FORMAL SPECIFICATION AND TESTING OF QUIC

Name : Raghav Gade
Roll no. : 20CS02003
Supervisor : Dr. Srinivas Pinisetty
School of Electrical and Computer Sciences

MOTIVATION

TCP & EVOLUTION

Widely used for decades, but struggles with network latency, congestion, and slow evolution due to being tightly integrated with operating systems.

WHY QUIC?

Designed to address TCP's limitations, QUIC brings faster connection setup, improved security, and flexible protocol evolution directly at the application layer.

It will serve as a basis to HTTP/3.

MOTIVATION

NEED FOR PROTOCOL VERIFICATION

Informal specifications and interoperability testing alone can miss subtle and critical errors. Formal verification is needed to ensure protocol correctness under all scenarios.

LIMITS OF INTEROPERABILITY

Even if implementations work together, they may share the same bugs. Formal specification and model checking can uncover flaws that interoperability tests cannot detect.

RECAP

LITERATURE SURVERY

TAKING A LONG LOOK AT QUIC (2017)

by M. Kakhki, S. Jero

Performance evaluation and discusses performance comparisons between TCP and QUIC.

FORMAL SPECIFICATION AND TESTING OF QUIC (2019)

by K. L. McMillan and L. D. Zuck,
Verification of QUIC -draft 18 was done by using a
methodology called "Network-centric Compositional testing" using IVy.

VERIFYING QUIC IMPLEMENTATIONS USING IVY(2021)

by C. Crochet, T. Rousseaux

Extension to McMillan’s work by using to same methodology to produce formal model for draft-29 of QUIC

FORMAL ANALYSIS OF QUIC HANDSHAKE PROTOCOL USING SYMBOLIC MODEL CHECKING (2021)

by J. Zhang, X. Gao,

Security analysis of the QUIC handshake protocol based on symbolic model checking using ProVerif and Verifpal

RFC 9000 PROPERTIES

MUST/SHOULD/MAY

MUST

MUST statements define mandatory requirements.
"A QUIC endpoint MUST NOT reuse a connection ID used on one network path on a different path."
- RFC 9000, Section 5.2

This is a strict rule. Endpoints are forbidden from reusing connection IDs across different network paths to maintain privacy and security.

SHOULD

SHOULD statements indicate recommended best practices.
"Endpoints SHOULD pad UDP datagrams containing Initial packets to at least the smallest allowed maximum datagram size."
- RFC 9000, Section 14.1

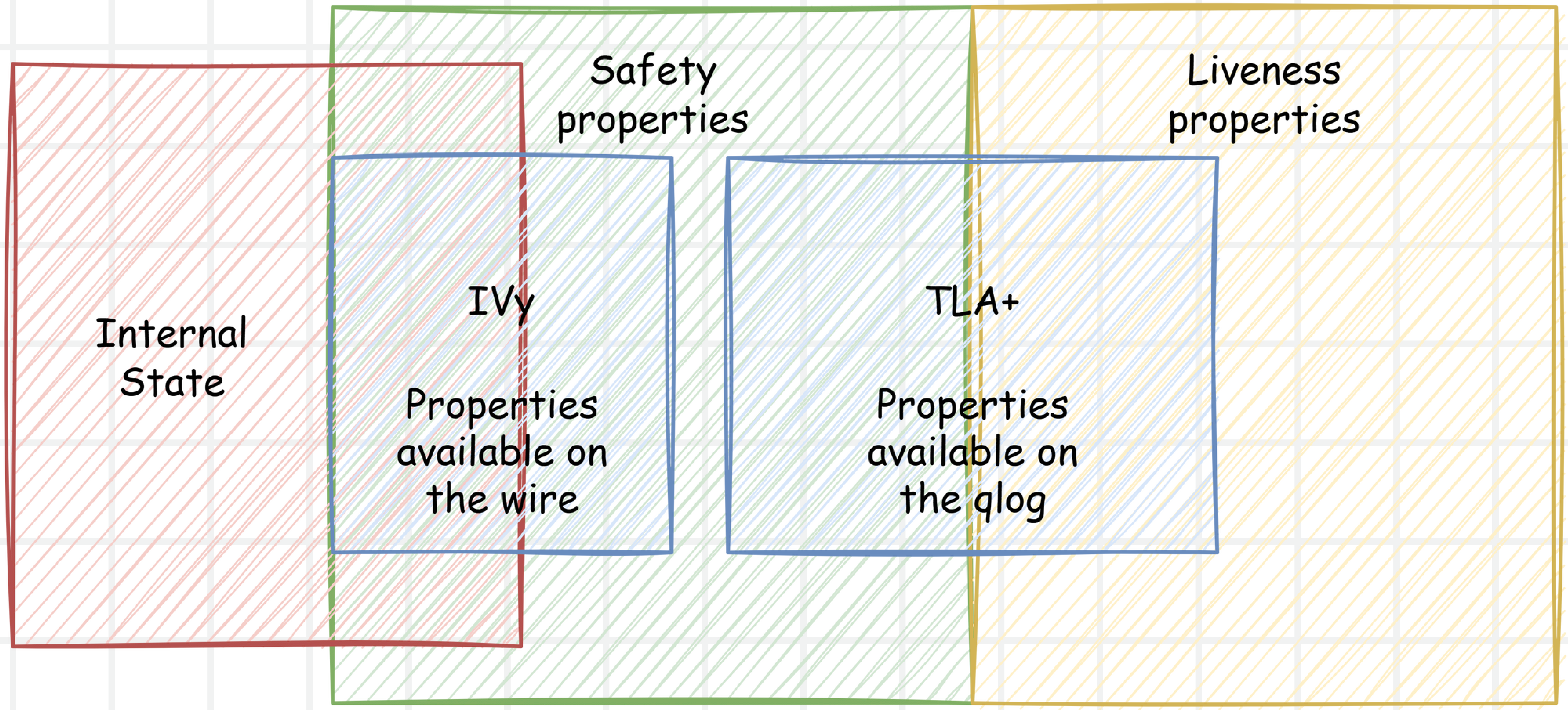
This is strongly recommended to improve network compatibility and prevent information leakage, but may be omitted for special situations.

MAY

MAY statements describe optional or permitted behaviors.
"An endpoint MAY change the connection ID it uses for a peer at any time."
- RFC 9000, Section 5.1.1

This is an optional action; endpoints have the freedom to change the connection ID during a connection if they choose.

PROPERTY CATEGORIES



PROPERTY CATEGORIES

INTERNAL STATE	Describe variables and statuses maintained by a QUIC implementation, such as the current list of open streams, flow control counters, encryption levels, or the packet number space.
SAFETY	They assert that "nothing bad happens." Ensure that the protocol never reaches undesirable or forbidden states, such as delivering data out of order, duplicate consumption, or violating flow control.
LIVENESS	Liveness properties guarantee that "something good eventually happens." In QUIC, this means progress properties like eventual handshake completion or stream closure, ensuring forward progress in communication.

IVY VS TLA+

IVy

- Ivy is focused mainly on safety properties and refines protocols without explicit temporal logic.
- Ivy emphasizes protocol synthesis and stepwise refinement from high-level abstract rules to implementations.
- Ivy as used in McMillan's work uses executable specifications and state matching to validate that real-world protocol executions conform to the formal model.

TLA +

- TLA+ supports temporal logic operators and is suited for expressing both safety and liveness properties.
- TLA+ is tailored toward exhaustive state-space exploration and general system modeling..
- TLA+ leverages its temporal logic for richer temporal property checking of observed protocol traces.

QLOG

- qlog is a standardized event logging format maintained by the IETF(draft-ietf-quic-qlog-main-schema-11), designed specifically for modern transport protocols like QUIC and HTTP/3. It captures protocol behavior and events in a JSON-based structure. Supported by aioquic, picoquic, quiche, etc.
- qlog stores detailed, structured records of protocol-level events such as -
  - Timing information for connections and streams
  - Packet send/receive events
  - Loss and recovery, congestion control
  - Version negotiation, handshake, and 0-RTT/1-RTT transitions
  - Stream-level data (open/close, data, flow control etc.)
  - Error and warning events
- qlog does not capture all possible internal state or cryptographic information, nor does it record every possible transient variable in a QUIC implementation.

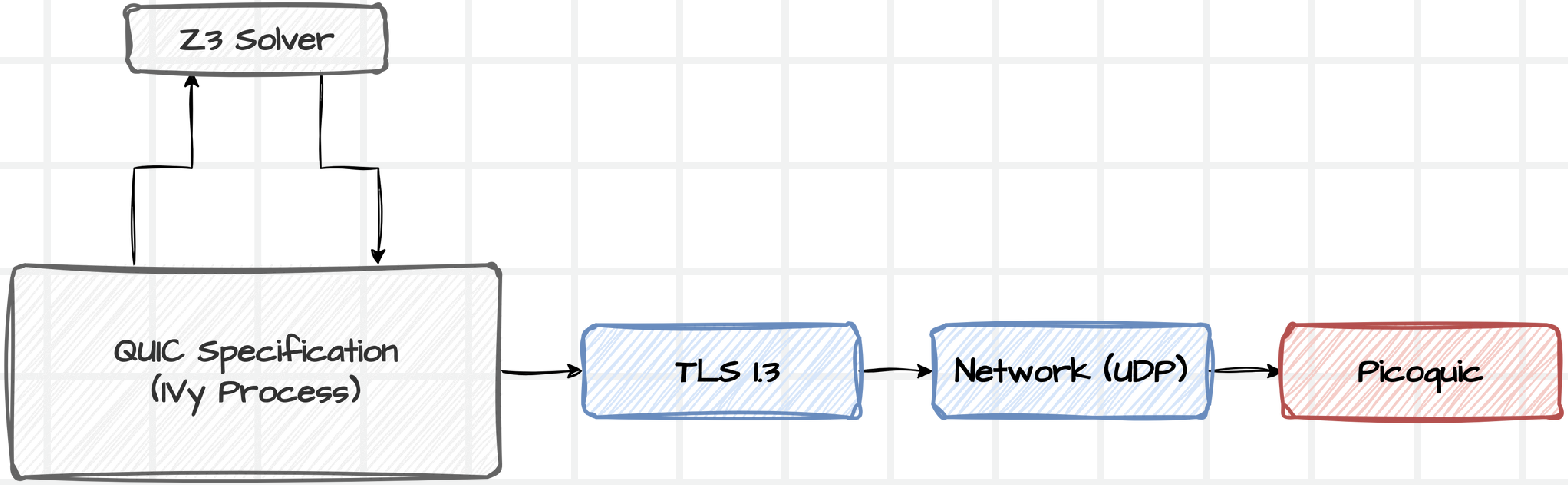
QLOG TO TLA+ CONSTANT

```
{
  "data": {
    "frames": [
      {
        "frame_type": "crypto",
        "length": 476,
        "offset": 0
      }
    ],
    "header": {
      "packet_number": 0,
      "packet_type": "initial",
      "dcid": "fcabe9223fe7d8c1",
      "scid": "f17cedeb9e99784e"
    },
    "raw": {
      "length": 524
    }
  },
  "name": "transport:packet_received",
  "time": 1745672395159.7234
},
```

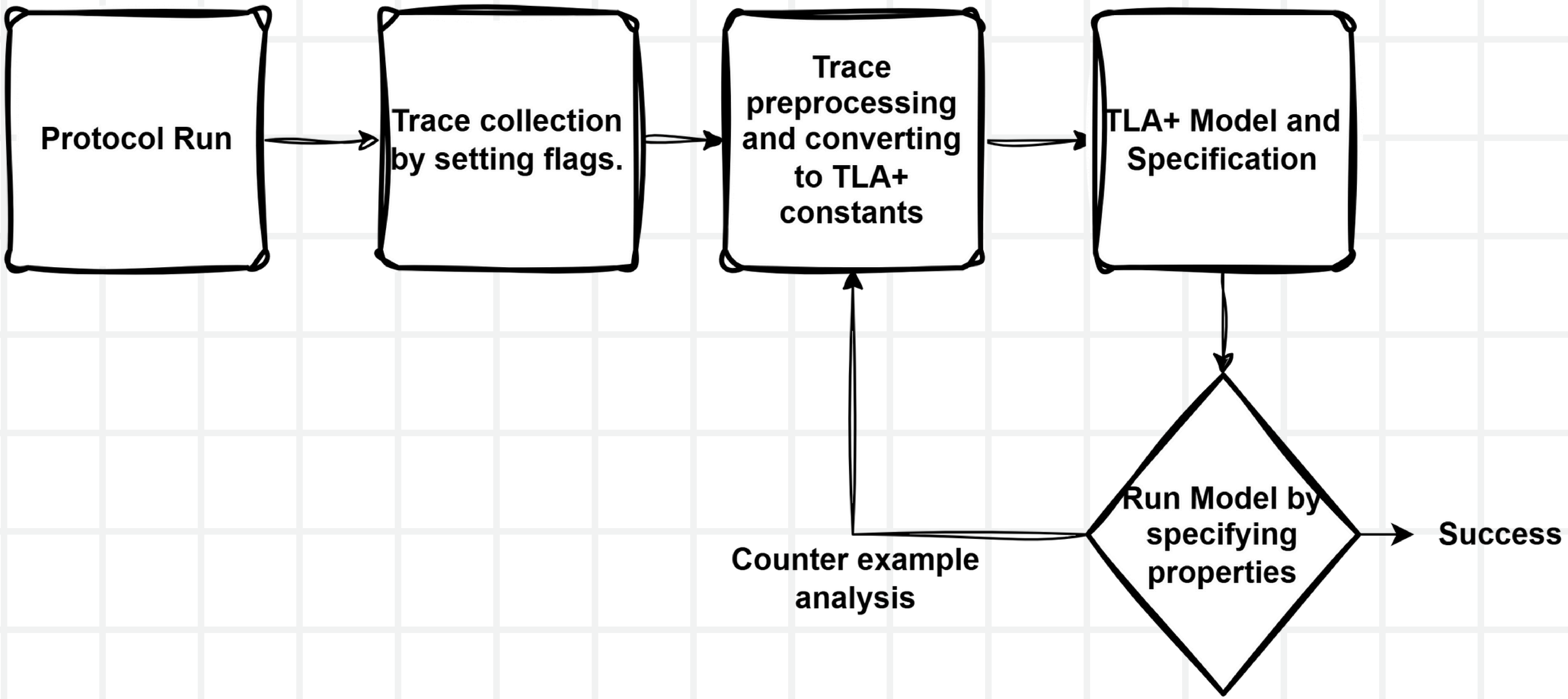
```
[
  data |-> [
    frames |-> <<
      [
        frame_type |-> "crypto",
        length |-> 476,
        offset |-> 0
      ]
    >>,
    header |-> [
      packet_number |-> 0,
      packet_type |-> "initial",
      dcid |-> "fcabe9223fe7d8c1",
      scid |-> "f17cedeb9e99784e"
    ],
    raw |-> [
      length |-> 524
    ]
  ],
  name |-> "transport:packet_received",
  time |-> 17456723951597234
],
```

METHODOLOGY

FORMAL SPECIFICATION IN IVY



WORKFLOW


Protocol Run
Trace collection by setting flags.
Trace preprocessing and converting to TLA+ constants
TLA+ Model and Specification
Run Model by specifying properties
Success
Counter example analysis

SAFETY

- Packet numbers in sent packets must always increase.
- Meaning: Enforces strict packet number ordering as required by QUIC.
- RFC 9000 reference: Section 12.3: ”Packet numbers MUST be assigned in increasing order, beginning with the value 0 for the first packet sent on each connection.”

```
RECURSIVE CollectSentIndices(_,_)
    CollectSentIndices (trace, upto) ==
        IF upto = 0 THEN <<>>
        ELSE
            LET tail == CollectSentIndices (trace, upto-1)
            IN IF IsPacketSent (trace[upto])
            THEN tail \o << upto >>
            ELSE tail

Safety_PktNosMonotonic ==
    LET
        cSentSeq == CollectSentIndices (ClientTrace, ci-1)
        cLen == Len(cSentSeq)
        sSentSeq == CollectSentIndices (ServerTrace, si-1)
        sLen == Len(sSentSeq)
    IN
        (cLen < 2 \/ (\A k \in 2..cLen :
        ClientTrace[cSentSeq[k]].data.header.packet_number >
        ClientTrace [cSentSeq[k-1]].data.header.packet_number))
        /\
        (sLen < 2 \/ (\A k \in 2..sLen :
        ServerTrace[sSentSeq[k]].data.header.packet_number >
        ServerTrace[sSentSeq [k-1]].data.header.packet_number))
```

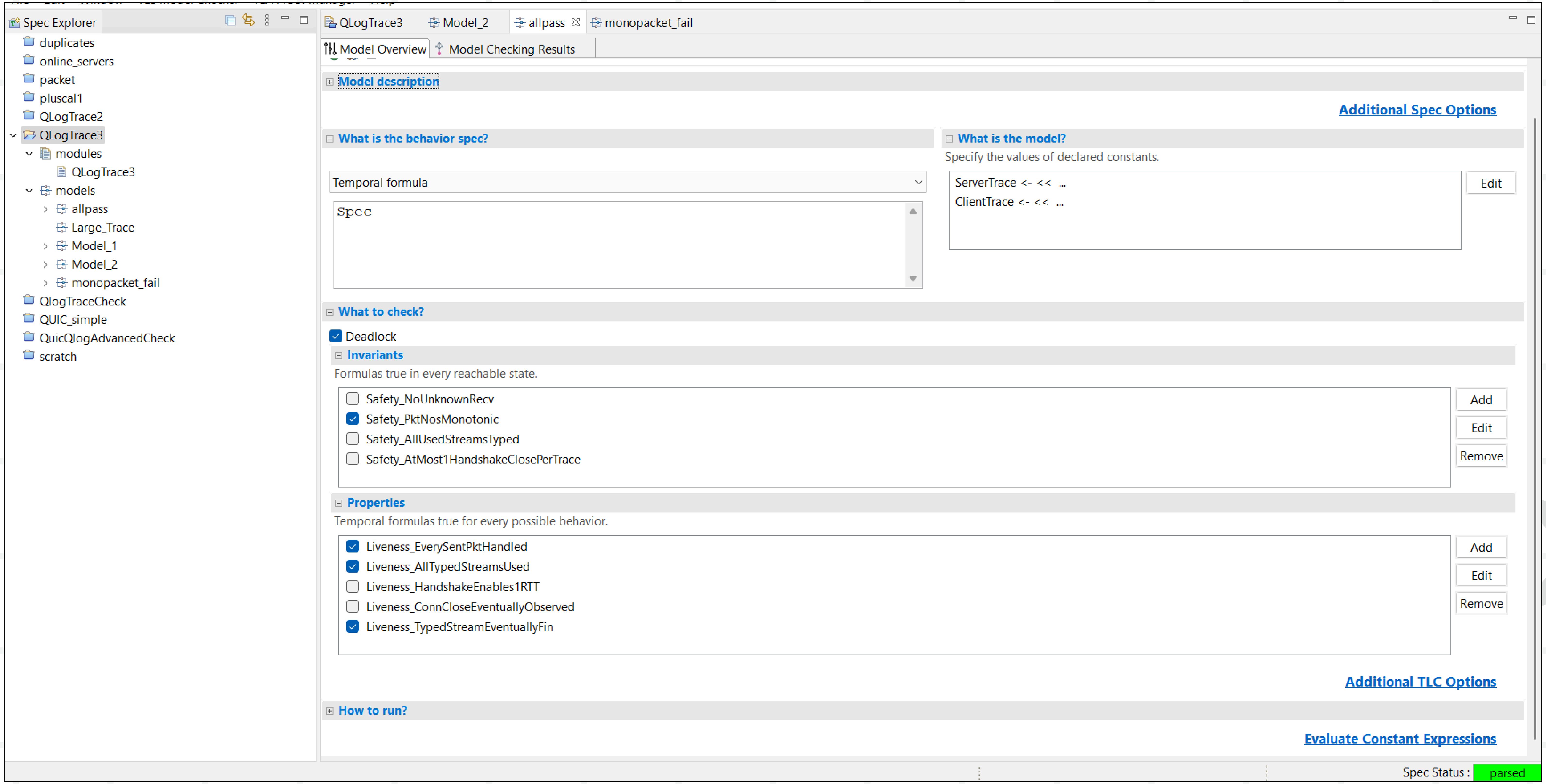
LIVENESS

- Every sent packet must eventually be either received or dropped by the other endpoint.
- Meaning: Ensures no sent packet is forever lost/unaccounted for in the logs.
- RFC 9000 reference: Section 12.2/12.3: ”Receivers MUST track and acknowledge all packets received.”

```
Liveness_EverySentPktHandled ==
  SentPktNums(ClientTrace, ci-1)
  \subseteq (RecvPktNums(ServerTrace, si-1)
  \cup DropPktNums(ServerTrace, si-1)) /\
  SentPktNums(ServerTrace, si-1)
  \subseteq (RecvPktNums(ClientTrace, ci-1)
  \cup DropPktNums(ClientTrace, ci-1))
```

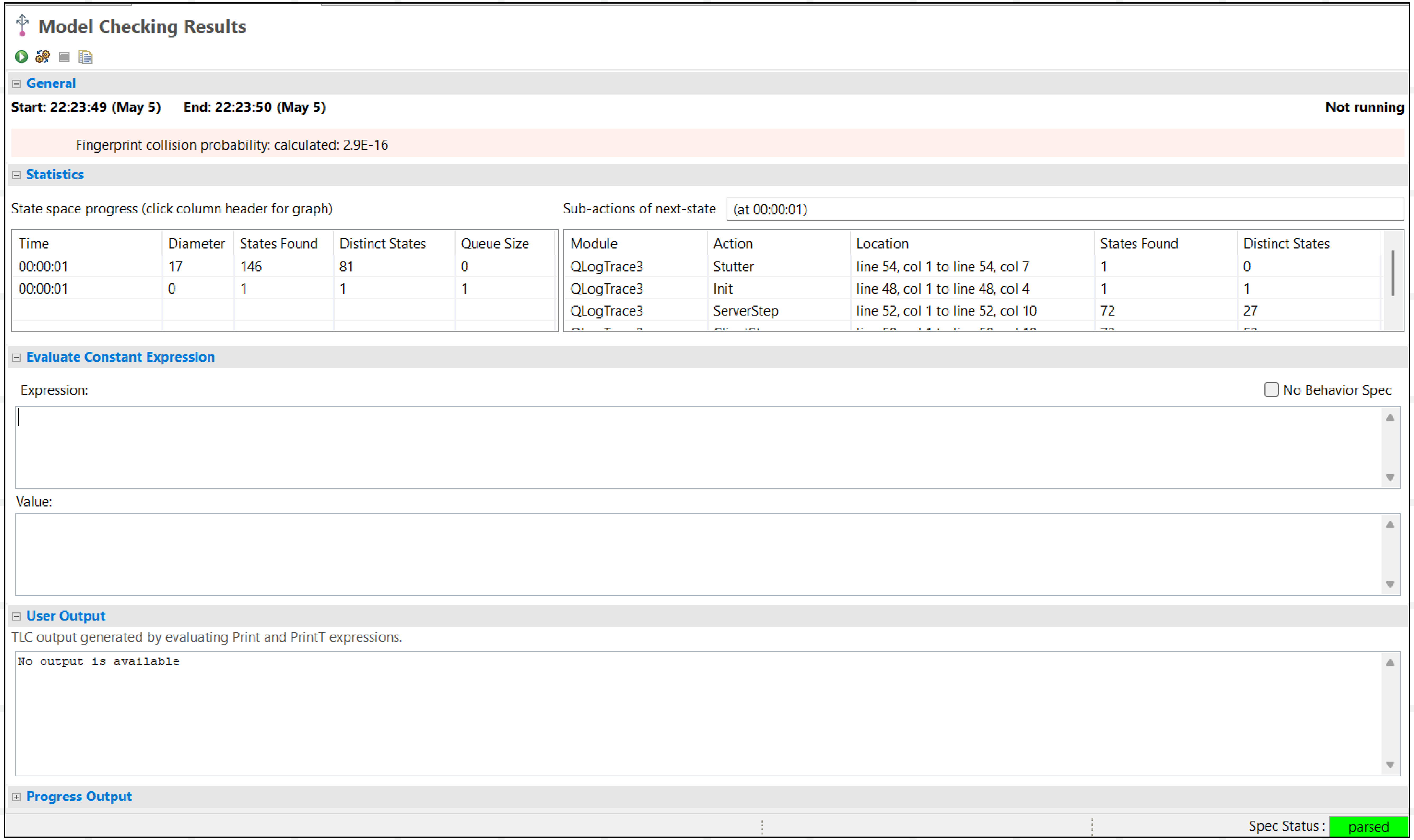
```
Liveness_HandshakeEnables1RTT ==
  \A t \in {"Client", "Server"}:
    LET
      trace == IF t = "Client"
      THEN ClientTrace
      ELSE ServerTrace
      hsidx == IF HandshakeSent(trace, Len(trace)) # {}
      THEN Min(HandshakeSent(trace, Len(trace))) ELSE Len(trace) + 1
      has1RTT == \E i \in SentEvents(trace, Len(trace)):
      i > hsidx /\ trace[i].data.header.packet_type = "1RTT"
    IN hsidx <= Len(trace) => has1RTT
```

TOOLBOX

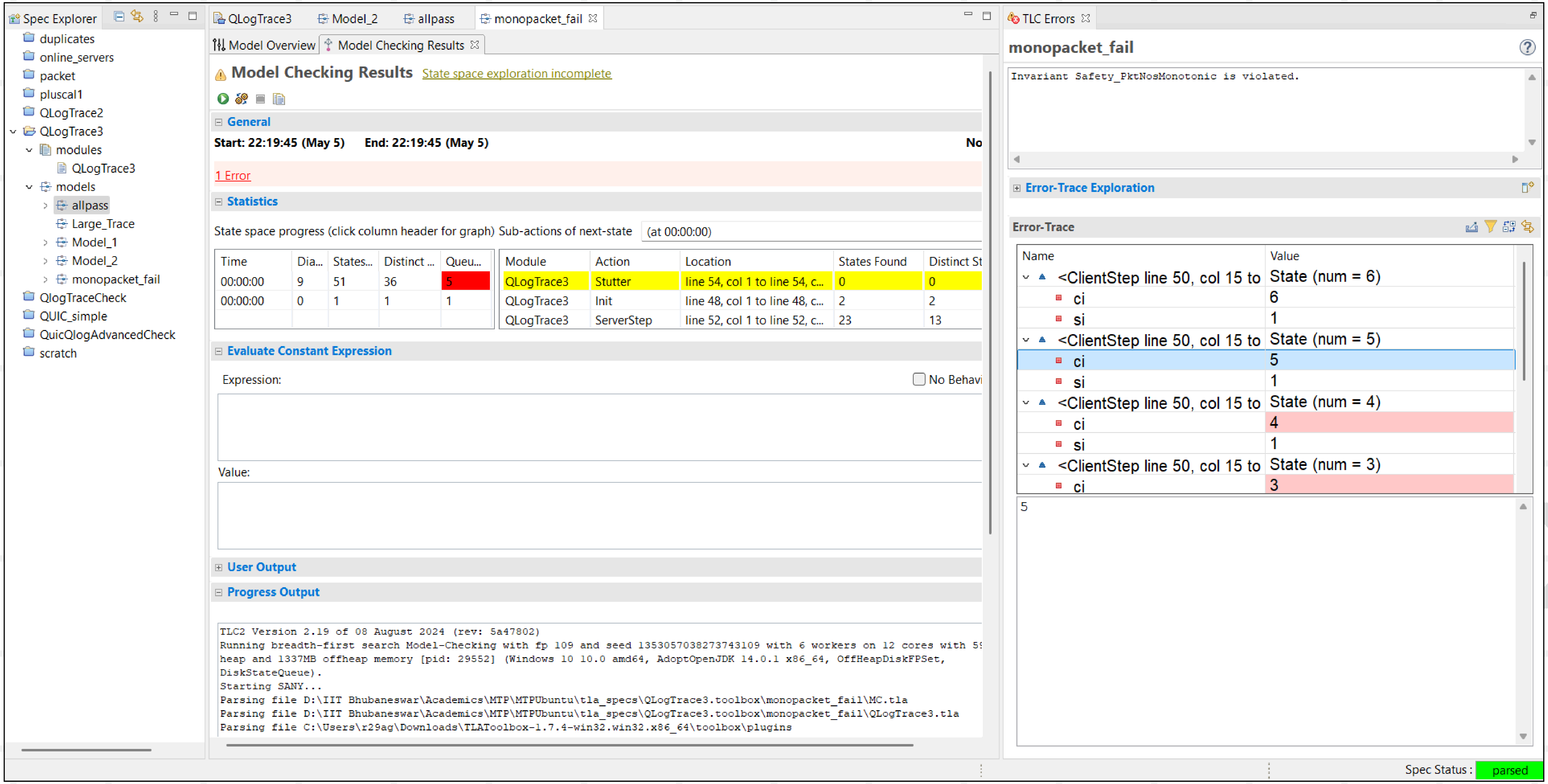


Spec Explorer
duplicates
online_servers
packet
pluscal1
QLogTrace2
QLogTrace3
modules
QLogTrace3
models
allpass
Large_Trace
Model_1
Model_2
monopacket_fail
QlogTraceCheck
QUIC_simple
QuicQlogAdvancedCheck
scratch
QLogTrace3
Model_2
allpass
monopacket_fail
Model Overview
Model Checking Results
Model description
Additional Spec Options
What is the behavior spec?
Temporal formula
Spec
What is the model?
Specify the values of declared constants.
ServerTrace <- << ...
ClientTrace <- << ...
Edit
What to check?
Deadlock
Invariants
Formulas true in every reachable state.
Safety_NoUnknownRecv
Safety_PktNosMonotonic
Safety_AllUsedStreamsTyped
Safety_AtMost1HandshakeClosePerTrace
Add
Edit
Remove
Properties
Temporal formulas true for every possible behavior.
Liveness_EverySentPktHandled
Liveness_AllTypedStreamsUsed
Liveness_HandshakeEnables1RTT
Liveness_ConnCloseEventuallyObserved
Liveness_TypedStreamEventuallyFin
Add
Edit
Remove
Additional TLC Options
How to run?
Evaluate Constant Expressions
Spec Status :
parsed

MODEL RESULTS



ERROR TRACE


Spec Explorer
duplicates
online_servers
packet
pluscal1
QLogTrace2
QLogTrace3
modules
QLogTrace3
models
allpass
Large_Trace
Model_1
Model_2
monopacket_fail
QlogTraceCheck
QUIC_simple
QuicQlogAdvancedCheck
scratch
QLogTrace3
Model_2
allpass
monopacket_fail
Model Overview
Model Checking Results
Model Checking Results
State space exploration incomplete
General
Start: 22:19:45 (May 5)
End: 22:19:45 (May 5)
1 Error
Statistics
State space progress (click column header for graph)
Sub-actions of next-state
(at 00:00:00)
Time
Dia...
States...
Distinct ...
Queu...
00:00:00 9 51 36 5
00:00:00 0 1 1 1
Module
Action
Location
States Found
QLogTrace3 Stutter line 54, col 1 to line 54, c... 0 0
QLogTrace3 Init line 48, col 1 to line 48, c... 2 2
QLogTrace3 ServerStep line 52, col 1 to line 52, c... 23 13
Evaluate Constant Expression
Expression:
Value:
User Output
Progress Output
TLC2 Version 2.19 of 08 August 2024 (rev: 5a47802)
Running breadth-first search Model-Checking with fp 109 and seed 1353057038273743109 with 6 workers on 12 cores with
heap and 1337MB offheap memory [pid: 29552] (Windows 10 10.0 amd64, AdoptOpenJDK 14.0.1 x86_64, OffHeapDiskFPSet,
DiskStateQueue).
Starting SANY...
Parsing file D:\IIT Bhubaneswar\Academics\MTP\MTPUbuntu\tla_specs\QLogTrace3.toolbox\monopacket_fail\MC.tla
Parsing file D:\IIT Bhubaneswar\Academics\MTP\MTPUbuntu\tla_specs\QLogTrace3.toolbox\monopacket_fail\QLogTrace3.tla
Parsing file C:\Users\r29ag\Downloads\TLAToolbox-1.7.4-win32.win32.x86_64\toolbox\plugins
TLC Errors
monopacket_fail
Invariant Safety_PktNosMonotonic is violated.
Error-Trace Exploration
Error-Trace
Name
Value
<ClientStep line 50, col 15 to
State (num = 6)
ci 6
si 1
<ClientStep line 50, col 15 to
State (num = 5)
ci 5
si 1
<ClientStep line 50, col 15 to
State (num = 4)
ci 4
si 1
<ClientStep line 50, col 15 to
State (num = 3)
ci 3
5
Spec Status :
parsed

RESULTS AND COMPARISONS

- Traditional protocol verification often relied on informal descriptions, interoperability and abstract checks, missing edge cases.
- TLA+ enables trace-based validation directly from qlog data, allowing us to check both safety and liveness properties as specified in RFC 9000.
- While IVY is strong at synthesizing protocol implementations and discovering invariants, it lacks TLA+’s ability to systematically evaluate end-to-end behavior over time, particularly critical liveness guarantees.

CONCLUSION

COMPUTATIONALLY EXPENSIVE	Using raw traces from implementations often led to TLC running out of memory or becoming unresponsive, even on systems with 8GB or more of RAM.
MORE EXAMPLES	Need more traces to generate more counter examples and make specification robust and identify any potential errors.
MORE SPECIFICATIONS	Specifications currently included don't completely specify the protocol. The specification can be extended to included more properties.

REFERENCES

- K. L. McMillan and L. D. Zuck, “Formal specification and testing of quic,” in Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM ’19). New York, NY, USA: Association for Computing Machinery,2019, pp. 227–240. [Online]. Available: https://doi.org/10.1145/3341302.3342087
- C. Crochet, T. Rousseaux, M. Piraux, J.-F. Sambon, and A. Legay, “Verifying quic implementations using ivy,” in Proceedings of the 2021 Workshop on Evolution, Performance and Interoperability of QUIC (EPIQ ’21). New York, NY, USA: Association for Computing Machinery, December 2021, pp. 35–41. [Online]. Available: https://doi.org/10.1145/3488660.3493803
- A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, “Taking a long look at quic: An approach for rigorous evaluation of rapidly evolving transport protocols,” in Proceedings of the 2017 Internet Measurement Conference (IMC ’17). New York, NY, USA: Association for Computing Machinery, November 2017, pp. 290–303. [Online]. Available: https://doi.org/10.1145/3131365.3131368
- J. Zhang, X. Gao, L. Yang, T. Feng, D. Li, and Q. Wang, “A systematic approach to formal analysis of quic handshake protocol using symbolic model checking,” Security and Communication Networks, vol. 2021, pp. 1–13, August 2021, academic Editor: Ruhul Amin. [Online]. Available: https://doi.org/10.1155/2021/1630223
- X. Zhang, S. Jin, Y. He, A. Hassan, Z. M. Mao, F. Qian, and Z.-L. Zhang, “Quic is not quick enough over fast internet,” in Proceedings of the ACM Web Conference 2024 (WWW ’24). New York, NY, USA: Association for Computing Machinery, May 2024, pp. 2713–2722. [Online]. Available: https://doi.org/10.1145/3589334.3645323

THANK YOU