

Sui

Tunnel Performance Verification Report

Final v1 | October 14, 2026

Prepared by CertiK



Table of Contents

Table of Contents.....	1
Executive overview.....	2
Prerequisite.....	3
Transaction Definition.....	4
Tunnel effective TPS.....	5
Concurrent tunnels on one time axis.....	5
Supplied Sui Source Data.....	6
1 Measurement methodology.....	7
1.1 Counting and ordering.....	7
1.2 Metric definitions.....	7
1.3 Peak Tunnel TPS.....	8
1.4 Sustained Tunnel TPS.....	8
2 Verification Procedure.....	9
Hardware and OS Information.....	9
2.1 Verification Overview.....	9
2.2 Full verification workflow.....	10
2.3 Verification performed.....	11
3 Performance Results.....	12
3.1 Operation and failure counts.....	12
3.2 Aggregate rates.....	13
3.3 Peak and sustained TPS results.....	16
Appendix.....	17
Data Summary.....	17
Sui Tunnel Introduction.....	18
off-chain execution, on-chain trust.....	18
Reduce L1 load, but opening and settlement still converge on Sui.....	18
Beyond payment channels.....	19

Executive overview

This report is based on the Mainnet performance campaign on October 07th, 2026 UTC, the Sui Basecamp Day live performance data. The campaign recorded 15,241,348,297* tunnel transactions during the measured play windows across 7,680 tunnels, all reported as opened and settled on chain. The TPS results are highlighted as below.

Peak Tunnel TPS Complete 1-second window	Sustained Tunnel TPS Maximum 60-second rolling average
40,682,328.00	40,339,143.65

Campaign available arrival complete second calculation; Units: committed tunnel transactions per second(TPS).

*: The time bucketed metrics excluded 81 app tunnels, resulting in a reported total of 15,241,348,297 committed transactions.

Verified counts and campaign averages

Campaign measure	Result
Executions recorded VERIFIED / expected	7,680 / 7,680
Committed in-window transactions	15,241,348,297
Full history admitted receipts	15,241,348,297

Note: Tunnel TPS in this report means qualifying tunnel(off-chain) transactions per second unless a metric explicitly refers to distinct on-chain transactions. It is not a measurement of Sui on-chain transaction capacity.

Prerequisite

CertiK independently verified and analyzed the data in this report, while the Sui team supplied the campaign data and executed all test scenarios. Funded and opened tunnels on the Sui blockchain, DOPA tunnels process authorized transitions off-chain before settling back on-chain. Measuring qualifying off-chain transitions per second, the primary metric does not reflect Sui instant on-chain throughput. Refer to the Appendix for more details on Sui Tunnel.

The campaign covers 10 runs across 5 workload types, comprising 960 slots and 7,680 verified tunnels. The retained ledger records every execution as *VERIFIED*. This report independently recalculates the counts and average rates from those records and reconciles checkpoint observations. It also explains the verifier's journal, authorization and replay checks, while distinguishing recorded outcomes from a fresh cryptographic replay.

Transaction Definition

A transaction applies an operation to an existing state and produces a result and a new state. TPS(Transaction Per Second) counts qualifying completed operations, not every state value they change. In this assignment, the transaction refers to the tunnel transition receipt.

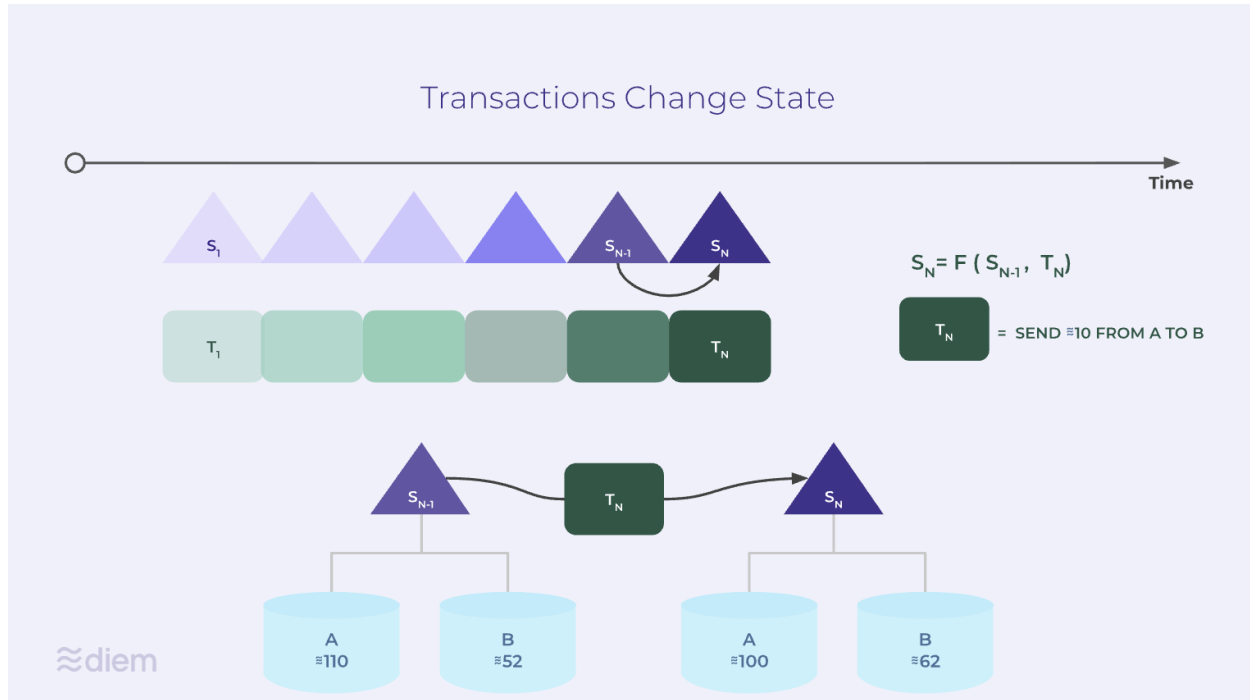


Figure 1. A transaction changes state. Transferring 10 units from A to B changes both balances (A: 110 to 100; B: 52 to 62), but counts as one transaction. Source: [Diem, Transactions and states](#)

Tunnel effective TPS

Sui Tunnel effective TPS: Count each qualifying authority committed tunnel transition receipt once. It must be legal under the frozen protocol, carry all required authorization proofs, be accepted by the authority and appear exactly once as *Committed* in the canonical journal. Participant actions and authority event transitions can both qualify.

A tunnel's final on-chain commitment can represent many off-chain transitions. Its settlement transaction and its off-chain transitions are different counting units. Settlement supports the final outcome; journals, authorization and replay evidence are needed to establish the off-chain work.

TPS is the qualifying count divided by elapsed seconds. For off-chain work, count *committed* records strictly after *WindowOpen* and before *WindowClose* in canonical journal order. An action submitted before the window can qualify if it commits inside it. Record position decides inclusion, while marker timestamps determine duration.

$$TPS_{effective} = \frac{N}{\Delta t} = \frac{1,000 \times N}{D}$$

Note: **N** is the qualifying count, **Δt** is elapsed seconds, and **D** is the same positive interval in milliseconds. Full-history receipt totals, verifier runtime and the evidence retrieval range do not replace the play-window count or duration. Report the boundaries, clock provenance and verification status with the rate; signatures and replay do not independently establish clock accuracy.

Concurrent tunnels on one time axis

Concurrent tunnels perform work during overlapping real time. With identical measurement boundaries, add their disjoint qualifying counts and divide once by the shared duration. Do not sum tunnel lifetimes or multiply the aggregate rate by the number of tunnels.



Figure 2. Multiple tunnels on a common timeline example; three lanes are shown. Squares represent qualifying commits schematically, not their count. Sui checkpoints follow independent boundaries.

All the tunnels open before measurement and share the 2–12 second play window. The values illustrated in the picture are used to explain the calculation and are not measurements from the mainnet campaign in practice.

Supplied Sui Source Data

The following details are fetched directly from the Sui mainnet run, provided as a Sui attributed summary. The presented table reflects the information as below:

Field	Details
Network	Sui mainnet
Chain identifier	4btiuiMPvEENsttpZC7CZ53DruC3MAgfznDbASZ7DR6S
arena_tunnel package	0x9498606724aa4c03dd0e50ce508b3f5555c4915d3f8d0d4525c0f98dd1737eb1
arena_tunnel publish transaction	4cw8ufs5Cgvb849NnQpUbFpqr3GPXBqBV8qusaEQy8yB
mtps package / Stake token	0xcb913929a95bc74d47c7d943730113bc19be32210374bc863729e4929b3eedcd
mtps publish transaction	8XQoaLvjoZtJnGSeiSxDFpdB2dj5d5hriUQrbk6RssRG
Workload distribution	simple_payments_v3: 5 runs, 24 machine reports (5 + 5 + 5 + 5 + 4) flash_v1: 2 runs, 3 machine reports (2 + 1) battleship_v3, tic_tac_toe_v3 and blackjack_v3: 1 run and 1 machine report each
Reported coverage	10 runs; 30 run-scoped machine reports × 32 slots × 8 tunnels = 7,680 tunnels, all opened and settled on chain
Query window [from_ms, to_ms]	1791362460000 to 1791363239000

The documented roster outlines a framework of 960 slots, with 8 tunnels hosted per slot and 10 runs covering 5 workload types. While package and transaction identifiers establish the defined on-chain structure, the query range serves purely as an evidence retrieval window instead of the denominator used for TPS calculations.

1 Measurement methodology

1.1 Counting and ordering

Records are included in the count only if they are *Committed* strictly between *WindowOpen* and *WindowClose* according to canonical journal order, excluding boundary markers. Eligibility is contingent upon commitment. This evaluation relies on the ledger's *committed_in_window* values without re-evaluating raw journal limits.

Within these defined scopes, the documented verifier validates gap and duplicate conditions, authorization and replay prevention, coordinator signatures and journal sequence, archive and segment continuity, as well as execution bindings and revisions. However, it does not construct a global transaction ordering across distinct tunnels.

In summary, the ledger records 15,241,348,297 committed in-window transactions alongside 15,241,348,297 full-history admitted receipts across 7,680 verified benchmark tunnels. Consequently, the retained figures indicate that no receipts were admitted outside of the signed play windows. However, because the ledger does not separately categorize warm-up or cooldown phases, the equality of these counts does not rule out the possibility that such activities took place.

1.2 Metric definitions

Metric	Calculation and scope
Peak Tunnel TPS	Maximum count in a complete UTC-aligned 1-second arrival bucket, after concurrent counts are merged.
Maximum 60-second average	Largest sum of 60 consecutive complete 1-second counts divided by 60.
Minimum 60-second average	Smallest sum of 60 consecutive complete 1-second counts divided by 60.
Summed slot play rate	Sum of each slot's count divided by its own play duration.
Play window average	$R_{play} = \frac{1000N_{play}}{t_{close,max} - t_{open,min}}$ Earliest play opening to latest play closing.
Chain lifecycle average	$R_{life} = \frac{1000N_{play}}{t_{chain,close,max} - t_{chain,open,min}}$ Earliest chain opening to latest chain closing.

Note: In the formulas, N_{play} denotes the committed in-window count. All timestamp endpoints are in milliseconds; the factor 1,000 converts rates to transactions per second.

The 8 tunnels in each slot share the same play-window start and end markers. To calculate TPS, we add the in-window transaction count from each tunnel once and divide the total by one shared time period. For each workload, and for the campaign as a whole, this period runs from

the earliest window opening to the latest window closing. We do not add each per-tunnel TPS value together or multiply a local rate by the number of tunnels. Counts and millisecond timestamps are kept at full precision, and the final average is rounded to two decimal places.

The lifecycle average uses the same transaction count but measures time from the earliest on-chain tunnel opening to the latest on-chain settlement. Play-window times come from the off-chain journal markers, while chain times come from the checkpoint timestamps returned by the observed fullnode. These timestamps reproduce the reported calculation, but they do not prove that the underlying clocks were perfectly synchronized or measure network latency and producer saturation.

1.3 Peak Tunnel TPS

Let B_k count eligible transactions arriving in the UTC-second interval $[k, k + 1)$. Buckets are left inclusive and right exclusive; K_1 contains eligible complete seconds. The saved progress files do not contain the arrival series needed to validate the preliminary implementation or its selected windows.

$$R_{peak} = \max_{k \in K_1} \frac{B_k}{1\ s}$$

To compute strict metrics, only fully observed buckets within the evaluation window are considered, while partial boundary buckets are omitted. When multiple counts occur concurrently, they must be merged prior to identifying the peak value. Fully observed seconds that record no arrivals are counted as zero, whereas unrecorded observations or invalid receipt timestamps denote coverage gaps rather than absence of activity.

1.4 Sustained Tunnel TPS

$$R_{60}(k) = \frac{\sum_{j=0}^{59} B_{k+j}}{60\ s}$$

$$R_{60}^{max} = \max_{k \in K_{60}} R_{60}(k) \ , \ R_{60}^{min} = \min_{k \in K_{60}} R_{60}(k)$$

K_{60} contains starting seconds for complete 60-second windows. Evaluate at one-second steps. The maximum is the best observed minute; the minimum is the lowest complete minute. No threshold verdict is assigned without a predeclared statistic, duration and acceptance threshold.

For audit metrics, this is defined as complete 1-second UTC bins with zero-filled missing intervals. Conversely, customer v1 incorporates partial boundary bins while excluding unusable arrival data. Ensure these distinct designations are maintained.

2 Verification Procedure

Hardware and OS Information

Machine Region	Workload Scenario	CPU and core allocation	RAM	OS and runtime	Swarms and tunnels
AWS ap-southeast-1	5 workload types across 10 runs	128 vCPU	247 GiB	Ubuntu 26.04	960 slots and 7,680 tunnels

2.1 Verification Overview

The documented server workflow checks retained evidence for 10 runs across 5 workload types, comprising 960 slots and 7,680 tunnels. A Python controller schedules the work in waves; within each wave, a global worker pool invokes the native Rust audit adapter independently for each tunnel. The adapter used the Sui supplied receipt admission, workload protocol and archive verification libraries. Figure 3 shows the operating design, not a new replay performed for this report.

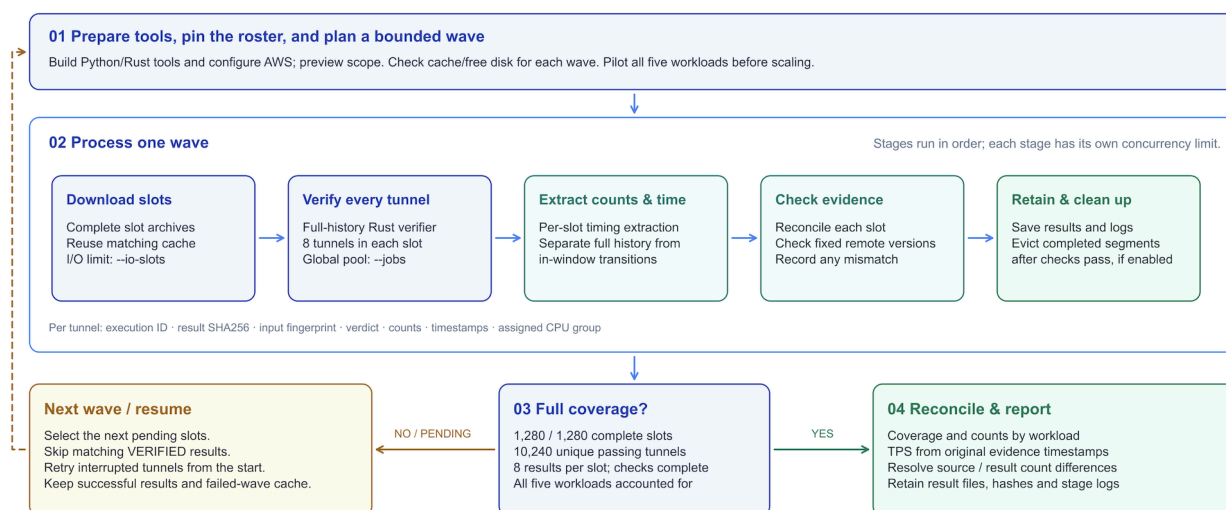


Figure 3. Independent verifier architecture

Controllers and workers. Batch runner in python controls scope, resource budgets and restart behavior. Its global pool spans multiple slots; each slot has eight tunnels. The scheduler changes parallel execution without changing the native Rust verification logic.

Checks and evidence. Per-tunnel outputs remain separate from the progress ledger and stage logs. The documented archive, signature, authorization and replay layers are explained in Section 2.3 states which mainnet results are supported by the retained evidence.

2.2 Full verification workflow

1. Script Preparation & Parameter Selection: Verify all 10 runs across 5 workload types, dependencies, and environment credentials before launch. Use target server dry runs to validate resource allocations and preview the processing plan across designated slots.

2. Execution Waves & Resource Allocation: Confirm disk availability, using the configured 16 GiB per-slot upper bound for conservative reservations, accounting for existing caches, and retaining 20 GiB of free space. Iterate through execution waves to fetch slot archives, execute native Rust verification, extract transaction counts, and validate temporal evidence against pinned remote version benchmarks. Completed segment caches may be purged following successful verification.

3. State Persistence & Execution Recovery: Store per execution attempts and immutable results under *tunnel-results/*, global status under *progress/*, compact slot timing evidence under *pilot/<slot>/*, and scheduler-stage logs under *batch-runs/*. If execution halts, completed *VERIFIED* tasks with matching inputs are skipped on resume, whereas interrupted tunnels restart from the initial state.

4. Reconciliation & Final Coverage Verification: Verify status via *audit_progress.py* to ensure all 7,680 executions across 960 slots are verified. Perform final cross checks on version integrity, event counts, and timing data, flagging any discrepancies.

CLI Execution Parameters Overview:

Control Flag	Functional Description
--slots	Specifies the overall slot processing scope for the execution.
--parallel-slots	Determines the number of slots active during a single processing wave.
--jobs	Sets the concurrency limit for simultaneous tunnel verifications.
--io-slots	Controls parallel I/O for downloading slots and extracting timing logs.
--cpus-per-worker	Allocates CPU cores per worker group (total job core demands must not exceed system capacity).

Throughput calculations strictly rely on valid in-window transaction metrics and original timestamp logs. Note that verifier execution duration, full history counts, and query ranges represent distinct parameters. Benchmarked performance metrics for 32, 64, or 128 worker threads on the target Linux host are not established in this documentation.

Full run script

This section is reserved for future updates. It will contain verified execution scripts alongside comprehensive guidelines covering hardware resource setup, roster selection, directory

structures, execution recovery procedures, and final validation checks. The explicit single command invocation for the complete campaign is currently unassigned.

Python

```
python run-batches.py \
  --slots 960 --parallel-slots 16 --jobs 128 \
  --cpus-per-worker 1 --io-slots 4 --cache-check-jobs 4\
  --max-slot-gib 16 --max-cache-gib 288 --reserve-free-gib 20 \
  --execute --evict-completed
```

2.3 Verification performed

All arithmetic and internal-consistency checks of the saved ledger and checkpoint index passed. The documented independent verifier explains the archive, authorization and replay checks. Its historical testnet evidence is kept separate from mainnet result coverage.

Check	Mainnet result and evidence boundary
Roster and final status	7,680/7,680 execution rows VERIFIED ; 960/960 slots fully verified. Unique execution and tunnel identifiers; eight executions per slot.
Receipt count reconciliation	15,241,348,297 admitted full-history receipts, and 15,241,348,297 in-window transactions. Every admitted full-history count matches its chain-declared count in the ledger.
Timestamp reconciliation	All execution opening and closing checkpoint numbers and millisecond timestamps agreed; 1,920 distinct transaction digests: 960 opening transactions and 960 closing transactions..
Archive integrity and coordinator checks	Documented verifier checks; final execution status is recorded. Detailed mainnet gate outputs and raw archive are outside this snapshot's coverage.
Receipt authorization and replay	Full-history admission counts and VERIFIED states are recorded for all executions. Individual signatures and deterministic transactions are not re-executed by this analysis.
Close state and payout effects	Receipt-count equality is reconciled. Final state roots, balances, nonces and recipient credit effects are not independently established.
Average rate arithmetic	Recomputed for all five runs and the combined campaign using exact counts and recorded outer-envelope timestamps.
Arrival peaks and physical timing	Supplemental peak figures are carried. Raw arrivals, clock accuracy and physical-time assurance are not established.

Note: Full history receipts were verified using chain-anchored tunnel information through corresponding workload protocol kernel. Authorization, deadline, state-continuity and resulting-state checks passed for all 7,680 executions. The TPS calculations were derived from the retained, hash-checked verifier outputs.

The ledger records *verifier_mode* = *current_local_build* and a verifier *SHA-256* identifier. It also records a result *SHA-256* and input fingerprint for each execution. These references identify recorded evidence; they do not by themselves demonstrate the content of a binary or a retained result file. All 960 slot cache states are *EVICTED*. Cache presence and verification status are explicitly separate in the ledger.

3 Performance Results

3.1 Operation and failure counts

Workload	Verified executions	In-window transactions	Full-history receipts
Blackjack	256	433,356,276	433,356,276
Tic tac toe	256	615,509,200	615,509,200
Flash	768	1,639,592,962	1,639,592,962
Simple payments	6,144	12,037,290,224	12,037,290,224
Battleship	512	515,599,635	515,599,635
Campaign total	7,680	15,241,348,297	15,241,348,297

Note: The campaign comprises 10 runs across 5 workload types: blackjack_v3, tic_tac_toe_v3, battleship_v3, simple_payments_v3, and flash_v1. Simple payments spans 768 slots across 5 runs, Flash spans 96 slots across 2 runs, and each remaining workload spans 32 slots in one run. Every slot contains eight executions. Comprehensive run identifiers are cataloged in the Data Availability Appendix.

The dataset incorporates 7,680 distinct execution IDs and 7,680 unique tunnel IDs, with every slot and execution successfully reaching a **VERIFIED** status. Throughout all executions, *full_history_admitted* matches *chain_declared_receipt_count* precisely, confirming complete internal reconciliation without signaling a fresh cryptographic replay across the full history.

Count or observation	Finding
Participant actions and authority events	Not separately classified in the ledger; only their qualifying combined count is reported.
Invalid or duplicate receipt records	Not quantified by the summary. Unique execution identifiers do not prove zero duplicate receipt attempts.
Attempted, accepted, rejected and failed attempts	Not assessed. Committed transactions do not supply all four attempt counters.
Production errors and restarts	Not assessed. Final VERIFIED state does not describe earlier attempts or producer restarts.

Here is the formula of number(N) of the units as effective off-chain transactions:

$$N_{attempted} = N_{accepted} + N_{rejected} + N_{failed}$$

Utilizing this relationship requires complete attempt level records paired with comprehensive, mutually exclusive outcome categories. Therefore, the absence of an explicit failure record or the achievement of a successful final verification status does not provide a sufficient basis for calculating a failure rate.

3.2 Aggregate rates

Both average metrics rely on the in-window transaction total as their numerator. Individual workload rates are derived using the specific counts outlined in Section 2.1.

Workload Scope	Play interval UTC	Play duration/s	Chain lifecycle interval UTC	Chain lifecycle duration/s
Blackjack	08:43:44.719 08:51:07.111	442.392	08:43:42.920 08:51:08.177	445.257
Tic tac toe	08:43:17.307 08:50:41.405	444.098	08:43:15.335 08:50:42.879	447.544
Flash/2 runs	08:44:00.315 08:52:34.960	514.645	08:43:57.951 08:52:36.619	518.668
Simple payments/5 runs	08:44:16.484 08:53:06.330	529.846	08:44:14.179 08:53:07.886	533.707
Battleship	08:41:53.862 08:49:04.663	430.801	08:41:51.539 08:49:06.156	434.617
Campaign	08:41:53.862 08:53:06.330	672.468	08:41:51.539 08:53:07.886	676.347

Note: Each interval list starts then ends on 7 October 2026. The calculator cross-checks chain timestamps against retained chain-times.json observations for 15,360 execution endpoint references. Closing uses chain_close_ms; settlement_event_ms differs in 5,552 executions by up to 210 ms and is not substituted.

Although the play windows overlap, they do not begin simultaneously. Consequently, the overall campaign timeframe encompasses intervals during which only a subset of workloads is actively running. Given that simple payments account for 78.98% of all committed in-window transactions, the aggregated average reflects this specific operational schedule and workload distribution rather than representing a workload-agnostic throughput capacity.

Campaign timing and additional measures

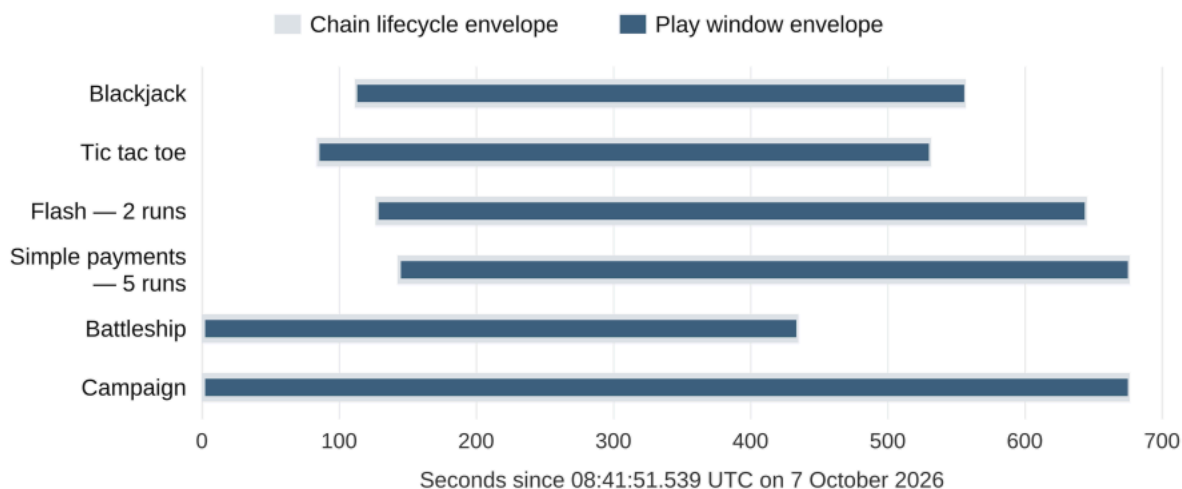


Figure 4. Outer timing envelopes derived from execution rows. Bars show recorded boundaries, not per-second throughput or continuous activity.

There are 960 distinct opening transaction digests and 960 distinct closing transaction digests, with no overlap between the sets. Each digest is referenced by 8 executions. This gives 1,920 distinct lifecycle transactions, rather than 15,360 independent opening and closing transactions.

Here are the more specific tunnels distributions per time since the campaign begins to play.

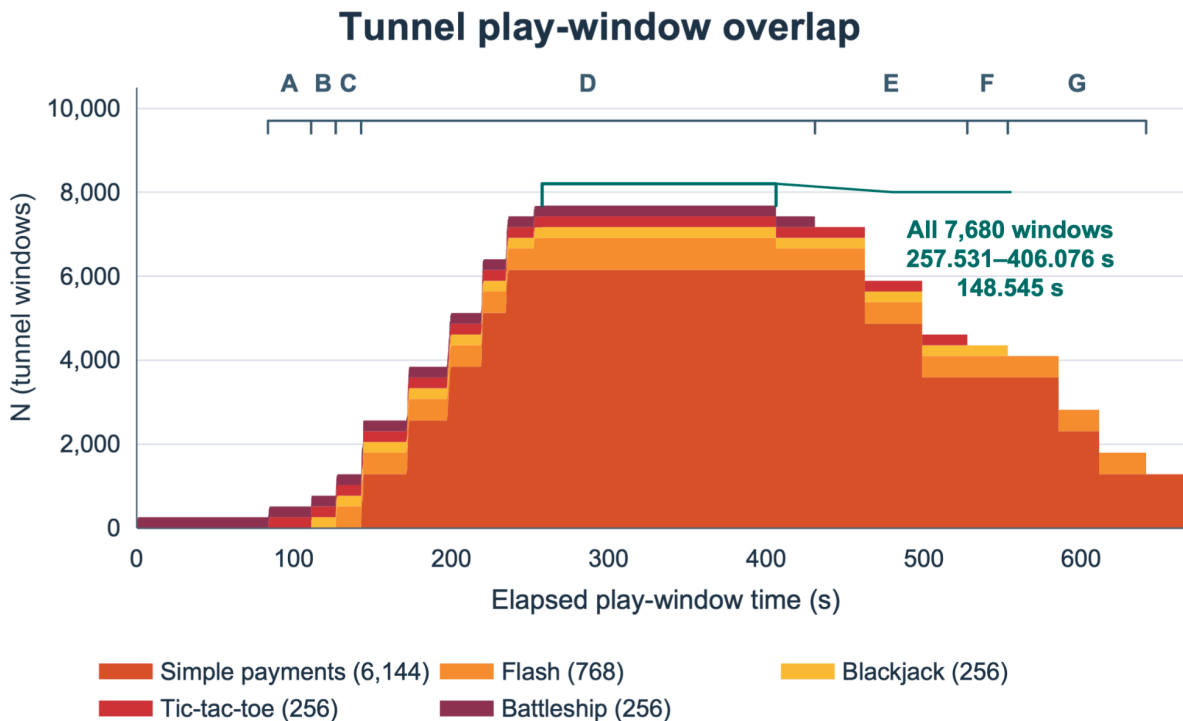


Figure 5. Recorded play window overlap. The green bracket marks all 7680 Tunnels. Time zero: 2026-10-07 08:41:53.862 UTC.

Figure 5 places time on the x axis and N, the number of tunnel executions covered by open recorded play windows, on the y axis. Colored step bands show the contribution of each workload. Their combined height gives total tunnel window overlap.

$N(t) = 8 \times$ the number of slots with $\text{start} \leq t < \text{end}$. Each listed workload has at least one open window. The strict journal marker rule for qualifying transactions in Section 1.1 is unchanged.

A–G identifies the overlap spans below, measured from 08:41:53.862 UTC on 7 October 2026. All five workload types overlap for 288.179 seconds (D). Within D, all 7,680 tunnel windows overlap for 148.545 seconds, from 08:46:11.393 to 08:48:39.938 UTC.

Mark	Overlapping workloads	Span (seconds)	Duration (s)
A	Battleship, Tic-tac-toe	83.445–110.857	27.412
B	Battleship, Tic-tac-toe, Blackjack	110.857–126.453	15.596
C	Battleship, Tic-tac-toe, Blackjack, Flash	126.453–142.622	16.169
D	All five workloads	142.622–430.801	288.179
E	Tic-tac-toe, Blackjack, Flash, Payments	430.801–527.543	96.742
F	Blackjack, Flash, Payments	527.543–553.249	25.706
G	Flash, Payments	553.249–641.098	87.849

Note: Spans include their start and exclude their end. Window overlap does not establish instantaneous TPS, receipt arrival times or simultaneous CPU activity; clock accuracy and cross-machine skew remain unassessed.

A supplementary on-chain checkpoint transaction rate is not assessed. The endpoint index alone does not establish the predecessor checkpoint time, an exhaustive successful target transaction population or all transaction effects required by that metric. Dividing off-chain transactions by lifecycle time must not be quoted as on-chain TPS.

Worker launch and final report timestamps are not in the ledger, so worker lifecycle TPS is not assessed. Execution *wall_seconds* records verification work and can overlap across workers; neither its sum nor a per-execution replay duration is a production TPS denominator.

$$R_{sum} = \sum_{i=1}^{960} \frac{N_i}{D_i} = 40,108,985.34 \text{ transactions/s}$$

This diagnostic sums local rates for 960 slots without per-slot flooring. Each slot count sums its eight executions; its duration uses the shared play endpoints. Unequal intervals make this distinct from campaign or sustained throughput.

The available records support a single mixed-workload campaign with one run per workload. They do not support a repeated-run spread, latency percentiles, a resource saturation profile or a statistical confidence interval for physical timing.

3.3 Peak and sustained TPS results

The following values are taken from the final summary and cross-checked by recomputing from the retained arrival-second counts. All 10 runs across 5 workload types, together with the campaign, are labelled COMPLETE, with zero missing or dummy arrival timestamps. These are calculation coverage labels; all 7,680 executions separately retain VERIFIED ledger status.

Supplemental arrival metrics

Scope and convention	Peak 1s	60s maximum
Campaign customer v1	40,682,328.00	40,339,143.65
Campaign available-arrival complete-second metric	40,682,328.00	40,339,143.65

It reports the same campaign values under customer v1 and available-arrival supplemental conventions. Numerical equality does not make their boundary rules identical. Strict metrics exclude incomplete boundary seconds, containing 1,984,404 transactions, while customer v1 includes valid boundary arrival buckets. This boundary exclusion does not represent missing arrival evidence.

The strict calculation evaluates **672 complete UTC-aligned seconds** and **613 consecutive 60-second windows**, advancing one second at a time.

Strict Metric	Winning Interval UTC:2026-10-07	TPS
Peak complete second	08:46:17–08:46:18	40,682,328.00
Maximum 60-second average	08:46:07–08:47:07	40,339,143.65
Minimum 60-second average	08:42:17–08:43:17	1,210,056.52

The maximum 60-second window contains **2,420,348,619 qualifying transactions**. The minimum is a separate diagnostic across the staggered campaign schedule; it is not the customer v1 sustained-peak metric or a certified baseline.

These results describe off-chain tunnel throughput under the retained signed arrival-time evidence. They do not independently establish physical clock accuracy, arbitrary millisecond-aligned peak throughput, simultaneous CPU activity, or Sui on-chain TPS. The sustained-peak figure is the **highest observed 60-second average**, not a minimum throughput guarantee.

Appendix

Data Summary

Off-chain test scenarios were executed and supplied directly by the Sui team. Documented entries below delineate preserved high-level data overview for the champion. File paths remain relative to the primary directory, where SHA-256 digests verify analyzed bytes while recorded verifier and reconciliation hashes represent distinct provenance attributes.

E1 verification-ledger.json

7,680 execution records across 960 slots

SHA-256 f7ca593f866304ad6336e247304c185f06cee9077f0342d54bcb7cd6deda2d0

E2 chain-times.json

1,920 transaction checkpoint observations; observed-fullnode time basis.

SHA-256 259ef6da73fa1d20c2dd324428cf6161569795d00c610b2f24f5d8354431086c

E3 TPS_calculation_original.md

SHA-256 4bb0763b1d3498c0cda2701a541bae226c7da9d1e71f142587e1046430570dd8

Mainnet run identifiers

Index/UTC	Slots	Reports	Run identity
08:41:56	32	1	0c6b0ab3c1cb7ddd9a6b14d840f5df618cf931123f336eb9fd0c884832bcd83e
08:43:19	32	1	7c38eb495b556fff4d8bc9d327e3a488b3fef114b27156dd144a0513aaf17719
08:43:46	32	1	95b6aeb6dc97a9d65d0b00e1dc209fc75ddff4a36ee5224cfb0bfcc5784752e3
08:44:00	64	2	bdd93b874dc9bf73b62ed5f98f19f515f1ca06f4ad5f8875bc343ca24b666fd1
08:44:19	160	5	f42fa89d478ce8d551d3f623419fae178cae67a9fb5f79c9e034b77bed6d0a9f
08:44:49	160	5	2b864fccd8cf73e09692f1793390afdec65e9c3ab4388a7527a72a6ed75944e2
08:45:13	160	5	79e82b8bb8cdc484732ba41f3d6a1e675ad2dbc71a4dcb7853ec2a7c9351d6f2

08:45:35	160	5	a6f79eb9402d39353765381cb9a88268861c92aa03694d4de28fb4024b8969f1
08:45:50	128	4	b4c8ab0e70a4f82df944c4852155f978ee7de18cb19688e9525191d86f62e97d
08:46:09	32	1	0adc68328df4fa6fa12b2551d413d4b425dd035286ad9d49b621a1eb2b875afe

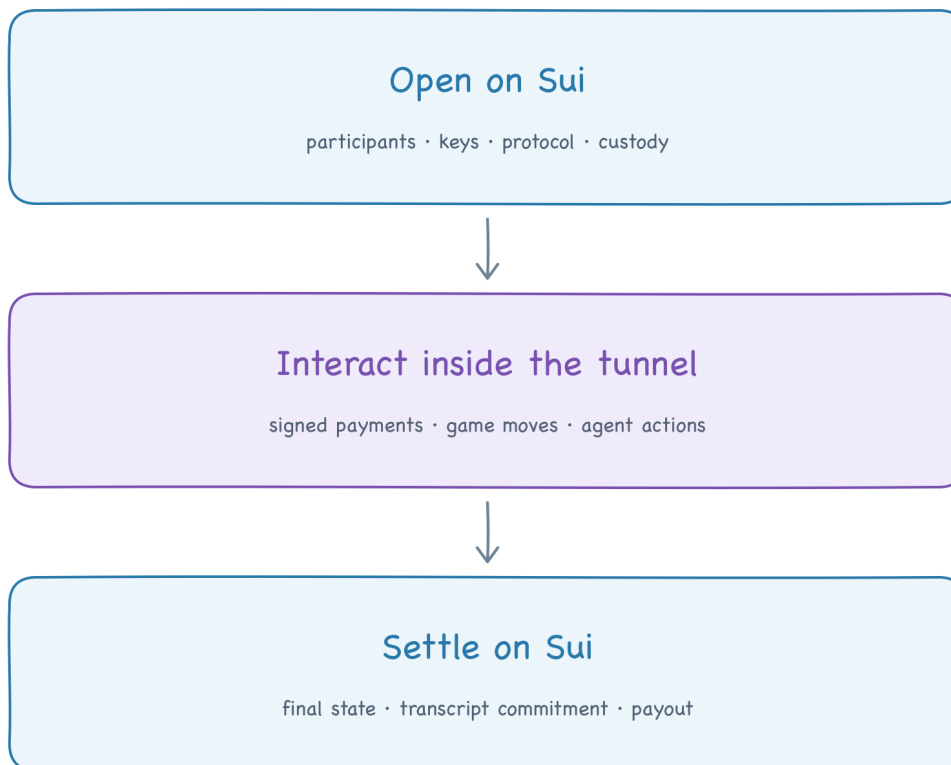
Sui Tunnel Introduction

off-chain execution, on-chain trust

A tunnel lets a small group execute many application actions without sending every action to on-chain. Sui establishes the boundaries: when the tunnel opens, it records the participants, authorization keys, protocol, initial state commitment and custody terms. Inside the tunnel, participants exchange signed transition receipts representing payments, game moves or agent actions. When the tunnel closes, Sui records the agreed final commitment, outcome and settlement.

The intermediate actions remain off-chain, but they remain verifiable. An auditor can replay the signed receipts through the application's protocol rules, rebuild the final state and transcript root, and compare them with the on-chain settlement record. This reduces L1 data and per-action

gas while keeping entry, custody and final settlement anchored to Sui.



Reduce L1 load, but opening and settlement still converge on Sui

Although tunnels remove the need to submit every intermediate action on-chain, they do not make L1 performance irrelevant. Large numbers of tunnels still need to open, fund and settle within a practical amount of time. If the base layer is slow, entry and exit become the bottleneck even when the transition inside the tunnel is fast.

Across the five audited payments runs, **6,144 tunnels** were created through **768 shared create/open PTBs** and settled through **768 shared closing PTBs**, with eight executions referencing each PTB. Within individual runs, observed opening checkpoint timestamps spanned **1.443–2.382 seconds**, while closing checkpoint timestamps spanned **0.885–1.419 seconds**.

Across all five payment runs combined, opening timestamps spanned **93.440 seconds** and closing timestamps spanned **211.055 seconds**. These combined spans reflect the staged execution schedule, rather than a single submission burst.

Across the complete campaign of **10 runs covering 5 workload types**, **7,680 tunnels** were covered by **960 shared create/open PTBs** and **960 shared closing PTBs**.

Beyond payment channels

Tunnels belong to the same state channel family as Bitcoin Lightning, but they target a wider application model. Lightning is optimized for payments and payment routing. A programmable tunnel can carry application specific state transitions, such as payments, poker moves or agent actions, while settling the final commitment and value transfer on Sui.

Tunnels can be combined with additional mechanisms where an application requires them. TEEs or zero-knowledge proofs can strengthen privacy or execution assurance, while MPC, verifiable randomness or other application-specific methods can support games and agent coordination. These mechanisms extend the tunnel design but were outside the scope of this verification.

Sui acts less like a processor for every interaction and more like the shared registry, escrow and final court of settlement. The parties agree on the rules when they enter, interact directly under those rules, and return to Sui with the final result or a dispute that the on-chain policy can handle.

Elevate Your Web3 Journey

CertiK is the largest Web3 security platform combining formal verification
with audits and comprehensive security solutions.

Performance Verification Report | Copyright © CertiK

