(페이로드)을 인증할 수 있게 하여, 어떤 테일 포킹 공격도 실패하게 만든다.

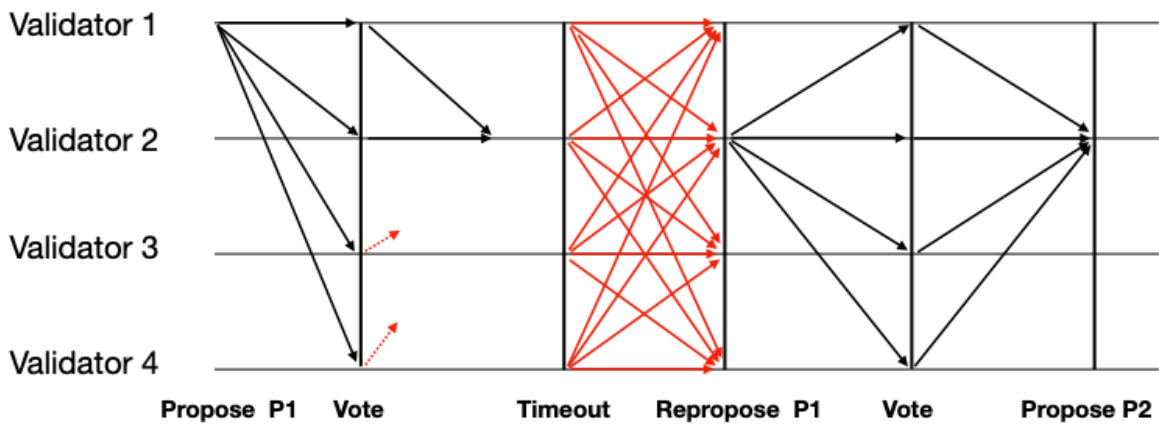


Fig. 3. Example of an unhappy path workflow in MonadBFT

선택된 *high_tip*은 그 블록이 리더에 의해 최초 제안되었을 때 최소 $2f + 1$ 명의 검증자들의 투표를 얻었을 수 있는 신규 제안을 나타낸다. 그러나 이러한 정족수에도 불구하고, 원래의 제안은 최종화된 체인에 포함되지 못했을 수 있다. 이 알고리즘은 실패 시나리오에서도 검증자들이 올바른 *high_tip*에 수렴하도록 보장하여, 테일 포킹을 방지한다.

알고리즘 3은 일련의 유효성 조건들에 기반하여 가장 적절한 *high_tip*이 선택되도록 보장하기 위해 구조화된 검사들을 통해 진행된다. 알고리즘 3(8행)에서, 초기 단계는 각 *tip*이 신규 제안들로부터 기원했다는 것을 검증하는 것이다. 추가로, *tip*이 NEC를 가지고 있다면, *tip*은 NEC에서의 것과 동일한 *view*를 가져야 하며(오래된 NEC가 사용되지 않도록), NEC가 제시하는 것과 동일한 QC를 확장해야 한다. 그런 다음 알고리즘은 후보 *tip*들 중에서 가장 높은 *view*를 가진 *tip*을 *high_tip*으로 선택한다. 만약 $high_tip.nec \neq \perp$ 라면, NEC 메시지는 신규 제안들의 이전 *tip*들이 충분한 지지를 검증자들로부터 받지 못했음을 입증하는 증거로 기능하며, 그 특정 *tip*(NEC를 담고 있는)을 새로운 *high_tip*으로 선택하는 것을 정당화한다.

*high_tip*이 식별되면, 프로토콜은 *high_tip*에 연관된 블록의 가용성에 따라 리더가 수행해야 하는 특정 동작들을 규정한다. 리더가 필요한 블록을 보유하고 있으면, 그것을 즉시 재제안하는 것이 그에게 부과된다(알고리즘 2, 15행과 28행). 블록이 즉시 가용하지 않은 경우, 리더는 네트워크에 블록을 요청해야 하며(알고리즘 2, 19행), 그 다음 그것의 재제안을 보장해야 한다(알고리즘 2, 28행). 효율적인 블록 검색 메커니즘은 5장에서 제시된다. 알고리즘 5는 메인 알고리즘 2에서 호출되는 블록 복구 방법들을 제공하며 블록 복구 메커니즘의 내부 메시지들도 처리한다.

비정상 경로 동안 후속 리더의 책임은 두 가지이다: *high_tip*의 블록을 재제안하여 합의를 지속하거나, 혹은 페이로드가 *high_tip*에 대해 $f + 1$ 명 이상의 올바른 검증자들에 의해 지지되지 않았음을(NEC를 통해) 논박의 여지 없이 입증하는 것이다. 이 조항은 상황이 필요할 때 리더가 대체 블록을 제안함으로써 진전을 이룰 수 있도록 허용하므로 중요하다.

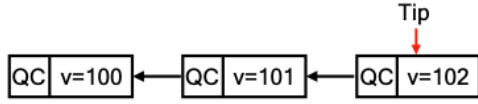


Fig. 4. The tip points to the latest proposal.

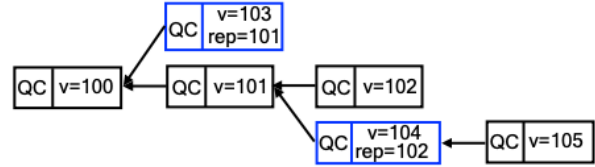


Fig. 5. Fresh proposals and repropoals.

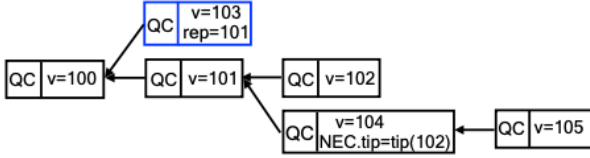


Fig. 6. NEC allowing a leader to make a fresh proposal

The figures illustrate the selection of the **high-tip** and the subsequent proposal processes during view failures. Blue rectangles indicate repropoals, while black rectangles indicate fresh proposals.

한편, 비정상 경로 이후 리더로부터 새로운 제안을 수신하는 검증자들은 그 제안이 수락하기에 안전한지 검증해야 한다(알고리즘 2의 38행에서 알고리즘 1의 SAFETYCHECK 호출에 의해). 비정상 경로에서 제안 p 에 대한 안전성 검사는 p 가 (이전 view로부터의) 유효한 TC와 (FINDHIGHTIP 알고리즘에 따라 계산된) 유효한 $high_tip$ 을 보유하는지 확인하는 것으로 시작된다. 그런 다음, NEC가 존재한다면, NEC는 (동일한 view에서의) 유효해야 하고 제안은 NEC가 제시한 것과 동일한 QC를 확장해야 한다($p.block.qc.view = p.nec.highqc.view$). 이러한 검사들이 성공하면, 검증자는 커밋 조건이 만족될 때 조부모 제안을 커밋한다(알고리즘 2, 41행). 그런 다음 검증자는 자신의 투표를 다음 리더에게 보내고(알고리즘 2, 47행과 48행), 자신의 view 번호와 tip을 갱신한다.

예로서, 그림 4는 한 검증자에 대한 tip을 보여준다. 그림 5는 다중 view 실패들을 보여준다: 먼저, view 102의 실패에서, view 101의 tip이 $high_tip$ 으로 선택되고 view 103 동안 재제안된다. view 102로부터의 제안은 충분한 수의 검증자들에게 전파되지 않았고, 그 결과 $tc.tips$ 에서 제외되었다. view 103의 실패 이후, view 102의 tip이 후속 TC의 일부로서 어떤 검증자에 의해 표면화되며, 이 tip이 $high_tip$ 으로 선택된다. ($high_tip$ 에 대한 후보로는 오직 신규 제안들만 고려되므로, view 103의 tip은 $high_tip$ 의 후보로 고려되지 않음을 주목하라) 그리고 그것은 이후 view 104에서 재제안으로 재제안된다.

그림 6은 view 104에 대한 대안 시나리오를 보여준다. view 103의 실패 이후, 리더는 $high_tip$ (view 102의 신규 제안에 연관된 tip)에 대응하는 블록을 보유하지 않으며, 따라서 네트워크에 그것을 요청한다. 올바른 검증자들은 view 102 동안 제안을 수신(투표)하지 않았다. 이후 view 104의 리더는 검증자들로부터 최소 $f + 1$ 개의 무지지 메시지들을 집계하여 $high_tip$ 에 대한 NEC를 구성하고, view 104에서 view 102로부터의 블록을 재제안으로 재제안하는 대신 신규 제안으로 진행할 수 있도록 한다. NEC는 최고로 관측된 QC의 view, view 101을 포함하며, 신규 제안은 그것을 확장해야 한다.

Equivocation Handling

이중 제안은 리더가 동일한 view 동안 동일한 시퀀스 번호에 대해 다수의 제안들

$p_{set} = \{p_1, p_2, \dots, p_k\} (k > 1)$ 을 제시할 때 발생한다. 어떤 이중 재제안들이든 `block_hash`의 차이들로 인해 올바른 검증자에 의해 단순히 거부된다(알고리즘 1, 20행). 신규 제안들의 경우, 만약 이러한 제안들 중 하나($p_i \in p_{set}$)가(`high_tip`을 통해) 최소 $f + 1$ 명의 올바른 검증자들에 의해 인증되면, 그것의 정족수 증명서(QC)는 후속 타임아웃 증명서(TC)에 나타날 것이다. 결과적으로, 다음 제안은 프로토콜의 규칙들에 따라 반드시 p_i 를 확장해야 한다. 그러나 p_i 를 인증하는 QC를 수신한 올바른 검증자들의 수가 $f + 1$ 명 이하이고, 이후 그것의 QC가 TC에 포함되지 않는다면, 후속 리더는 p_i 를 확장하지 않는 상충 제안 p'_i 를 제안할 수 있다. 이러한 시나리오들에서, 이전에 p_i 를 인증하고, p_i 를 추측적으로 실행했으며, p_i 에 대해 *Early-Confirmation*을 보냈던 검증자는 알고리즘 2, 44행에서 규정된 바에 따라 p'_i 를 수신하면 자신의 결정을 되돌릴 것이다.

중요하게도, 클라이언트들은 최소 $2f + 1$ 개의 검증자들로부터의 조기 확인을 수신한 후에만 거래를 커밋된 것으로 인식한다. 앞서 언급했듯이, *Early-Confirmations*는 동일한 view 동안 생성되어야 한다. 따라서, 만약 p_i 내의 거래가 되돌려지면, 그것은 클라이언트들에 의해 최종화된 것으로 간주되지 않을 것이다. 이 메커니즘은 이중 제안의 부정적 효과들에 대하여 프로토콜의 강인성을 뒷받침한다.

Leader Rotation During Re-proposal

정상 조건 하에서, 리더 회전은 각 라운드 후에 발생한다. 그러나, 라운드가 실패한 경우, 새로운 리더는 실패한 리더로부터의 블록을 재제안해야 한다.

이러한 시나리오들에서, 현재의 새로운 리더는 재제안에 대한 QC를 성공적으로 형성한 후 자신의 신규 제안을 제안할 기회를 부여받는다(다른 검증자들은 이전에 실패한 신규 제안의 블록을 재제안할 때 현재 리더에게 자신의 투표들을 향하게 해야 한다). 우리는 이러한 운영 모델을 LSOS(Leader-Speaks-Once-for-Self) 모델이라고 부른다.

이 접근법은 공정성의 한 층을 도입한다. 리더의 초기 역할은 실패한 신규 제안의 블록의 복구를 돕는 것이며, 그 다음 자신의 신규 제안을 제안할 수 있는 능력이 제공된다. 리더가 재제안하고 그 다음 신규 제안을 하도록 허용함으로써, 프로토콜은 검증자들 사이에서 공평한 리더 회전과 자원 분배를 촉진하며, 블록 구성으로부터의 MEV 보상 [2, 12]과 관련하여 합의의 강건성을 향상시킨다. MonadBFT에 의해 보장되는 NTF(No-Tail-Forking) 속성은 테일 포킹을 방지하면서 정직한 블록들을 추가함으로써 체인 성장을 보장한다. 추가로, 프로토콜은 커밋 조건(연속된 QC들)이 충족되면 체인의 미커밋 프리픽스를 커밋한다.

MonadBFT는 LSOS 모델 하에서 클라이언트들에 대해 AHL 속성을 보장한다(정리 5). 직관적으로, GST 이후에, 리더 L_1 이 올바르면, $2f + 1$ 명의 검증자들이 그 리더의 제안 P 에 투표할 것이다. 실패한 다음 view의 경우, MonadBFT의 No-Tail-Forking 속성 때문에, 다음 정직한 리더(L_2)는 P 를 재제안할 것이다. LSOS 체제에서, L_2 는 또한 P 를 가리키는 QC q 를 포함하는 자신의 신규 제안 P' 를 만들 기회를 갖는다. $2f + 1$ 명의 정직한 검증자들이 P' (그리고 P 의 블록에 대한 QC q)를 수신하면, 그들은 P 의 블록에 대해 *Early Confirmation*을 보내어, 클라이언트가 *Early-Confirmation*을 통해 P 의 블록을 최종화된 것으로 간주할 수 있게 한다.

이 메커니즘은 HotStuff-1 [24]에서 제안된 슬롯팅 접근법으로부터 벗어나며, 항상 테일 포킹을 방지하는 동시에 리더가 자신의 블록을 제안할 수 있도록 허용함으로써 뚜렷한 이점을 제공한다. MonadBFT가 재제안 이후 추가 슬롯을 부여하는 이 수정 없이도 테일 포킹 공격에 대해 강인함을 갖는다는 점에 유의하라(표 2 참조). 대조적으로, 간소화된 HotStuff-1 프로토콜은 슬롯팅을 통해 단지 테일 포킹을 지연시킬 뿐이며, 리더 회전 동안 취약한 상태로 남아 있다.

Model	NTF (No Tail-Forking)	AHL (Any Honest Leader)
LSO	Yes	No
LSOS	Yes	Yes (through early confirmation at clients)

Table 2. 서로 다른 리더 회전 모델 하에서의 MonadBFT의 속성들. LSO(Leader-Speak-Once)는 리더가 매 라운드 후에 회전됨을 수반한다. 우리는 LSO의 완화로서 LSOS(Leader-Speak-Once-for-Self)를 정의하는데, 이는 리더가 자신의 신규 제안을 만든 후에 회전됨을 수반하며(이전에 더 이른 리더의 블록의 재제안을 허용한다).

4.3 Pacemaker

페이스메이커는 뷰 전환을 조정하고 동기 네트워크 조건(부분 동기에서 GST 이후) 하에서 진행을 보장하는 책임이 있다. 뷰 v 에서 올바른 검증자의 페이스메이커는 모든 올바른 검증자들이 유한한 시간 내에 동일한 뷰 v 또는 더 높은 뷰 $v' > v$ 로 결국 동기화될 것을 보장한다. 올바른 검증자 $f + 1$ 명의 부분 집합이 결함 있는 검증자 f 명과 함께 나머지 f 명의 올바른 검증자들보다 먼저 뷰 $v' > v$ 로 진행할 가능성이 있지만, 페이스메이커는 모든 올바른 검증자가 뷰 v 로 전환하거나 이후에 뷰 v' 로 따라잡도록 보장한다(보조정리 10). 이 동기화는 페이스메이커(알고리즘 4, 2행)를 통해 정족수 증명서(알고리즘 2, 35행, [3–5]도 참조) 또는 타임아웃 증명서(알고리즘 2, 35행)를 사용함으로써 달성되어 생존성을 보장한다.

비정상 경로에서, MonadBFT는 DiemBFT [3]에서도 구현된 전략인 Bracha의 신뢰 가능한 브로드캐스트 메커니즘 [4], [5]을 채택한다(알고리즘 4 참조). 이 메커니즘은 전역 안정화 시간(GST) 이후 상수 시간 2Δ 내에 뷰 동기화를 달성하여, 검증자들이 각자의 *tip*을 브로드캐스트하는 동안 자신의 뷰들을 동기화하도록 보장한다. MonadBFT의 뚜렷한 특징은 동일 계열의 다른 프로토콜처럼 가장 높은 QC에만 의존하는 대신, 검증자들이 타임아웃 메시지 내에서 각자의 *tip* 정보를 공유한다는 점이다. 이 메커니즘은 올바른 *high_tip*의 선택과 검증을 모두 용이하게 한다.

검증자가 자신의 로컬 타임아웃 기간 내에 제안을 수신하지 못하면 타임아웃 메시지를 브로드캐스트한다. $2f + 1$ 개의 타임아웃 메시지를 수집하면, 검증자는 타임아웃 증명서(TC)를 구성하고 자신의 뷰를 증가시킨다(알고리즘 2, 35행). Bracha 기반 페이스메이커에서, 검증자는 세 가지 메커니즘을 통해 자신의 뷰를 증가시킨다:

- **제안 처리:** 제안 p 를 관측하면, 검증자는 자신의 뷰를 $p.qc.view + 1$ 로 증가시키거나; 만약 $p.tc \neq \perp$ 이면 $p.tc.view + 1$ 로 증가시킨다.
- **타임아웃 메시지 관측:** 검증자는 타임아웃 메시지 t 에 포함된 QC 또는 TC에 기반하여 자신의 뷰를 갱

신할 수도 있다. 즉 $\max(t.\text{tip.block.qc.view} + 1, t.\text{tc.view} + 1)$ 로 갱신한다(알고리즘 4, 14행).

- 더 나아가, 검증자가 아직 타임아웃되지 않았지만 $f + 1$ 개의 타임아웃 메시지를 관측하면 Bracha 메커니즘 [4]에 따라 자신의 타임아웃을 트리거하고, 아직 그렇지 않았다면 자신의 타임아웃 메시지를 브로드캐스트한다. $2f + 1$ 개의 타임아웃 메시지를 수신하면 검증자는 타임아웃 증명서 tc 를 구성하고 뷰 $tc.view + 1$ 로 진행한다. 검증자는 자신의 뷰보다 낮은 어떤 타임아웃 메시지도 거부한다.

5 Block Propagation and Recovery

블록 전파는 리더가 자신의 제안을 다른 모든 검증자들에게 보내는 메커니즘이다. 블록 복구도 리더가 아직 수신하지 못한 블록을 재제안하기 위해 시작되며, 리더가 블록을 복구하여 이를 재제안하거나, 리더가 신규 블록을 제안하는 데 사용되는 무지지 증명서(NEC)로 귀결된다.

전파를 위해, 단순한 접근법은 리더가 전체 블록 제안을 브로드캐스트하도록 하는 것이다. 유사하게, 단순한 복구 접근법은 리더가 모든 검증자들에게 블록 제안에 대한 요청을 브로드캐스트하고, 블록을 가진 모든 검증자들이 직접 응답하도록 하는 것이다. 그러나 블록 전파와 복구 모두에서, 리더의 통신 오버헤드는 검증자 수에 선형적으로 스케일한다. 검증자 집합이 커짐에 따라, 이 오버헤드는 리더에서 병목을 만들어 빠르게 감당할 수 없게 된다. 효율적인 블록 제안 및 복구 메커니즘은 다음의 속성들을 달성해야 한다:

- 노드당 통신 부하가 상수일 것.
- 지연이 일관될 것.
- 비잔틴 결함 허용일 것.

이 목표들을 달성하기 위해, MonadBFT는 블록 전파와 복구를 위해 검증 가능한 정보 분산(Verifiable Information Dispersal, VID) [8,15]의 변형을 사용하며, 이는 자체적으로 삭제 부호(erasue coding) 메커니즘을 사용한다.

이 절에서, 먼저 삭제 부호를 간략히 소개하고 그것이 블록 전파에 어떻게 사용되는지를 보인 다음, 블록 복구 메커니즘을 설명한다. 삭제 부호를 사용하는 블록 전파 메커니즘은 표준적이지만, 우리는 복구 실패를 증명할 수 있는 새로운 복구 메커니즘을 도입하며, 이는 테일 포킹 내성을 달성하는 데 결정적이다.

5.1 Block Propagation

블록 전파를 위해 삭제 부호를 활용하면 리더의 전송 병목을 줄인다. 직접 제안은 한 단계에서 완료되지만, 리더가 인코딩된 청크를 검증자들에게 보내고 각 검증자가 자신의 청크를 재브로드캐스트하는 접근법은 중계 단계를 추가한다. 이는 전파 지연을 증가시키지만, 리더의 전송 시간을 크게 줄인다. 예를 들어, 1Gbit/s 링크에서 100명의 검증자에게 2MB 블록을 보내는 것은 1.6초가 소요되지만, 삭제 부호화된 청크를 분산하면 전송 지연이 50ms로 줄어든다.

Algorithm 4: Pacemaker for View Synchronization

```
1 Global Variables: curView, local timer per view;
2 Function INCREMENTVIEW(cert):
3   if cert.view  $\geq$  curView then
4     |   newView  $\leftarrow$  cert.view + 1;           // Increment view to next
5     |   STARTTIMER(newView);
6 Function STARTTIMER(newView):
7   Stop local timer for curView;
8   curView  $\leftarrow$  newView;
9   Start local timer for curView;
10 Function HANDLETIMEOUT(timeoutMsg):
11   v  $\leftarrow$  timeoutMsg.view;
12   if v < curView then
13     |   return  $\perp$ ;
14   INCREMENTVIEW(timeoutMsg.tip.block.qc);
15   INCREMENTVIEW(timeoutMsg.tc);
16   timeoutMsgsv  $\leftarrow$  timeoutMsgsv  $\cup$  {timeoutMsg}; // Accumulate by view
17   if |timeoutMsgsv|  $\geq 2f+1$  then
18     |   tc  $\leftarrow$  Create Timeout Certificate from timeoutMsgsv;
19     |   INCREMENTVIEW(tc);
20     |   return tc;
21   if |timeoutMsgsv|  $\geq f+1$  and broadcastTimeoutv  $\neq$  true then
22     |   trigger timeout event; // Bracha Mechanism: Broadcast timeout
23     |   broadcastTimeoutv  $\leftarrow$  true; // Ensure only one timeout broadcast
24   return  $\perp$ ;
25 Function HANDLEVOTE(vote):
26   v  $\leftarrow$  vote.view;
27   id  $\leftarrow$  vote.proposal_id;
28   if v < curView then
29     |   return  $\perp$ ;
30   votesid  $\leftarrow$  votesid  $\cup$  {vote}; // Accumulate votes by id
31   if |votesid|  $\geq 2f+1$  then
32     |   qc  $\leftarrow$  Create Quorum Certificate from votesid;
33     |   INCREMENTVIEW(qc);
34     |   return qc;
35   return  $\perp$ ;
```

삭제 부호에서, 블록 제안과 같은 메시지는 j 개의 부호화 청크 $C_1, C_2, \dots, C_j$ 로 분할되며, 그중 임의의 k ($\leq j$)개의 청크 부분집합으로 원본 메시지를 재구성할 수 있다. 우리의 설정에서는 검증자당 하나의 청크가 있으므로 $j = n = 3f + 1$ 이다. 통신 오버헤드와 결함 허용의 균형을 위해, 우리는 k 를 $f + 1$ 로 설정한다. 이는 최대 $f + 1$ 명의 올바른 검증자만 유효한 청크를 제공하면 되도록 보장하면서 각 검증자가 보내야 하는 데이터의 양을 최소화한다. MonadBFT에서 리더는 제안²을 인코딩하고, 부호화된 청크를 생성하여 제안하며, 추가 오버헤드로 머클 트리의 서명된 루트 형태의 암호학적 증명을 포함한다. 우리는 청크 C_i 를 튜플 $C_i = (i, v, data_i, \pi_i)$ 로 형식적으로 정의하는데, 여기서 i 는 왼쪽에서 오른쪽으로 정렬된 머클

트리를 구성하는 리프 노드들의 시퀀스 내에서 청크의 인덱스를 나타낸다. 값 v 는 뷰 식별자로, 청크가 제안되는 합의 뷰를 지정한다. 구성 요소 $data_i$ 는 삭제 부호화된 페이로드의 i -번째 청크이다. 마지막으로, π_i 는 청크의 페이로드 $data_i$ 를 머클 트리의 (서명된) 루트에 연결하여 그 진정성과 무결성을 증명하는 머클 증명이다. 이 메커니즘은 리더가 청크 C_i 를 해당 검증자 i 에게 배포할 수 있게 하며, 검증자는 그 청크가 인코딩된 제안의 일부이며 뷰 v 에 대한 제안 디코딩에 사용할 수 있음을 검증할 수 있다. C_i 를 검증한 후, 각 검증자는 청크를 브로드캐스트하여, 모든 검증자들이 제안을 재구성하기에 충분한 청크를 집합적으로 모으도록 보장한다.

청크를 검증하는 것은 뷰와 머클 루트 해시에 대한 리더의 서명을 검증하고, 머클 증명을 검증하는 것을 포함한다. 이는 청크의 진정성과 올바른 합의 라운드와의 연관성을 보장한다. 물론, 제안을 생성하고 청크를 배포한 리더가 비잔틴이라면, 디코딩 과정이 실패할 수 있다. 그러나 올바른 리더의 경우, VID는 아래에 설명된 복구 메커니즘뿐 아니라 블록 전파 메커니즘도 제안 전체(블록 필드를 포함하여)의 복구를 가능하게 함을 보장한다. 우리의 설계는 검증자들이 *디코딩 이전에* 각 청크를 인증하고 검증할 수 있게 하여³, 검증 과정을 최적화한다. 사전 디코딩 검증은 비잔틴 결함 시나리오에서 특히 유리한데, (i) 비잔틴 리더가 디코딩 실패를 유발하는 불일치 청크를 생성할 때 이를 입증 가능하게 만들고, (ii) 올바른 리더가 생성한 청크의 조작은 디코딩의 계산 오버헤드 없이 머클 증명 검증 실패를 기반으로 즉시 탐지될 수 있기 때문이다.

5.2 Block Recovery

블록 복구 메커니즘은 리더가 *high_tip* 제안의 블록을 보유하지 않을 때 시작되며, 리더는 새로운 (재)제안을 제안하기 전에 이를 검색하려고 시도해야 한다. 블록이 사용 가능하지 않은 경우, 리더는 어떤 올바른 검증자도 *high_tip*의 제안을 인증하지 않았음을 입증하는 증거를 제공해야 한다.

앞서 언급했듯이, 효율적인 복구 메커니즘은 삭제 부호를 사용한다 [7,27]. 블록체인에서 효율적이고 검증 가능한 데이터 분산을 위한 삭제 부호는 이전에도 사용되었지만 [19,27,33], 여기서 제시하는 블록 복구 메커니즘은 페이로드 또는 데이터가 제대로 전파되지 않았음을 증명하는 증명서 중 하나의 획득을 보장한다는 점에서 기존 기법을 넘어선다. 이 증명서는 새로운 리더에게 이전에 블록을 담고 있던 제안을 인증하기 위한 QC가 형성되지 않았음을 검증 가능한 증거로 제공하여, 리더가 신규 제안을 제안할 수 있게 한다.

블록 복구 메커니즘의 의사코드는 알고리즘 5에 제시되어 있다. 코딩 메커니즘 자체는 블랙박스로 취급된다. 페이로드 복구 프로세스는 다음 단계를 따른다:

² 단순화를 위해, 우리는 전체 제안을 인코딩하는데, 전체 제안을 복구하는 오버헤드가 무시할 만하기 때문이다.

³ 각 청크의 진정성은 디코딩 이전 [1] 또는 디코딩 이후 [23]에 검증될 수 있다.

1. 요청 단계(알고리즘 2의 19행에서 알고리즘 5의 SENDRECOVERYREQUEST 호출):

리더는 서명된 요청 $R = \langle v, high_tip, tc, \sigma \rangle$ 을 뷰 v 와 *high_tip.proposal_id* 필드에 대해 서명하여 모든 검증자에게 브로드캐스트한다.

2. **검증자 동기화 단계**(알고리즘 2의 33행에서 알고리즘 5의 VALIDATORSYNC 호출):

검증자들은 $v = tc.view + 1$ 이고 $FINDHIGHTIP(tc) = high_tip$ 인지 확인함으로써 $high_tip$ 의 유효성을 검증한다.⁴

검증자는 아직 v 로 진행하지 않았다면 $INCREMENTVIEW(R.tc)$ 를 호출하여 자신의 뷰를 v 로 증가시킨다.

- 제안의 블록을 보유한 검증자들은 모든 i 에 대해 i 번째 검증자에게 뷰 번호에 대해 서명된 *Chunk Relay* 메시지 $CR_i = \langle v, C_i, \sigma \rangle$ 를 전송한다.
- 제안의 블록을 보유하지 않은 검증자들은 뷰 v 와 $high_tip.proposal_id$ 필드에 대해 서명된 *No Data* 메시지 $\langle v, high_tip.proposal_id, tc, \sigma \rangle$ 를 브로드캐스트한다.

3. **청크 수신 단계**(알고리즘 5의 19행):

- 각 검증자는 $2f + 1$ 명의 검증자로부터의 응답을 수집하며, 자신의 *Chunk Relay* 또는 *No Data* 메시지 중 하나를 수신한다.

4. **검증자에서 리더로의 통신**(계속):

- 검증자 i 는 청크를 보유하고 있으면 자신의 서명이 포함된 *Chunk Relay* CR_i 를 리더에게 보내거나, *No-Endorsement (NE)* 메시지 $\langle view, high_tip.qc.view, \sigma \rangle$ 를 보낸다. *No-Endorsement* 메시지는 두 필드 모두에 대해 서명된다.

5. **리더 복구**(알고리즘 2의 21행에서 알고리즘 5의 LEADERRECOVERY 호출):

- 리더가 검증자들로부터 *유효한 암호학적 증명을 담은* 서로 다른 $f + 1$ 개의 청크를 수집하면, 블록을 재구성하려 시도한다. 원래 대상 블록을 제안한 리더가 올바르게 행동했다면, 이 청크들은 새로운 리더가 페이로드를 신뢰성 있게 복구하도록 해준다.
- 대신 리더가 $f + 1$ 개의 *No-Endorsement* 메시지를 수신하면, 이는 해당 블록이 어떤 올바른 검증자에 의해서도 인증되지 않았음을 입증하는 증거가 되어, 리더가 대체 블록을 제안할 수 있게 한다.

6. **디코딩 실패**(알고리즘 5의 44행과 29행):

- 리더가 $f + 1$ 개의 부호화 청크를 최소 포함하는 $2f + 1$ 개의 메시지를 수집했음에도 블록 디코딩에 실패한 경우, 디코딩 실패를 증명하는 메시지를 브로드캐스트한다:

$$\langle v, S \rangle, \quad \text{where } S = \{C_i \mid C_i = (i, v, data_i, \pi_i)\}, \quad |S| \geq f + 1.$$

여기서 S 는 리더가 수집했으나 디코딩에 실패한 최소 $f + 1$ 개의 부호화 청크 집합이다. 이 증명은 뷰와 S 내 각 C_i 의 해시를 연결(concatenate)한 해시에 대한 리더의 서명으로 이루어진다.

S 를 모든 검증자에게 직접 브로드캐스트하는 대신, 리더는 두 번째 층의 인코딩을 적용하여 크기가 $3f + 1$ 인 더 큰 집합 S' 를 생성할 수 있으며, S' 로부터 임의의 $f + 1$ 개의 청크로 S 를 재구성할 수 있다. 그런 다음 리더는 제안 전파에 이미 사용된 동일한 삭제 부호와 머클 증명 메커니즘을 활용하여, S' 로부터의 인코딩된 하나의 청크 C'_j 를 각 검증자에게 배포한다. 검증자가 S' 로부터 최소 $f + 1$ 개의 유효한 청크를 수집하면, S 를 재구성하고 원래의 디코딩 프로세스를 진행한다.

- 디코딩 실패 증명 $\langle v, S \rangle$ 를 수신하면, 각 검증자는 다음의 검사를 수행한다:

- (a) 각 $C_i \in S$ 의 진정성과 무결성을 커밋(예: 머클 루트 또는 다항식 커밋)에 대해 검증하고 $|S| \geq f + 1$ 임을 확인한다.
- (b) S 에 있는 청크를 사용하여 블록 디코딩을 시도한다.
- (c) 디코딩이 실패하고 증명이 유효하면, 검증자는 이전과 같이 리더에게 NE 메시지로 응답한다. 그렇지 않으면, 검증자는 단순히 타임아웃한다.

디코딩은 또한 복구된 제안이 SAFETYCHECK를 위반하거나 블록 해시 검증이 실패할 때와 같이, 유효하지 않은 블록(제안)의 복구로 이어질 수 있다. 전반적으로, 안전성 점검에서의 어떤 실패든 제안을 무효로 만들며 디코딩 실패로 간주된다.

합의 메커니즘에서와 마찬가지로, 각 검증자는 진행을 보장하기 위해 블록 복구에 대한 타임아웃 값을 유지한다. 실제로, 블록 복구 메커니즘은 어떤 합의 프로토콜에서도 필수 구성 요소이지만, 종종 프로토콜 설명에서 간과된다. 전통적으로, 합의 프로토콜은 인증된 블록의 복구에 초점을 맞추지만, MonadBFT는 테일 포킹 보호를 위해 인증되지 않은 블록의 복구도 허용한다. 특히 실패 이후 리더에 의해 시작될 때, 블록 복구는 추가 지연 비용을 도입한다는 점에 유의하라.

이제 블록 복구 메커니즘과 관련된 두 가지 유용한 보조정리를 제시한다.

⁴ 지분증명에서, 잘못된 $high_tip$ 에 대한 증거는 블록 복구 메커니즘의 허위 트리거를 억제하기 위해 리더를 처벌(“슬래시”)하는 데 사용될 수 있다.

Lemma 1. 이중 제안이 아니고 올바르게 인코딩되었으며⁵ 신규 제안 p 에 대한 정족수 증명서(QC) 또는 $2f + 1$ 개의 투표가 주어지면, 블록 복구 메커니즘을 사용하여 p 의 블록을 복구할 수 있다.

Proof. p 가 $2f + 1$ 명의 검증자에 의해 투표되었으므로, 이들 중 최소 $f + 1$ 개의 투표는 올바른 검증자에게서 온 것이며, 그들은 또한 p 의 블록을 보유한다. 블록 복구 메커니즘에서, 검증자 동기화 단계 동안, $f + 1$ 명의 올바른 검증자 각각은 청크 C_i 를 검증자 i 에게 보낸다. 표준적인 정족수 교차 논증에 의해 검증자 i 는 $2f + 1$ 개의 응답을 수신하므로 자신의 청크 C_i 를 수신하는 것이 보장된다. 그 후 리더는 $2f + 1$ 개의 메시지를 수신하며, 이 중 최소 $f + 1 (\geq k)$ 개는 올바른 검증자에게서 온 것이다. 그러면 리더는 임의의 k 개의 청크를 사용하여 블록을 복구한다.

Lemma 2. 이중 제안이 아니고 올바르게 인코딩된 신규 제안에 대한 정족수 증명서(QC) 또는 $2f + 1$ 개의 투표가 존재한다면, 리더가 해당 제안의 블록에 대해 유효한 무지지 증명서(NEC)를 구성하는 것은 불가능하다.

Proof. Lemma 1과 유사하게, 최소 $f + 1$ 명의 올바른 검증자가 신규 제안의 블록을 보유하므로, 그들은 청크 수신 단계에서 모든 올바른 검증자들이 자신의 청크를 받도록 보장하고, 이어서 리더에게 자신의 청크를 보낸다(NE 메시지가 아니라).

비잔틴 검증자는 최대 f 명이므로, 리더는 유효한 NE 메시지 $f + 1$ 개를 수신하지 못하며, 따라서 유효한 NE 증명서를 형성할 수 없다.

⁵ 이중 제안에서 나온 청크나 잘못 인코딩된 청크는 디코딩 실패를 초래할 수 있다.

```

1 Global Variables: LatestRequest, highqc_view, CRv, chunksv, NESetv,v',
  curView;
2 Function SENDRECOVERYREQUEST( $v, high\_tip, tc$ ):
3   Leader broadcasts a request  $R = \langle v, high\_tip, tc, \sigma \rangle$ ;
4    $highqc\_view \leftarrow R.high\_tip.qc.view$ ; LatestRequest  $\leftarrow R$ ;
5 Function CHUNKMATCHESREQUEST( $R, CR$ ):
6    $\text{return } CR.view = R.view \text{ and } CR.C.view = R.high\_tip.view$ ;
7 Function VALIDATORSYNC( $R$ ):
8   if  $R.view \neq R.tc.view + 1$  or  $FINDHIGHTIP(R.tc) \neq$ 
     $R.high\_tip$  or  $R.view < curView$  then
9      $\text{return}$ 
10  INCREMENTVIEW( $R.tc$ ); LatestRequest  $\leftarrow R$ ;
     $highqc\_view \leftarrow R.high\_tip.qc.view$ ;
11  if block for  $R.high\_tip$  then
12    Send CR message  $\langle R.view, C_i \rangle$  to each validator  $i$ ;
13  else
14    Broadcast No Data message  $\langle R.view, R.high\_tip.proposal\_id, R.tc, \sigma \rangle$ ;
15  for  $CR \in CRSet^{R.view}$  do
16    if  $CHUNKMATCHESREQUEST(R, CR)$  then
17      Send CR to the leader if have not sent any message (CR or NE) for
        view  $R.view$  already;
18      break;
19 upon Receipt of input for  $curView \leq input.view$  by validator  $i$  during chunk
    reception phase do
20   if input is Chunk Relay CR then
21      $CRSet^{CR.view} \leftarrow CRSet^{CR.view} \cup CR$ 
22     if  $CHUNKMATCHESREQUEST(LatestRequest, input)$  then
23       Send CR (replacing its signature) to the leader if have not sent any
        message (CR or NE) for the view  $CR.view$ ;
24   else if input is No Data message for proposal id  $id$  and view  $v$  then
25      $nodataMsgSet^{id,v} \leftarrow nodataMsgSet^{id,v} \cup input$ ;
26     if  $|nodataMsgSet^{id,v}| \geq 2f+1$  then
27        $highqc\_view \leftarrow \max(highqc\_view,$ 
         $input.tc.high\_tip.block.qc.view)$ ;
28       Send NE message  $\langle v, highqc\_view, \sigma \rangle$  to the leader if have not sent
        a CR or an NE message for view  $v$ ;
29 upon decoding failure proof  $\langle v, chunks \rangle$  do
30   if  $|chunks| < f + 1$  or any chunk in  $chunks$  is invalid then
31     trigger timeout event;
32   else
33      $block, error \leftarrow \text{reconstruct block from } chunks$ ;
34     if  $error \neq \perp$  or block is invalid then
35       Send NE message  $\langle v, highqc\_view, \sigma \rangle$  to the leader;
36     else
37       trigger timeout event;

```

Algorithm 5: Block Recovery (continued)

```
38 Function LEADERRECOVERY(input):
39   if input is Chunk Relay and CHUNKMATCHESREQUEST(LatestRequest,
      input) then
40      $\text{chunks}^v \leftarrow \text{chunks}^v \cup \{\text{input}\};$ 
41     if  $|\text{chunks}^v| \geq f+1$  then
42        $\text{block}, \text{error} \leftarrow \text{reconstruct block from } \text{chunks}^v;$ 
43       if  $\text{error} \neq \perp$  or block is invalid then
44         Broadcast decoding failure proof  $\langle v, \text{chunks}^v \rangle;$ 
45       else
46         return  $\perp, \text{block};$ 
47   else if input is an NE message from view  $v$  then
48      $\text{NESet}^{v, \text{highqc-view}} \leftarrow \text{NESet}^{v, \text{highqc-view}} \cup \{\text{input}\};$ 
49     if  $|\text{NESet}^{v, \text{highqc-view}}| \geq f+1$  then
50        $\text{nec} \leftarrow \text{Aggregate signatures in } \text{NESet}^{v, \text{highqc-view}} \text{ into a}$ 
        No-Endorsement Certificate;
51       return  $\text{nec}, \perp;$ 
52   return  $\perp, \perp;$ 
```

6 Proofs of Correctness

정족수 증명서(qc)는, qc 에 캡슐화된 내용 $\langle \text{view}, \text{seq}, \text{proposal_id}, \text{state_hash} \rangle$ 에 관한 서명 검증이 성립하면 유효한 것으로 간주된다.

Lemma 3. 주어진 임의의 뷰 v 에서, 두 개의 구별되는 유효한 정족수 증명서 qc_1 과 qc_2 가 존재하는 것은 불가능하다. 다시 말해, $qc_1.view = qc_2.view$ 이면 $qc_1.proposal_id = qc_2.proposal_id$ (그리고 $qc_1.seq = qc_2.seq$)가 성립한다.

Proof. 모순을 위해, 동일한 뷰에서 두 개의 구별되는 정족수 증명서가 형성되었다고 가정한다(각 증명서는 $2f + 1$ 명의 검증자들의 투표로 구성된다). 정족수 교차 논증에 의해, 최소 한 명의 올바른 검증자가 동일한 뷰에서 두 개의 구별되는 블록에 투표한 것이 된다. 그러나 MonadBFT 프로토콜(알고리즘 2)은 단일 뷰 내에서 어떠한 검증자도 한 번 이상 투표하는 것을 명시적으로 금지한다. 이 규칙에서, 검증자는 투표 후(알고리즘 2, 47행)와 타임아웃 후(알고리즘 2, 51행) 자신의 가장 높은 투표 뷰를 증가시키며, 자신의 가장 높은 투표 뷰보다 낮은 뷰를 가진 이후의 어떤 블록에 대한 투표도 거부한다(알고리즘 2, 38행).

Lemma 4. 만약 L 이 뷰 v 에서 신규 제안 p 를 제안하고, L 이 이중 제안을 하지 않으며, $2f + 1$ 명의 검증자가 p 에 투표했다면, 어떤 올바른 검증자도 뷰 $v' (\geq v)$ 에서 상충 제안 p' (즉, p' 는 $p.block$ 을 포함하지도 확장하지도 않음)에 투표하지 않을 것이다.

Proof. 우리는 뷰 번호 v' 에 대한 귀납법으로 이 명제를 증명한다. 먼저 Lemma 4를 직접 함의하는 더 강한 귀납 가설 $X(v')$ 를 정식화하고, 상호 귀납을 위해 두 개의 다른 귀납 가설 $Y(v')$ 와 $Z(v')$ 를 정의하며, $v' \geq v$ 에 대해 $X(v'), Y(v'), Z(v')$ 가 성립함을 증명한다. 편의를 위해, p 의 tip을 t^p 로 표기한다.

만약 L 이 뷰 v 에서 시퀀스 번호 s 를 가진 신규 제안 p 를 제안하고, L 이 이중 제안을 하지 않으며, $2f + 1$ 명의 검증자가 p 에 투표했다면:

- $X(v')$: 뷰 v' 에서의 임의의 유효한 제안(즉, 안전성 점검(알고리즘 1)을 통과한) p' 는 p . *block*을 확장($p'.seq > s$)하거나, p . *block*을 포함($p'.seq = s$)한다.
- $Y(v')$: 뷰 v' 에서의 임의의 TC의 *high_tip*은 t^p 이거나 p . *block*을 확장한다.
- $Z(v')$: 뷰 v' 에서의 임의의 TC의 *high_tip*은 t^p 이거나 뷰 $> v$ 의 신규 제안이다.

Base Case

- $X(v' = v)$: 뷰 v 에서 만들어진 유일한 제안이 p 이므로 자명하다.
- $Y(v' = v)$: 올바른 다수의 검증자들이 신규 제안 p 에 투표했으므로, 그들은 자신의 로컬 tip으로 t^p 를 가진다. 따라서 표준적인 정족수 교차 논증에 의해 t^p 는 TC에 반드시 등장한다. t^p 는(L 의 이중 제안이 없으므로) 가능한 최고 뷰 (v)를 가진 유일한 tip이므로, *high_tip*은 t^p 와 같다(알고리즘 3, 10행).
- $Z(v' = v)$: Y 의 기저 사례와 동일하다.

Induction Step

- $Z(v' + 1)$:
올바른 다수의 검증자 집합 S 가 뷰 v 에서 자신의 tip으로 t^p 를 가진다. S 의 검증자는 (알고리즘 1의 UPDATETIP 메서드를 통해) 뷰 $> v$ 에서의 새로운 신규 제안을 받거나, 뷰 $\in [v, v']$ 에서의 TC의 *high_tip*을 통해 자신의 tip을 갱신하는데, 귀납 가설 Z 에 의해 이들 역시 다시 t^p 이거나 뷰 $> v$ 의 신규 제안의 tip이다.
표준적인 정족수 교차에 의해, 뷰 $v' + 1$ 에서의 TC T 의 *tips*는 S 의 검증자 중 한 명의 tip을 포함해야 한다. 따라서 TC T 의 *high_tip*은 최소 뷰 v 를 가지며, 그러므로 t^p 이거나 뷰 $> v$ 의 신규 제안의 tip이다.
- $Y(v' + 1)$:
뷰 $v' + 1$ 에서의 TC와 그 *high_tip* t 를 고려하자. *high_tip*의 뷰를 a 라고 하자($v' + 1 \geq a$). 귀납 가설 $Z(v' + 1)$ 에 의해, t 는 t^p 이거나 뷰 $a > v$ 의 신규 제안의 tip이다.
 t 가 t^p 가 아닌 비자명한 경우, 신규 제안 tip t 는 두 가지 유형일 수 있다:
 - **정상 경로:**
만약 $t.view = t.block.qc.view + 1$ 이면, t 의 QC는 (뷰 $a - 1$ 의) 유효한 제안을 가리키며, 이는 상호 귀납 가설 $X(b)$ ($v' \geq b = a - 1 \geq v$)에 의해 p . *block*을 포함하거나 확장한다. p . *block*을 확장하는 블록의 QC를 가지므로, t 자신도 p . *block*을 확장한다.
 - **비정상 경로:**
high_tip t 는 동일한 뷰 a 에서 구성된 NEC(무지지 증명서)를 가진다(알고리즘 3, 15행). 이 NEC는 뷰 $a - 1$ 의 TC T^* 에 응답하여 생성되었고, 그 *high_tip* t^*

($= T^*.high_tip$)에 포함된 QC q^* 의 뷰 번호 v^* 를 포함한다. 귀납 가설 $Y(b)$ ($v' \geq b = a - 1 \geq v$)에 의해, t^* 는 t^p 이거나 $p.block$ 을 확장한다.

NEC가 요청 t^* 에 응답하여 생성되었으므로, t^* 는 p 일 수 없다(Lemma 2). 따라서 t^* 는 그 QC q^* 를 통해 $p.block$ 을 확장해야 한다. t 는 동일한 뷰 v^* 를 가진 QC q 를 포함하므로(알고리즘 3, 15행), Lemma 3에 의해 $q.proposal_id = q^*.proposal_id$ 가 성립하며, 이는 q 또한 $p.block$ 을 확장함을 의미한다. t 가 $p.block$ 을 확장하는 블록의 QC를 가지므로, t 자신도 $p.block$ 을 확장한다.

- $X(v' + 1)$:

만약 뷰 $v' + 1$ 에서의 p' 가 뷰 v' 의 QC를 포함한다면, 그것은 $p.block$ 을 확장한다(귀납 가설 $X(v')$ 에 의해).

그렇지 않다면, p' 는 뷰 v' 의 TC를 포함한다. 이제 $p'.TC$ 의 TC의 $high_tip$ t 를 살펴본다. 상호 귀납 가설 $Y(v')$ 에 의해, TC의 $high_tip$ t 는 p 이거나 $p.block$ 을 확장한다. 전자의 경우, $high_tip$ p 에 대해 NEC를 구성할 수 없으므로(Lemma 2), p' 는 p 의 블록을 재제안해야 한다. 후자의 경우, p' 는 t 와 동일한 QC를 포함하므로(알고리즘 1, 16행), p' 또한 $p.block$ 을 확장한다.

Corollary 1. 정직한 리더에 의해 GST 이후 원래 제안된 블록 b 에 대한 추측(조기 확인)은 되돌려지지 않는다.

Proof. b 가 뷰 v 에서 신규 제안 p 의 일부로 정직한 리더에 의해 원래 제안되었다고 하자. b 에 대한 추측이 되돌려지는 경우는 (뷰 $> v$)에서 b 를 확장하지 않는 QC를 수신했을 때뿐이다. Lemma 4는 그러한 QC가 존재할 수 없음을 보여준다.

Lemma 5. 만약 $f + 1$ 명의 올바른 검증자가 뷰 $v + 1$ 에서 뷰 v 의 제안 p 에 대한 QC를 수신했다면, 뷰 $v'(> v)$ 에서의 임의의 유효한 제안은 p 를 확장해야 한다.

Proof. 우리는 뷰 번호 v' 에 대한 귀납법으로 증명한다. 상호 귀납을 위해 귀납 가설 $X(v')$, $Y(v')$ 및 $Z(v')$ 를 정의하고, $v' \geq v + 1$ 에 대해 $X(v')$, $Y(v')$, $Z(v')$ 가 성립함을 증명한다.

만약 $f + 1$ 명의 올바른 검증자가 뷰 $v + 1$ 에서 뷰 v 의 제안 p 에 대한 QC를 수신했다면:

- $X(v')$: 뷰 v' 에서의 임의의 유효한 제안은 p 를 확장해야 한다.
- $Y(v')$: 뷰 v' 에서의 임의의 TC의 $high_tip$ 은 p 를 확장한다.
- $Z(v')$: 뷰 v' 에서의 임의의 TC의 $high_tip$ 은 뷰 $\geq v + 1$ 의 신규 제안이다.

Base Case

$X(v' = v + 1)$: 뷰 v 에서 p 에 대한 QC가 존재함을 알고 있다. 따라서 표준적인 정족수 교차 논증에 의해 뷰 v 에서 TC는 존재할 수 없다. 그러므로 뷰 $v + 1$ 에서의 어떤 유효한 제안도 뷰 v 의 QC를 포함해야 하며(알고리즘 1, 13행), 이는 Lemma 3에 의해 p 를 확장해야 한다.

$Z(v' = v + 1)$: $f + 1$ 명의 검증자들은 자신의 tip을 p 로 설정한다. 표준적인 정족수 교차 논증에 의해, 이 tip들 중 하나 t 는 뷰 $v + 1$ 의 TC에 포함되어야 한다. 따라서 $high_tip$ 은 뷰 $v + 1$ 에서 온 것이다.

$Y(v' = v + 1)$: Z 의 기저 사례에서 보였듯이, $high_tip$ 은 뷰 $v + 1$ 에서 온 것이다. 뷰 v 는 성공적이므로(뷰 v 에서 TC가 존재하지 않음), 뷰 $v + 1$ 에서 NEC는 생성되지 않는다. 따라서 선택된 $high_tip$ 은 $v + 1$ 의 신규 제안이어야 하며, 뷰 v 의 QC를 포함해야 한다. Lemma 3에 의해, 이 QC 또한 p 를 가리킨다.

Induction Step

$Z(v' + 1)$: $f + 1$ 명의 올바른 다수 검증자 집합 S 는 뷰 $v + 1$ 에서 자신의 tip을 뷰 $v + 1$ 로 갱신한다. S 의 검증자들은 자신의 tip을(알고리즘 1의 UPDATETIP 메서드를 통해) 뷰 번호 $\geq v + 1$ 로만 갱신하는데, 이는 뷰 $> v + 1$ 에서 새로운 신규 제안을 수신하거나, 뷰 $\in [v + 1, v']$ 의 TC의 $high_tip$ 을 통해서이며, 귀납 가설 Z 에 의해 이들 역시 다시 뷰 $\geq v + 1$ 에서 온 것이다. 표준적인 정족수 교차에 의해, 뷰 $v' + 1$ 의 TC T 의 tip들은 S 의 검증자 중 한 명의 tip을 포함해야 한다. 따라서 TC T 의 $high_tip$ 은 뷰 $\geq v + 1$ 에서 온 것이다.

$Y(v' + 1)$: 뷰 $v' + 1$ 에서의 TC와 그 $high_tip$ t 를 고려하자. $high_tip$ 의 뷰를 a 라 하자($v' + 1 \geq a$). 귀납 가설 $Z(v' + 1)$ 에 의해, t 는 뷰 $a \geq v + 1$ 의 신규 제안이다. 신규 제안 t 는 두 가지 유형일 수 있다:

- **정상 경로:** 만약 $t.view = t.block.qc.view + 1$ 이면, t 의 QC는 (뷰 $a - 1$ 의) 유효한 제안을 가리키며, 이는 상호 귀납 가설 $X(a - 1)$ 에 의해 p 를 확장하거나, $a = v + 1$ 일 때는 p 와 동일하다(Lemma 3).
- **비정상 경로:** $high_tip$ t 는 동일한 뷰 a 에서 구성된 NEC를 가진다(알고리즘 3, 15행). 이 NEC는 뷰 $a - 1$ 의 TC T^* 에 응답하여 생성되었고, 그 $high_tip$ $t^*(= T^*.high_tip)$ 에 포함된 QC q^* 의 뷰 번호 v^* 를 포함한다. 귀납 가설 $Y(b)$ (여기서 $v' \geq b = a - 1 \geq v$, $Y(v)$ 는 뷰 v 에서 TC가 존재하지 않으므로 자명하게 참)에 의해, t^* 는 그 QC q^* 를 통해 p 를 확장한다. t 는 동일한 뷰 v^* 를 가진 QC q 를 포함하므로(알고리즘 3, 15행), Lemma 3에 의해 $q.proposal_id = q^*.proposal_id$ 가 성립하며, 이는 q 또한 p 를 확장함을 의미한다.

두 경우 모두에서, t 는 p 를 확장하는 블록의 QC를 가지므로, t 자체가 p 를 확장한다.

$X(v' + 1)$: 만약 p' 가 뷰 $v' + 1$ 에서 뷰 v' 의 QC를 포함한다면, 그것은 p 를 확장한다(귀납 가설 $X(v')$ 에 의해). 그렇지 않다면, p' 는 뷰 v' 의 TC를 포함한다. 우리는 이제 $p'.TC$ 의 TC의 $high_tip$ t 를 살펴본다. 상호 귀납 가설 $Y(v')$ 에 의해, TC의 $high_tip$ t 는 p 를 확장한다. p' 는 t 와 동일한 QC를 포함하므로(알고리즘 1, 16행), p' 또한 $p.block$ 을 확장한다.

Theorem 1 (No Tail-Forking)

L 이 뷰 v 에서 시퀀스 번호 s 를 가진 신규 제안 p 를 제안하고, L 이 이중 제안을 하지 않으며, $2f + 1$ 명의 검증자가 p 에 투표했다면, 시퀀스 번호 s 에서 $p.block$ 이외의 어떤 블록도 커밋되지 않는다.

Proof. 모순을 위해, 블록 $b^*(\neq p.block)$ 가 동일한 시퀀스 번호 s 에서 커밋되었다고 가정하자(즉, 둘 중 어느 것도 다른 것을 확장하지 않음). b^* 는 제안 p^* 의 일부로 커밋되었다고 하자($p^*.view = v^*$) 그리고 p^* 는 b^* 를 포함하거나 확장한다. Lemma 4 덕분에, $v^* > v$ 는 불가능함을 안다.

이제 $v^* < v$ 인 경우를 분석하자, 즉 p 가 제안되기 전에 p^* 가 이미 커밋되었다. p^* 가 커밋되려면, $2f + 1$ 명의 검증자가 뷰 $v^* + 1$ 에서 p^* 에 대한 QC를 수신했어야 한다. Lemma 5에 의해, (p 를 포함한) 임의의 유효한 제안은 p^* 를 확장해야 하므로, 따라서 b^* 를 확장해야 하며, 이는 모순을 준다.

Theorem 2 (Safety of Early-Confirmation)

클라이언트가 조기 확인을 통해 거래 $t \in p.\text{block.payload}$ 를 최종화된 것으로 간주했다면, t 와 이를 캡슐화하는 블록 $p.\text{block}$ 은 결국 모든 올바른 검증자들에 의해 커밋된다.

Proof. 클라이언트는 동일한 뷰 v 에서 온 $2f + 1$ 개의 *Early-Confirmation* 메시지를 수신하면 조기 확인을 통해 자신의 거래를 최종화된 것으로 간주한다. 이 *Early-Confirmation* 메시지들 중 최소 $f + 1$ 개는 정직한 검증자에게서 왔어야 하며, 이들은 뷰 v 에서 뷰 $v - 1$ 의 제안 p' 에 대한 QC를 수신했는데, 여기서 p' 는 $p.\text{block}$ 을 확장한다. Lemma 5에 의해, 더 높은 뷰에서의 임의의 유효한 제안은 p' 를 확장해야 하므로, 추이적으로 $p.\text{block}$ 을 확장한다. MonadBFT의 생존성 속성과 결합하면, $p.\text{block}$ 은 결국 모든 올바른 검증자들에 의해 커밋될 것이다.

Theorem 3 (Safety)

동일한 시퀀스 번호 s 를 가진 두 개의 제안 p 와 p^* 가 두 올바른 검증자에 의해 커밋되는 일은 없다.*

Proof. 반증을 위해, 두 개의 제안 p 와 p^* 가 동일한 시퀀스 s 에서 커밋되었다고 가정하자. 그러면 이들은 상충하며, 이는 p 도 p^* 도 서로를 확장하지 않음을 의미한다. 일반성을 잃지 않고, $p.\text{view} < p^*.\text{view}$ 라고 가정하자.

올바른 검증자는 제안 p' 에 대한 QC를 포함하는 제안 p'' 를 수신한 후 p 를 커밋하는데, 여기서 $p''.\text{qc.proposal_id} = p'.\text{proposal_id}$ 이고 $(p'.\text{qc.proposal_id} = p.\text{proposal_id} \wedge p'.\text{view} = p'.\text{qc.view} + 1)$ 가 성립한다. 올바른 검증자에 의한 $p''.\text{qc}$ 의 수신은 최소 $f + 1$ 명의 올바른 검증자가 이미 p' 를 수신했고 따라서 뷰 $p.\text{view} + 1$ 에서 p 에 대한 QC($p'.\text{qc}$)를 수신했음을 의미한다. Lemma 5에 의해, p^* 는 p 를 확장해야 하므로, 모순에 도달한다.

다음으로, 우리는 부분 동기 네트워크 모델에서 MonadBFT의 생존성 보장을 확립한다. 전역 안정화 시간 (GST)이 존재하여 시간 t 에 전송된 모든 메시지는 시간 $\max(t, \text{GST}) + \Delta$ 까지 전달된다. 우리는 적절히 구성된 뷰 타이머 길이 τ 가 존재한다고 가정하며, 이는 페이스메이커와 함께 GST 이후 올바른 검증자들이 동일한 뷰에서 동기화되도록 보장한다. 이 가정은 본질적으로, 리더가 알려진 가장 높은 증명서를 식별하고, 블록을 제안하고, 검증자들이 투표할 충분한 시간을 부여하고, 투표와 타임아웃 메시지를 처리하기에 뷰 타이머가 충분히 길다는 것을 의미한다. 우리는 GST 이후 최초로 동기화된 뷰를 v_s 로 표시한다. 단순화를 위해, 타이밍 경계 계산 시 제안 전파와 디코딩 실패 증명은 5장에서 설명한 최적화된 릴레이 기반 2단계 전달 메커니즘이 아닌 순진한 단일 단계 전달을 사용한다고 가정한다. 삭제 부호를 활용한 최적화된 2단계 전달 메커니즘을 고려하면 추가적인 Δ 를 고려해야 한다. 그러나 최적화된 2단계 전달은 두 번의 단방향 메시지를 포함함에도, (제안 또는 증명 전파에 대한) 총 지연 2Δ 가 실제로는 순진한 1단계 전달 메커니즘의 단일 Δ 보다 작아 더 효율적이다.

Lemma 6. 뷰 v 에 있는 올바른 검증자 i 는 다음 조건들 하에서 뷰 v 로 전환한다.

- i 가 뷰 v 에 대한 잘 형성된 제안을 수신할 때,
- i 가 뷰 v 에 대한 타임아웃 메시지를 수신할 때, 또는
- i 가 뷰 $v - 1$ 에 대한(TC를 구성하기 위한) $2f + 1$ 개의 타임아웃 메시지 집합을 수신할 때,
- i 가 $R.tc.view = v - 1$ 을 만족하는 블록 복구 요청 메시지 R 을 수신할 때.

Proof. 올바른 검증자는 세 가지 주요 메커니즘을 통해 뷰 v 로 전환한다:

- **올바른 리더:**
뷰 v 보다 작은 뷰에 있는 검증자는 $p.qc.view + 1 = p.view = v$ 를 만족하는 제안 p 를 수신하면 뷰 v 로 전환한다(알고리즘 2, 35행).
- **리더 실패:**
검증자는 다음 조건들 중 하나에서 뷰 v 로 전환한다:
 - (i) $p.tc.view = v - 1$ 을 만족하는 타임아웃 증명서 tc 를 가진 제안 p 를 수신할 때(알고리즘 2, 35행).
 - (ii) 뷰 $v - 1$ 에 대한 다른 검증자들의 타임아웃 메시지 $2f + 1$ 개를 수신할 때(알고리즘 4, 19행; [4, 5]도 참조).
 - (iii) $t.high_tip.block.qc.view = v - 1$ (알고리즘 4, 14행) 또는 $t.tc.view = v - 1$ (알고리즘 4, 15행)을 만족하는 타임아웃 메시지 t 를 수신할 때.
- **블록 복구:**
검증자는 뷰 v 에 있고 $R.tc.view = v - 1$ 을 만족하는 요청 메시지 R 을 수신하면 뷰 v 로 전환한다(알고리즘 5, 10행 참조).

Lemma 7. GST 이후의 뷰 v 에 대해 L 이 지정된 리더라고 하자. 모든 올바른 검증자들이 L 을 리더로 합의한다면, 최대 2Δ 내에 모든 올바른 검증자들은 동기화되어 뷰 v 로 전환한다.

Proof. 뷰 v 에 최초로 진입하는 올바른 검증자를 고려하자. 리더 L 이 뷰 v 에 최초로 진입하면, 제안 p 를 브로드캐스트한다. 이 제안은 Δ 이내에 모든 올바른 검증자에게 전파되어, 여전히 더 이른 뷰에 있는 모든 검증자들이 v 로 전환할 수 있게 한다. 뷰 v 에 최초로 진입하는 검증자가 L 이 아니라면, 그 검증자는 뷰 $v - 1$ 에 대한 TC를 생성함으로써 그렇게 한다. TC를 구성하려면 $2f + 1$ 개의 타임아웃 메시지가 필요하며, 그중 최소 $f + 1$ 개는 올바른 검증자에게서 온 것이다. 이 올바른 검증자들은 뷰 $v - 1$ 에 대한 자신의 타임아웃 메시지를 브로드캐스트하며, 이는 Δ 이내에 다른 모든 올바른 검증자들에게 전파된다.

Lemma 6에 의해, 이러한 타임아웃 메시지들은 $v - 1$ 보다 이른 뷰에 있는 모든 올바른 검증자들이 $v - 1$ 로 진전하도록 보장한다. Bracha 타임아웃 메커니즘은 뷰 $v - 1$ 에 있는 임의의 검증자가 뷰 $v - 1$ 에 대한 $f + 1$ 개의 타임아웃 메시지를 수신하자마자 즉시 타임아웃하도록 보장한다(알고리즘 4, 22행). 2Δ 이내에, 뷰 $v - 1$ 에 있는 모든 올바른 검증자들은 TC를 형성하기 위한 $2f + 1$ 개의 타임아웃 메시지를 모으고, 그들을 뷰 v 로 증가시킬 수 있다(알고리즘 4).

따라서 2Δ 이내에, 모든 올바른 검증자들이 뷰 v 에 진입하고 동기화된다.

Lemma 8. 뷰 $v \geq v_s$ 에서, 올바른 리더 X 가 또 다른 올바른 리더 Y 가 생성한 블록 B 에 대해 블록 복구를 시작하고, B 가 최소 $f + 1$ 명의 올바른 검증자에게 전파되었다면, 리더 X 는 3Δ 이내에 B 를 복구할 수 있고, 모든 올바른 검증자들은 4Δ 이내에 X 의 재제안을 수신한다.

Proof.

X 에 의해 요청 메시지 $R = \langle v, high_tip, tc, \sigma \rangle$ 가 브로드캐스트되면, 올바른 검증자는 최대 Δ 이내에 R 를 수신한다. 뷰 $< v$ 에 있는 모든 올바른 검증자는 $IncrementView(R.tc)$ 를 호출하여 뷰 v 로 전환한다(Lemma 6).

이후, 각 올바른 검증자는 자신의 피어들에게 응답하며, 이는 검증자 i 를 위한 청크(C_i)이거나 No Data 메시지이다. 각 올바른 검증자 i 는 추가 Δ 이내에 다른 검증자들로부터 그러한 메시지 $2f + 1$ 개를 수신한다. 최소 $f + 1$ 명의 올바른 검증자들이 B 를 수신했고 해당 청크를 각 검증자에게 전달하므로, 표준적인 정족수 교차 논증에 의해 모든 올바른 검증자 (i)는 자신의 청크(C_i)를 수신한다.

그 다음 각 올바른 검증자 i 는 자신의 청크 C_i 를 리더 X 에게 보내는 데 최대 또 다른 Δ 가 필요하며, X 는 이어서(청크들이 올바른 리더 Y 에게서 왔으므로) 오류 없이 B 를 재구성한다. 총 시간은 3Δ 가 된다. 마지막으로, X 의 재제안은 추가 Δ 이내에 모든 올바른 검증자들에게 전파된다.

Lemma 9. 뷰 $v \geq v_s$ 에서, 블록 복구는 진행을 보장한다: 올바른 리더는 3Δ 이내에 유효한 (재)제안을 생성하거나 5Δ 이내에 무지지 증명서(NEC)(신규 제안을 위한)를 생성한다.

Proof.

Lemma 8과 유사하게, 블록 복구를 시작한 리더는 3Δ 이내에 청크와 NE 메시지로 구성된 $2f + 1$ 개의 응답을 수신한다. 이 응답들 중, $f + 1$ 개는 NE 메시지이거나 청크이다.

전자의 경우, 리더는 $f + 1$ 개의 NE 메시지에서 NEC를 생성할 수 있다. 후자의 경우, 청크들이 블록을 복구하기 위해 성공적으로 디코딩되거나, 리더는 블록 복구의 “디코딩 실패” 단계에 들어간다. Δ 이내에 모든 올바른 검증자들은 리더로부터 디코딩 실패 증명을 수신하고, NE 메시지로 응답한다. 리더는 또 다른 Δ 이내에 이러한 NE 메시지들을 수신하며, 총 5Δ 를 넘지 않는 시간 내에 NEC를 생성할 수 있다.

Lemma 10. 타임아웃 간격 τ 와 메시지 전파 지연이 유한하다면, 모든 올바른(정직한) 검증자들은 계속해서 더 높은 뷰로 전진한다.

우리는 이 보조정리를 두 부분으로 증명한다:

1. 먼저, 올바른 검증자 A 가 뷰 v 에 있고 다른 올바른 검증자 B 가 뷰 $v' > v$ 에 있다면, A 는 결국 자신의 뷰를 전진한다는 것을 증명한다: B 는 반드시 뷰 $\geq v$ 의 TC 또는 QC를 관측함으로써 뷰 v' 에 진입했어야 한다. B 가 뷰 $\geq v'$ 에서 언젠가 타임아웃한다면, B 는 자신의 타임아웃 메시지에서 뷰 $\geq v$ 의 TC 또는 QC를 A 에게 보내어, A 가 자신의 뷰를 전진하도록 한다(Lemma 6). 반대로, B 가 결코 타임아웃하지 않는다면, B 는 뷰 $\geq v'$ 의 모든 뷰에서 지속적으로 제안들을 관측해야 한다. 이러한 제안들 중 하나는(MonadBFT의 라운드 로빈 스케줄 때문으로) 정직한 리더에게서 온 것이어야 하며, 이는 A 에게도 제안을 보내어 A 가 자신의 라운드를 전진하도록 한다(Lemma 6).
2. 다음으로, 모든 올바른 검증자들이 동일한 뷰 v 에 있다면, 최소 한 명의 올바른 검증자가 뷰 $v + 1$ 로 전진한다는 것을 증명한다: 이는 타임아웃 간격이 유한하다는 사실로부터 직접적으로 따른다. 올바른 검증자들이 달리 자신의 라운드를 전진시키지 않는다면, 결국 모두가 타임아웃하여, 모든 이들이 TC를

생성하고 자신의 뷰를 전진시키도록 허용한다.

Lemma 11. *MonadBFT의 라운드 로빈 스케줄은 임의의 연속된 $3f + 3$ 명의 리더 시퀀스에서, 세 명의 연속된 올바른 리더의 그룹이 존재하도록 보장한다.*

Proof.

연속된 $3f + 3$ 명의 리더 시퀀스에는 세 명이 연속된 리더 그룹이 $3f + 1$ 개 포함된다. 라운드 로빈 스케줄로 인해, $3f + 1$ 명의 각 검증자는 $3f + 1$ 개의 그룹 중 정확히 세 그룹에 등장한다. 올바른 검증자는 $2f + 1$ 명이므로, 비둘기집 원리에 의해 이들 그룹 중 하나는

$$\left\lceil \frac{3 \times (2f + 1)}{3f + 1} \right\rceil = 3$$

명의 올바른 리더를 포함해야 한다.

Theorem 4 (Strong Liveness)

GST 이후, MonadBFT의 올바른 리더들은 유효한 제안을 생성하며, 올바른 리더의 임의의 제안은 결국 커밋된다.

Proof.

모든 올바른 검증자들은 더 높은 뷰로 계속 전진한다(Lemma 10). GST 이후, 블록 복구 메커니즘이 진행을 보장하므로(Lemma 9), 모든 올바른 리더는 유효한 제안을 생성할 수 있다.

모든 올바른 검증자들은 2Δ 이내에 리더의 뷰로 동기화되고(Lemma 7), 타임아웃 간격 τ 가 충분히 길기 때문에, GST 이후 모든 올바른 검증자는 올바른 리더의 제안에 투표한다. Lemma 4와 결합하면, GST 이후 올바른 리더의 제안은 반드시 후속 올바른 리더들에 의해 확장된다.

세 명의 연속된 올바른 리더 그룹이 무한히 자주 발생하므로(Lemma 11), GST 이후 모든 올바른 리더의 제안은 결국 커밋된다.

Theorem 5

MonadBFT는 LSOS 모델 하에서 AHL 속성을 만족한다. GST 이후, 올바른 리더 L 이 블록 b 를 담은 제안 p 를 제안했다면, (연속적이지 않을 수도 있는) 단일 후속 올바른 리더만으로도 프로토콜을 클라이언트 측에서 블록 b 를 최종화하는 방향으로 이끌기에 충분하다.

Proof.

이 정리를 증명하기 위해, 우리는 상호 배타적이고 완전한 두 경우를 분석한다:

1. 올바른 리더가 제안한 블록이 연속된 뷰들에서 또 다른 올바른 리더에 의해 확장되는 경우, 그리고
2. 올바른 리더가 제안한 블록이 비연속적인 올바른 리더의 신규 제안에 의해 확장되는 경우.

GST 이후, 연속된 리더들이 올바른 경우, 뷰 v 에서의 올바른 리더 L_v 가 p_v 를 제안한다. 이어서, 뷰 $v + 1$ 에서의 또 다른 올바른 리더 L_{v+1} 가 부모 제안 p_v 를 확장하는 p_{v+1} 를 제안한다. p_{v+1} 을 수신하면, 각

올바른 검증자는 p_v 에 대해 *Early-Confirmation* 메시지를 보낸다. L_v 와 L_{v+1} 가 모두 올바르므로, 클라이언트는 뷰 $v + 1$ 에서 $p_v.block$ 에 대한 $2f + 1$ *Early-Confirmation* 메시지를 수신하여 $p_v.block$ 을 커밋한다.

이제 다른 경우를 고려하자. 뷰들의 시퀀스 $V_{set} = \{v + 1, v + 2, v + 3, \dots, v + k\}$ 가 실패한다고 하자(여기서 $k \geq 1$). v 이후 처음으로 리더가 올바른 뷰를 $v + k + 1$ 이라고 하고, 그 리더를 L_{v+k+1} 로 표기하자. Lemma 4 및 생존성 속성에 의해, L_{v+k+1} 는 제안 p_{v+k+1} 을 만들며, 이는 반드시 $p_v.block$ 을 재제안해야 하므로, 즉 $p_{v+k+1}.block = p_v.block$ 이다.

더 나아가, LSOS 모델 하에서, MonadBFT는 뷰 $v + k + 1$ 이 재제안인 경우 뷰 $v + k + 1$ 과 $v + k + 2$ 의 리더를 동일하게 유지한다. 따라서 뷰 $v + k + 2$ 의 리더 또한 올바르다. 결과적으로, 뷰 $v + k + 2$ 의 리더는 p_{v+k+2} 를 만들며, 이는 생존성 속성 덕분에 p_{v+k+1} 을 확장한다.

첫 번째 경우와 유사하게, p_{v+k+2} 를 수신하면, 최소 $2f + 1$ 명의 올바른 검증자들이 p_{v+k+1} 에 대해 *Early-Confirmation* 메시지를 보내며, 이는 클라이언트가 $p_{v+k+1}.block = p_v.block$ 을 커밋하도록 이끈다. 따라서 MonadBFT는 LSOS 모델 하에서 클라이언트 측에서 AHL 속성을 보장한다.

아래에서는 다음 정의의 도움을 받아 일부 추가적인 낙관적 시간 경계를 제공한다.

Definition 1 (Global Direct Commit)

뷰 v 에서 제안된 제안 p_v 는, 뷰 $v + 1$ 에서 제안 p_{v+1} 에 대해 적어도 $f + 1$ 명의 올바른 검증자가 투표하고, 그때 p_{v+1} 이 p_v 를 인증하는 경우(즉, $p_{v+1}.block.qc.proposal_id = p_v.proposal_id$), *Global Direct Commit*을 달성했다고 말한다.

제안이 글로벌 직접 커밋을 달성했다면, 추측적 실행을 통해 클라이언트 측에서 조기 최종화를 달성할 수 있다는 점은 주목할 만하다.

이하에서는, 정직한 리더 시퀀스의 다양한 시나리오에서, 동기 상태 동안 *Global Direct Commit*과 최종화에 대한 시간 경계를 제공한다.

Lemma 12

뷰 $v \geq v_s$ 에서 시작하여 세 명의 연속된 올바른 리더 L_v, L_{v+1}, L_{v+2} 가 존재한다면, L_{v+1} 이 제안한 제안 p_{v+1} 은 p_{v+1} 을 최초로 수신한 올바른 검증자로부터 3Δ 이내에 글로벌 직접 커밋을 달성한다.

Proof.

글로벌 직접 커밋의 정의에 따르면, 뷰 $v + 2$ 에서

$p_{v+2}.block.qc.proposal_id = p_{v+1}.proposal_id$ 를 만족하는 후속 제안 p_{v+2} 에 대해 $f + 1$ 명의 올바른 검증자가 투표하면, 제안 p_{v+1} 은 이 속성을 달성한다.

L_{v+1} 이 올바르므로, 최초의 올바른 검증자 j 는

$p_{v+1}.block.qc.proposal_id = p_v.proposal_id$ 를 만족하는 제안 p_{v+1} 을 수신하고 뷰 $v + 1$ 로 진입한다. 최대 Δ 이내에, 다른 모든 올바른 검증자들은 p_{v+1} 을 수신하고, 자신의 뷰를 $v + 1$ 로 증가시키며, 자신의 *tip*을 p_{v+1} 로 갱신하고, L_{v+2} 에게 자신의 투표를 보낸다.

올바른 리더 L_{v+2} 는 추가 Δ 구간 내에 p_{v+1} 에 대한 최소 $2f + 1$ 개의 투표를 수신한다. 이러한 투표들을 사용하여, L_{v+2} 는 $p_{v+2}.block.qc.proposal_id = p_{v+1}.proposal_id$ 를 만족하는 새로운 제안 p_{v+2} 를 구성하고, 이를 모든 검증자에게 브로드캐스트한다. 추가 Δ 이내에, 모든 올바른 검증자들은 p_{v+2} 를 수신하고, 자신의 뷰를 $v + 2$ 로 증가시키며, 자신의 tip 을 $tip(p_{v+2})$ 로 갱신하고, p_{v+2} 에 대한 투표를 L_{v+3} 에게 보낸다.

이 시점에서, 적어도 $f + 1$ 명의 올바른 검증자들이 p_{v+1} 을 인증하는 p_{v+2} 에 대해 투표했으므로, 글로벌 직접 커밋 속성이 만족된다. 이는 p_{v+1} 이 글로벌 직접 커밋을 달성했음을 보장한다.

따라서, p_{v+1} 을 최초로 수신하여 $v + 1$ 로 전환한 시점부터, L_v, L_{v+1}, L_{v+2} 가 올바른 경우 p_{v+1} 의 글로벌 직접 커밋 달성 과정은 3Δ 이내에 완료된다.

Lemma 13

뷰 $v \geq v_s$ 에서 시작하여 두 명의 연속된 올바른 리더 L_v 와 L_{v+1} 이 존재한다면, L_v 가 제안한 제안 p_v 는 뷰 v 에 최초로 진입한 올바른 검증자로부터 5Δ 이내에 글로벌 직접 커밋을 달성한다.

Proof.

Lemma 7에 따라, 뷰 v 에 최초로 진입한 올바른 검증자로부터 2Δ 이내에 모든 올바른 검증자들이 뷰 v 로 전환한다. 리더 L_v 는 유효한 제안 p_v 를 제안하고, 이는 Δ 이내에 모든 검증자에게 전파된다. 검증자들은 p_v 에 대한 자신의 투표를 L_{v+1} 에게 보낸다. 투표가 L_{v+1} 에게 수신되는 데에는 최대 Δ 가 소요된다.

다음 올바른 리더 L_{v+1} 은 p_v 에 대한 $2f + 1$ 개의 투표를 수집하고, p_v 에 대한 정족수 증명서(QC)를 담은 새로운 제안 p_{v+1} 을 구성하여 브로드캐스트한다. 모든 올바른 검증자들은 p_{v+1} 을 수신하고 또 다른 Δ 이내에 자신의 $tips$ 을 p_{v+1} 로 갱신한다. p_{v+1} 을 $f + 1$ 명의 올바른 검증자가 관측하고 그에 대해 투표하면, p_v 의 글로벌 직접 커밋 속성이 만족된다.

만약 L_v 가 Lemma 7에 따라 최대 2Δ 이후에 뷰 v 에 진입하는 마지막 검증자라면, 모든 검증자들이 p_v 의 글로벌 직접 커밋을 달성하는 데 추가로 최대 3Δ 가 소요된다. 따라서, p_v 의 글로벌 직접 커밋에 필요한 총 시간은 뷰 v 에 최초로 진입한 올바른 검증자로부터 최대 5Δ 이다.

Lemma 14

뷰 $v \geq v_s$ 에서 시작하여 네 명의 연속된 올바른 리더 $L_v, L_{v+1}, L_{v+2}, L_{v+3}$ 가 존재한다면, L_{v+1} 이 제안한 제안 p_{v+1} 은 p_{v+1} 을 최초로 수신한 올바른 검증자로부터 5Δ 이내에 커밋(최종화)된다.

Proof.

Lemma 12에 따르면, L_{v+2} 가 제안한 p_{v+2} 를 올바른 검증자들이 수신하고 p_{v+2} 에 대한 자신의 투표를 던질 때, 제안 p_{v+1} 은 3Δ 이내에 글로벌 직접 커밋을 달성한다. 이러한 투표들은 L_{v+3} 에게 전달되며, 전달에는 최대 Δ 가 소요된다. $2f + 1$ 개의 투표를 수신하면, L_{v+3} 는 p_{v+2} 에 대한 정족수 증명서(QC)를 담은 새로운 제안 p_{v+3} 를 구성하여 브로드캐스트한다(

$p_{v+3}.block.qc.proposal_id = p_{v+2}.proposal_id$). 이 제안은 Δ 이내에 모든 올바른 검증자들에게 도달한다.

커밋 조건은 검증자들이 두 개의 연속된 QC를 관측하면 만족된다: p_{v+1} 을 인증하는 $p_{v+2}.block.qc$ 와 p_{v+2} 를 인증하는 $p_{v+3}.block.qc$.

따라서, $p_{v+3}.block.qc$ 가 관측되면 제안 p_{v+1} 은 커밋된다. $L_v, L_{v+1}, L_{v+2}, L_{v+3}$ 가 올바른 경우, p_{v+1} 을 최초로 수신한 올바른 검증자 시점부터 p_{v+1} 이 커밋되는 데 필요한 총 시간 상한은 $3\Delta + \Delta + \Delta = 5\Delta$ 이다.

Lemma 15

뷰 $v \geq v_s$ 에서 시작하여 세 명의 연속된 올바른 리더 L_v, L_{v+1}, L_{v+2} 가 존재한다면, L_v 가 제안한 제안 p_v 는 뷰 v 에 최초로 진입한 올바른 검증자로부터 7Δ 이내에 커밋(최종화)된다.

Proof.

Lemma 13에 따르면, 제안 p_v 는 뷰 v 에 최초로 진입한 올바른 검증자로부터 5Δ 이내에 글로벌 직접 커밋을 달성한다. 이어서, 올바른 검증자들은 p_{v+1} 에 대한 자신의 투표를 던지며, 이는 최대 Δ 이내에 L_{v+2} 에게 수신된다. $2f + 1$ 개의 투표를 수신하면, L_{v+2} 는 p_{v+2} 를 제안하고, 이는 추가 Δ 이내에 모든 올바른 검증자들에게 도달한다.

p_{v+2} 내의 정족수 증명서(QC)가 p_{v+1} 과 p_v 를 연속적으로 연결한다는 사실($p_{v+2}.block.qc.proposal_id = p_{v+1}.proposal_id$ 및 $p_{v+1}.block.qc.proposal_id = p_v.proposal_id$)을 관측하면, 커밋 규칙이 만족되어 p_v 가 최종화된다.

따라서, p_v 가 커밋되는 데 필요한 총 시간은, 뷰 v 에 최초로 진입한 올바른 검증자 시점부터 $5\Delta + \Delta + \Delta = 7\Delta$ 로 경계 지어진다.

Acknowledgment

우리는 본 작업에 대한 유익한 논의와 피드백을 제공해 준 John Bergschneider, Yacine Dolivet, Babak Gilkalaye, Tobias Klenze, Michael Setrin에게 감사한다. 특히, 본 작업의 초안에 대해 제안을 해 주고 상세한 수정을 해 준 Tobias Klenze에게 특별한 감사를 전한다.

참고 문헌

1. Alhaddad, N., Duan, S., Varia, M., Zhang, H.: *Succinct erasure coding proof systems*. Cryptology ePrint Archive, Paper 2021/1500 (2021), <https://eprint.iacr.org/2021/1500>
2. Babel, K., Daian, P., Kelkar, M., Juels, A.: Clockwork finance: Automated analysis of economic security in smart contracts. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 2499–2516. IEEE (2023)
3. Baudet, M., Ching, A., Chursin, A., Danezis, G., Garillot, F., Li, Z., Malkhi, D., Naor, O., Perelman, D., Sonnino, A.: State machine replication in the Libra blockchain (2019), <https://diem-developers->

components.netlify.app/papers/diem-consensus-state-machine-replication-in-the-diem-blockchain/2019-06-28.pdf

4. Bracha, G.: Asynchronous byzantine agreement protocols. *Information and Computation* **75**(2), 130–143 (Nov 1987), [https://doi.org/10.1016/0890-5401\(87\)90054-X](https://doi.org/10.1016/0890-5401(87)90054-X)
5. Bravo, M., Chockler, G., Gotsman, A.: Making byzantine consensus live. *Distributed Computing* **35**(6), 503–532 (Dec 2022), <https://doi.org/10.1007/s00446-022-00432-y>
6. Buchman, E.: Tendermint: Byzantine fault tolerance in the age of blockchains (Jun 2016)
7. Byers, J.W., Luby, M., Mitzenmacher, M., Rege, A.: A digital fountain approach to reliable distribution of bulk data. In: *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. pp. 56–67. SIGCOMM '98, Association for Computing Machinery, New York, NY, USA (1998)
8. Cachin, C., Tessaro, S.: Asynchronous verifiable information dispersal. In: *Proceedings of the 19th International Conference on Distributed Computing (DISC'05)*, Springer-Verlag, Berlin, Heidelberg (2005), https://doi.org/10.1007/11561927_42
9. Castro, M.: *Practical Byzantine Fault Tolerance*. Ph.D., MIT (Jan 2001), also as Technical Report MIT-LCS-TR-817
10. Castro, M., Liskov, B.: Practical Byzantine fault tolerance. In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99)*, USENIX Association, Berkeley, CA, USA (1999)
11. Chan, T.H.H., Pass, R., Shi, E.: Pala: A simple partially synchronous blockchain. *Cryptology ePrint Archive*, Report 2018/981 (2018), <https://eprint.iacr.org/2018/981>
12. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: *2020 IEEE Symposium on Security and Privacy (SP)*. pp. 910–927 (2020)
13. Doidge, I., Ramesh, R., Shrestha, N., Tobkin, J.: Moonshot: Optimizing chain-based rotating leader BFT via optimistic proposals (2024), <https://arxiv.org/abs/2401.01791>
14. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)* **35**(2), 288–323 (1988)
15. Garay, J.A., Gennaro, R., Jutla, C., Rabin, T.: Secure distributed storage and retrieval. *Cryptology ePrint Archive*, Paper 1998/025 (1998), <https://eprint.iacr.org/1998/025>
16. Gelashvili, R., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A., Xiang, Z.: Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback (2021)
17. Giridharan, N., Suri-Payer, F., Ding, M., Howard, H., Abraham, I., Crooks, N.: BeeGees: Stayin' alive in chained BFT. In: *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*. pp. 233–243. PODC '23,

Association for Computing Machinery, New York, NY, USA (2023).

<https://doi.org/10.1145/3583668.3594572>

18. Golan-Gueta, G., Abraham, I., Grossman, S., Malkhi, D., Pinkas, B., Reiter, M.K., Seredinschi, D., Tamir, O., Tomescu, A.: SBFT: a scalable decentralized trust infrastructure for blockchains. *CoRR* abs/1804.01626 (2018)
19. Guo, B., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Dumbo: Faster asynchronous bft protocols. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. pp. 803–818. CCS '20, Association for Computing Machinery, New York, NY, USA (2020). <https://doi.org/10.1145/3372297.3417262>
20. He, Z., Li, J., Wu, Z.: Don't trust, verify: The case of slashing from a popular ethereum explorer. In: *Companion Proceedings of the ACM Web Conference 2023*. pp. 1078–1084 (2023)
21. Jalalzai, M.M., Feng, C., Lemieux, V.: Vbft: Veloce byzantine fault tolerant consensus for blockchains (2024), <https://arxiv.org/abs/2310.09663>
22. Jalalzai, M.M., Niu, J., Feng, C., Gai, F.: Fast-hotstuff: A fast and resilient hotstuff protocol (2021)
23. Kadhe, S., Chung, J., Ramchandran, K.: Sef: A secure fountain architecture for slashing storage costs in blockchains (2019), <https://arxiv.org/abs/1906.12140>
24. Kang, D., Gupta, S., Malkhi, D., Sadoghi, M.: Hotstuff-lt: Linear consensus with one-phase speculation (2024), <https://arxiv.org/abs/2408.04728>
25. Kotla, R., Clement, A., Wong, E., Alvisi, L., Dahlin, M.: Zyzzyva: Speculative Byzantine fault tolerance. *Communications of the ACM* **51**(11), 86–95 (Nov 2008). <https://doi.org/10.1145/1400214.1400236>
26. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. *ACM Transactions on Programming Languages and Systems* **4**(3), 382–401 (1982)
27. Luby, M.: Lt codes. In: *Proceedings of the 43rd Symposium on Foundations of Computer Science*. p. 271. FOCS '02, IEEE Computer Society, USA (2002)
28. Malkhi, D., Nayak, K.: Extended abstract: Hotstuff-2: Optimal two-phase responsive BFT. *Cryptology ePrint Archive*, Paper 2023/397 (2023), <https://eprint.iacr.org/2023/397>
29. Niu, J., Gai, F., Jalalzai, M.M., Feng, C.: On the performance of pipelined hotstuff. In: *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. pp. 1–10 (2021)
30. *Slashed validators*. <https://www.beaconcha.in/validators/slashings>, accessed: 2024-10-04
31. Vonlanthen, Y., Sliwinski, J., Albarello, M., Wattenhofer, R.: Banyan: Fast rotating leader BFT. In: *Proceedings of the 25th International Middleware Conference*. p. 494–507. Middleware '24, Association for Computing Machinery, New York, NY, USA (2024), <https://doi.org/10.1145/3652892.3700788>
32. Wensley, J.H., Lamport, L., Goldberg, J., Charles, Milton, W., Green, LEVITT, K.N., Melliar-Smith, P.M., Robert, E.: Sift: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE* **66**, 1240–1255 (1978), <https://api.semanticscholar.org/CorpusID:4002660>
33. Yang, L., Park, S.J., Alizadeh, M., Kannan, S., Tse, D.: DispersedLedger: High-throughput byzantine consensus on variable bandwidth networks. In: *19th*

USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). pp. 493–512. USENIX Association, Renton, WA (Apr 2022), <https://www.usenix.org/conference/nsdi22/presentation/yang>

34. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: Hotstuff: BFT consensus in the lens of blockchain. *arXiv preprint arXiv:1803.05069* (2018)
35. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: Hotstuff: BFT consensus with linearity and responsiveness. In: *Proceedings of the 2019 ACM PODC*. pp. 347–356. PODC '19, Association for Computing Machinery, New York, NY, USA (2019)