APPENDIX B
TLA+ FORMAL SPECIFICATION AND VERIFICATION

To rigorously validate the safety and liveness properties of AdaptiveBFT, particularly its *Adaptive View-Change* (AVC) and *Adaptive Pipeline Scheduling* (APS) mechanisms, we developed a comprehensive TLA+ formal specification and checked it with TLC.

A. AdaptiveBFT Formalization Structure

A formal specification of AdaptiveBFT appears in Figure 10, with supporting definitions in Figures 11 to 13 and correctness properties in Figure 14.

The split follows a five-figure, reviewer-friendly structure:

1) Main protocol state-machine excerpt (Fig. 10).
2) Message/proposal/state type system (Fig. 11).
3) AVC + RVS support definitions (Fig. 12).
4) APS + mempool support definitions (Fig. 13).
5) Correctness properties only (Fig. 14).

In addition to this split view, we also provide two complementary rendered artifacts in the repository root/module folder: `AdaptiveBFT.pdf` (monolithic consolidated rendering) and `APS_Scheduler.pdf` (scheduler-focused support rendering).

B. Modeling Scope and Abstractions

To keep exhaustive TLC exploration tractable without obscuring the core protocol logic, we make the following explicit abstractions:

A1: Bounded BFT domain. The baseline model instantiates $f = 1$, $n = 3f + 1 = 4$, and $q = 2f + 1 = 3$, with finite bounds on views, batching, pipeline depth, timeout, reputation range, and transaction set.

A2: Typed message/proposal semantics. Protocol artifacts (MINORMESSAGE, FULLMESSAGE, TEPROPOSAL, VPROPOSAL, NPMESSAGE, SYNMESSAGE) are represented as typed TLA+ records.

A3: Cryptographic abstraction. QC checks and RVS/VRF behavior are encoded as deterministic typed operators over bounded domains with explicit witness validation; cryptographic hardness assumptions remain out of model scope.

A4: Timer and network abstraction. Timeout and delay effects are represented via symbolic state (`stageTimer`, timeout votes, `Stable`/`Unstable`) rather than packet-level or wall-clock models.

A5: Adaptive adversary abstraction. Adaptive corruption is modeled as bounded perturbation transitions parameterized by `MaxAttackCount` in attack-enabled configurations.

A6: Reputation-game abstraction. Bayesian-report normalization, finite PageRank-style updates, and payoff/cost-based truthful-report utilities are encoded as finite operators over bounded report/reputation domains.

A7: APS control-plane abstraction. Adaptive scheduling is modeled as a finite reconfiguration loop over five abstract states: Monitor, Sample, Estimate, Explore, and Deploy, with finite candidate configurations.

A8: Eventual synchrony abstraction. Liveness is checked under explicit fairness constraints in the liveness wrappers, capturing eventual progress after unstable episodes.

A9: Safety-wrapper quorum compression. In `MC_AdaptiveAttack`, per-replica vote/ack interleavings are collapsed into quorum-completion steps to keep adaptive-attack safety exploration tractable.

These assumptions define the exact claim boundary of the model-checking results reported below.

C. Checked Properties and TLC Results

The correctness profile in Figure 14 consists of 13 safety invariants (I1–I13), an aggregate `Safety` formula, and temporal liveness goals (`Liveness` and related progress formulas).

a) Baseline claim (`MaxView=2`).: Using the TLC model-checker, we are able to check that all possible executions of the bounded AdaptiveBFT model in a 4-node system with one Byzantine node satisfy the `safety` and `liveness` properties in Figure 14 under the eventual-synchrony abstraction (encoded by fairness constraints), for a maximum of 3 protocol rounds (views 0..2). In this baseline setting, the full campaign (`MC_AdaptiveAttack`, `MC_LivenessAPS`, `MC_LivenessAPSAttack`) completes in about 26 seconds on an AMD Ryzen 7 7840H platform (16 vCPUs, 15 GiB RAM, Java `-Xmx8G`), with no TLC-reported invariant or temporal-property violations.

b) Strengthened claim (`MaxView=4`).: To increase depth coverage, we provide a strengthened profile that changes only `MaxView` from 2 to 4 (all other constants fixed). Under this profile, TLC reports no invariant or temporal-property violation: `MC_AdaptiveAttack` explores 113,041 generated states (32,774 distinct) in 21s, `MC_LivenessAPS` explores 16,791 generated states (5,047 distinct) in 39s, and `MC_LivenessAPSAttack` explores 104,191 generated states (33,269 distinct) in 55s.

c) Refinement-transfer diagnostics.: To reduce the remaining gap between bounded concrete exploration and the abstract theorem layer, we additionally check a dedicated transfer wrapper (`MC_RefinementTransfer`) that enforces `StepProjectionChecked` and `StepBoxProjectionChecked` under a tractable state-space constraint. In baseline/`MaxView=3`/`MaxView=4` profiles, TLC reports no transfer-property violation and explores 269,339 generated states (47,796 distinct, diameter 30), with runtime 1m10s/1m21s/1m21s, respectively. In addition, we embed wrapper-level transfer checks directly in `MC_AdaptiveAttack`, `MC_LivenessAPS`, and `MC_LivenessAPSAttack`; these also pass in baseline and `MaxView=4` runs.

Detailed state-space scale and per-model runtime are reported in Tables II to X.

Local variables:
transition families M_1, M_2, M_3, M_4:
M_1: ingress + pre-validation
M_2: APS control loop
M_3: consensus pipeline
M_4: AVC + RVS + sync update
$Next \triangleq M_1 \vee M_2 \vee M_3 \vee M_4$

procedure MOne ▷ MainFlow:9–12
  return $InjectTxStep \vee PreValidateStep \vee AgeTxStep$
procedure MTwo ▷ MainFlow:14–19
  return $DetectAnomaly \vee SampleGrid \vee EstimateGrid$ $\vee ExploreGrid \vee DeployConfig$
procedure MThree ▷ MainFlow:21–30
  return $PreOrder \vee GenTentative \vee RecoverFull$ $\vee PrepareVote \vee PrepareQC$ $\vee PreCommitVote \vee PreCommitQC$ $\vee CommitVote \vee DecideBlock$
procedure MFour ▷ MainFlow:32–40
  return $Tick \vee CastTimeoutVote \vee StartViewChange$ $\vee BroadcastNP \vee ConfirmNewPrimary$ $\vee BroadcastSyn \vee SendSynAck$ $\vee CompleteViewChange$
procedure Next ▷ MainFlow:42–46
  return $MOne \vee MTwo \vee MThree \vee MFour$

Fig. 10: High-level TLA+ state-machine excerpt of AdaptiveBFT.

Local variables:
universes: $MsgType$, $ConsensusPhase$, $SchedulerStateType$, $NetworkConditionType$
record tags: `Minor`, `Full`, `TeProposal`, `ReProposal`, `VProposal`, `NPMessage`, `SynMessage`
helper operators: $QC(view)$, $NilQC$, $SamePrefix$

procedure MkMinor(v, a, q, s) ▷ Types:23–24
  return $\langle$`Minor`$, v, a, q, s\rangle$
procedure MkFull(v, tx, q, p, s) ▷ Types:26–27
  return $\langle$`Full`$, v, tx, q, p, s\rangle$
procedure MkTe(v, al, q, p, s)
  return $\langle$`TeProposal`$, v, al, q, p, s\rangle$
procedure MkRe(v, tx, q, p, s)
  return $\langle$`ReProposal`$, v, tx, q, p, s\rangle$
procedure MkV(v, rv, q, p, s)
  return $\langle$`VProposal`$, v, rv, q, p, s\rangle$
procedure MkNP(v, l, t, k, π, q, s)
  return $\langle$`NPMessage`$, v, l, t, k, \pi, q, s\rangle$
procedure MkSyn(v, l, rv, q, s)
  return $\langle$`SynMessage`$, v, l, rv, q, s\rangle$
procedure SamePrefix(s_1, s_2)
  $m \leftarrow MinNat(Len(s_1), Len(s_2))$
  return $\forall i \in 1..m : s_1[i] = s_2[i]$

Fig. 11: TLA+ type and record definitions for messages, proposals, and protocol state.

Local variables:
$rep[n]$: bounded reputation
$thr, view, \kappa$: threshold, epoch, strike parameter
evidence tuple $ev = \langle ticket, strikes, proof\rangle$

procedure CandSet(rep, thr) ▷ AVC_RVS:38–46
  $base \leftarrow \{n : rep[n] \geq thr\}$
  $hi \leftarrow \{n \in base : rep[n] \geq AverageRep(rep)\}$
  return hi if non-empty; else $base$ if non-empty; else all nodes
procedure RVSContext(rep, thr) ▷ AVC_RVS:56–65
  $cand \leftarrow CandSet(rep, thr)$
  return $\langle cand, SumRep(rep, cand), RVSKappa(cand, thr), \max(rep)\rangle$
procedure SelectLeader($rep, thr, view$) ▷ AVC_RVS:67–86
  $ctx \leftarrow RVSContext(rep, thr)$
  $wins \leftarrow ValidWinners(rep, ctx, view)$
  return $PickWinner(rep, wins, ctx, view)$
procedure LeaderEvidence($rep, thr, view, ldr$) ▷ AVC_RVS:88–100
  $ctx \leftarrow RVSContext(rep, thr)$
  $r \leftarrow SortitionResult(rep, ldr, ctx, view)$
  return $\langle r.ticket, r.strikes, r.proof\rangle$
procedure VerifyLeader($rep, thr, view, ldr, ev$) ▷ AVC_RVS:102–139
  $ctx \leftarrow RVSContext(rep, thr)$
  $r \leftarrow SortitionResult(rep, ldr, ctx, view)$
  $ok \leftarrow SortitionVerify(rep, ldr, ctx, view, r)$
  return $(ldr \in ctx.cand) \wedge ok$ $\wedge\ (ev = \langle r.ticket, r.strikes, r.proof\rangle)$
procedure DecayUpdate(rep, n, h, m, a, b) ▷ AVC_RVS:141–160
  $obs \leftarrow$ m if h else 0
  return $DecayUpdateByObservation(\ldots)$ with arguments (rep, n, obs, m, a, b)

Fig. 12: TLA+ support definitions for AVC and RVS.

Local variables:
APS config $cfg = [batchSize, pipelineDepth, timeout]$
scheduler state $\in \{\texttt{Monitor}, \texttt{Sample}, \texttt{Estimate}, \texttt{Explore}, \texttt{Deploy}\}$
mempool state: $pool, age, hotTx, warmTx, agingThreshold$

procedure NetFeasible(cfg, net) ▷ APS_Mempool:17–24
  if $net = \texttt{Unstable}$ **then**
    return $(cfg.timeout \geq 2) \wedge (cfg.batchSize \leq 2) \wedge (cfg.pipelineDepth \leq 2)$
  else
    return $(cfg.timeout \geq 1) \wedge (cfg.batchSize \geq 1) \wedge (cfg.pipelineDepth \geq 1)$
procedure Perf(cfg, net) ▷ APS_Mempool:26–40
  return $2 \cdot LatencyScore(cfg, net) - ThroughputScore(cfg)$
procedure NextSched(st) ▷ APS_Mempool:47–52
  return Monitor → Sample → Estimate → Explore → Deploy → Monitor
procedure Prio($tx, age, hot, warm, th$) ▷ APS_Mempool:56–59
  return High if $(tx \in hot) \vee (age[tx] \geq th)$ else Mid if $(tx \in warm)$ else Low
procedure Front($pool, age, hot, warm, th$) ▷ APS_Mempool:65–93
  $(hi, mid, lo) \leftarrow$ partition by Prio($\cdot$)
  return hi if non-empty; else mid if non-empty; else lo
procedure PickBatch($pool, age, hot, warm, th, lim$) ▷ APS_Mempool:95–100
  $front \leftarrow Front(pool, age, hot, warm, th)$
  return non-empty $chosen \subseteq front$ with $|chosen| \leq lim$
procedure Recover($abst, valid$) ▷ APS_Mempool:102–103
  return $abst \cap valid$
procedure BumpAge($age, tx, maxAge$) ▷ APS_Mempool:105–109
  return bounded age increment; reset on commit

Fig. 13: TLA+ support definitions for APS and mempool processing.

Local variables:
safety invariants: $I1, \ldots, I13$
liveness goals: $L1$ (one commit), $L2$ (two commits), $L3$ (infinite collect), $L4$ (view progress)
macros: `Safety`, `Liveness`, wrapper checks

procedure SafetyProfile ▷ Properties:22–35
  return $(I1 \wedge I2 \wedge I3 \wedge I4 \wedge I5 \wedge I6)$
  $\wedge (I7 \wedge I8 \wedge I9 \wedge I10 \wedge I11 \wedge I12 \wedge I13)$
procedure Safety ▷ Properties:44
  return SafetyProfile
procedure Liveness ▷ Properties:48
  return $L2$
procedure LiveSet ▷ Properties:37–40
  return $\{L1, L2, L3, L4\}$
procedure ClaimBoundary ▷ explicit boundary
  return finite-domain TLC checks
  + fairness-based eventual synchrony
  + bounded adaptive perturbation model
procedure ReportedClaim
  return all explored executions satisfy safety and liveness

Fig. 14: TLA+ formalization of AdaptiveBFT safety and liveness properties.

TABLE II: AdaptiveBFT verification settings and checked property profile.

Model	Assumptions	Invariants (abbr.)	Liveness target
MC_AdaptiveAttack	$f = 1$, $n = 4$, $q = 3$; Node = $\{r1, r2, r3, r4\}$; bounded transaction/pipeline/view/timeout/reputation domains; `MaxAttackCount=1`; quorum-compressed vote/ack transitions	I1–I13 + B1–B3[a,d]	W-A*[c]
MC_LivenessAPS	Same bounded domain as left (without attack bound); fairness-structured liveness wrapper	I1–I13 + B1–B3[a,d]	L1 + W-P*[c]
MC_LivenessAPSAttack	Same bounded domain as left + `MaxAttackCount=1`; joint fairness and recovery wrapper	I1–I13 + B1–B3[a,d]	L2 + W-J*[c]
MC_RefinementTransfer	Same bounded constants as baseline/mv3/mv4; transfer constraint (`schedulerState=Monitor`, stable network, initial view/chain)	I1–I13 + B1–B3[a,d]	T0–T4[b]

[a] I1 TypeOK; I2 Consistency; I3 NoForkPerView; I4 PipelineBounded; I5 QCLocked; I6 QCViewSafety; I7 MempoolSoundness; I8 ProposalFlowSafety; I9 ViewChangeMessageSafety; I10 ChainParentSafety; I11 PrimaryEligibilitySafety; I12 ReconfigurationSafety; I13 DecoupledPipelineSafety.
[b] Transfer checks: T0 InitProjectionChecked; T1 StepProjectionChecked; T2 StepBoxProjectionChecked; T3 SpecProjectionChecked; T4 SpecToAbstractSpecChecked.
[c] Wrapper projection families: W-A* = attack wrapper projection checks (Init/Step/StepBox/Spec/SpecToAbstractSpec); W-P* and W-J* denote the same five check kinds for APS and Joint wrappers.
[d] Wrapper finite bridge diagnostics: B1 RefinementInvariantCore*; B2 SafetyBridgeFinite*; B3 CommitProjectionShape* (`Attack`/`APS`/`Joint`/`Transfer` suffix by model).

TABLE III: AdaptiveBFT deepened-profile TLC outcomes (`MaxView=5`, bounded full-suite).

Model	States Generated	Distinct States	Diameter	Time	TLC Result	Claim Scope
MC_AdaptiveAttack	299,733	86,886	80	01min 42s	Pass	Safety + W-A*
MC_LivenessAPS	20,143	6,053	115	01min 14s	Pass	Safety + L1 + W-P*
MC_LivenessAPSAttack	143,795	45,553	124	01min 39s	Pass	Safety + L2 + W-J*
MC_RefinementTransfer	269,339	47,796	30	01min 27s	Pass	T0–T4

Snapshot source: `make suite-mv5` (artifact: `docs/results/mv5_suite_latest.md`, executed on 2026-02-21). This is bounded finite-domain evidence with deeper view depth; it is not an unbounded theorem proof.

TABLE IV: AdaptiveBFT quick n=7 profile outcome (`n=7`, `f=2`, `q=5`, `MaxView=1`).

Model	States Generated	Distinct States	Diameter	Time	TLC Result	Claim Scope
MC_AdaptiveAttack (`n7lite`)	5,009	1,454	39	03min 19s	Pass	Safety + W-A*

Snapshot source: `make scale-n7lite` (artifact: `docs/results/scale_n7lite_suite_latest.md`, executed on 2026-02-21).
Invariant profile: `TypeOKLite` + I2–I13 (scalable field-level typing for large constructor domains).
This profile is a bounded quick sanity check, not an unbounded theorem proof.

TABLE V: AdaptiveBFT bounded scale-sanity outcome (`n=7`, `f=2`, `q=5`, `MaxView=2`).

Model	States Generated	Distinct States	Diameter	Time	TLC Result	Claim Scope
MC_AdaptiveAttack (`n7`)	15,045	4,366	49	04min 50s	Pass	Safety + W-A*

Snapshot source: `make scale-n7` (artifact: `docs/results/scale_n7_attack_latest.md`, executed on 2026-02-21).
Invariant profile: `TypeOKLite` + I2–I13 (scalable field-level typing for large constructor domains).
This profile is a bounded scale sanity check, not an unbounded theorem proof.

TABLE VI: AdaptiveBFT refinement-transfer TLC outcomes (bounded diagnostics).

Profile	States Generated	Distinct States	Diameter	Time	TLC Result	Checked Properties
baseline (`MaxView=2`)	269,339	47,796	30	01min 10s	Pass	T0–T4
`MaxView=3`	269,339	47,796	30	01min 21s	Pass	T0–T4
`MaxView=4`	269,339	47,796	30	01min 21s	Pass	T0–T4

Commands: `make test-transfer`, `make test-transfer-mv3`, `make test-transfer-mv4` (executed on 2026-02-21).
T0–T4 definitions are listed in Table II.
Transfer diagnostics are bounded and constraint-scoped; they provide executable evidence for concrete-step to abstract-step/stutter alignment, not a closed unbounded refinement proof.

TABLE VII: AdaptiveBFT wrapper-integrated projection outcomes (baseline and `MaxView=4`).

Profile	Model	States Generated	Distinct States	Diameter	Time	Checked Projection Properties
baseline	MC_AdaptiveAttack	15,045	4,366	49	11s	W-A*
baseline	MC_LivenessAPS	10,087	3,035	63	13s	W-P*
baseline	MC_LivenessAPSAttack	45,095	14,737	73	21s	W-J*
MaxView=4	MC_AdaptiveAttack	113,041	32,774	70	43s	W-A*
MaxView=4	MC_LivenessAPS	16,791	5,047	98	39s	W-P*
MaxView=4	MC_LivenessAPSAttack	104,191	33,269	107	01min 01s	W-J*

Snapshot source: `make wrapper-projection` (latest artifact: `docs/results/wrapper_projection_latest.md`, executed on 2026-02-21).
W-A*/W-P*/W-J* abbreviations are defined in Table II.
These checks are bounded finite-domain diagnostics integrated into the main wrappers.

TABLE VIII: AdaptiveBFT TLC outcomes (bounded model checking, baseline profile).

Model	States Generated	Distinct States	Diameter	Time	TLC Result	Claim Scope
MC_AdaptiveAttack	15,045	4,366	49	5s	Pass	Safety only
MC_LivenessAPS	10,087	3,035	63	< 1s	Pass	Safety + L1
MC_LivenessAPSAttack	45,095	14,737	73	20s	Pass	Safety + L2

Snapshot source: command `make matrix` (executed on 2026-02-15, after weighted-strike RVS upgrade).
Runtime platform: AMD Ryzen 7 7840H, 16 vCPUs, 15 GiB RAM, Linux (WSL2 Ubuntu), Java `-Xmx8G`.
Worker policy: `MC_AdaptiveAttack` uses 4 workers; liveness models use `-workers auto`.
These are bounded finite-domain model-checking results, not unbounded theorem proofs.

TABLE IX: AdaptiveBFT deeper-profile TLC outcomes (`MaxView=3`, bounded model checking).

Model	States Generated	Distinct States	Diameter	Time	TLC Result	Claim Scope
MC_AdaptiveAttack	5,936,703	1,196,611	69	7m39s	Pass	Safety only
MC_LivenessAPS	14,762,371	3,405,172	89	59m13s	Pass	Safety + L1
MC_LivenessAPSAttack	56,392,041	14,807,882	98	3h24m	Pass	Safety + L2

Snapshot source: `make test-attack-mv3`, `make test-aps-mv3`, `make test-joint-mv3` (executed on 2026-02-15).
Runtime platform: AMD Ryzen 7 7840H, 16 vCPUs, 15 GiB RAM, Linux (WSL2 Ubuntu), Java `-Xmx8G`.
Worker policy: `MC_AdaptiveAttack` uses 4 workers; liveness models use `-workers auto`.
These are bounded finite-domain model-checking results, not unbounded theorem proofs.

TABLE X: AdaptiveBFT strengthened-profile TLC outcomes (`MaxView=4`, bounded model checking).

Model	States Generated	Distinct States	Diameter	Time	TLC Result	Claim Scope
MC_AdaptiveAttack	113,041	32,774	70	21s	Pass	Safety only
MC_LivenessAPS	16,791	5,047	97	39s	Pass	Safety + L1
MC_LivenessAPSAttack	104,191	33,269	106	55s	Pass	Safety + L2

Snapshot source: `make test-attack-mv4`, `make test-aps-mv4`, `make test-joint-mv4` (executed on 2026-02-15).
Runtime platform: AMD Ryzen 7 7840H, 16 vCPUs, 15 GiB RAM, Linux (WSL2 Ubuntu), Java `-Xmx8G`.
Worker policy: `MC_AdaptiveAttack` uses 4 workers; liveness models use `-workers auto`.
Strengthened profile parameters keep $n = 4$, $f = 1$, and $q = 3$, changing only `MaxView` from 2 to 4.
These are bounded finite-domain model-checking results, not unbounded theorem proofs.