

A Framework for AI-Assisted Network Protocol Testing from Specifications

draft-cui-nmop-auto-test-00

Yong Cui, **Yunze Wei**, Kaiwen Chi, Xiaohui Xie, Shailesh Prabhu

Tsinghua University, Nokia

Background

Why Protocol Testing Matters

- Validates conformance, functionality, robustness, performance, and security.

Why It Remains Labor-Intensive

- Engineers must:
 - Interpret specifications and identify test-relevant behavior
 - Design test cases: topologies, procedures, parameters, and expected results
 - Produce coordinated tester scripts and DUT configurations
 - Execute tests and analyze results

Key Challenges in Specification-Derived Testing

Semantic Modeling

- Extract test-relevant semantics from specification text
- Represent messages, states, algorithms, and error handling
- Preserve semantic dependencies and links to source text

Test Case Synthesis

- Generate representative scenarios rather than restating specification text
- Specify topology, preconditions, and test procedures
- Derive and validate test oracles

Heterogeneous Test Execution

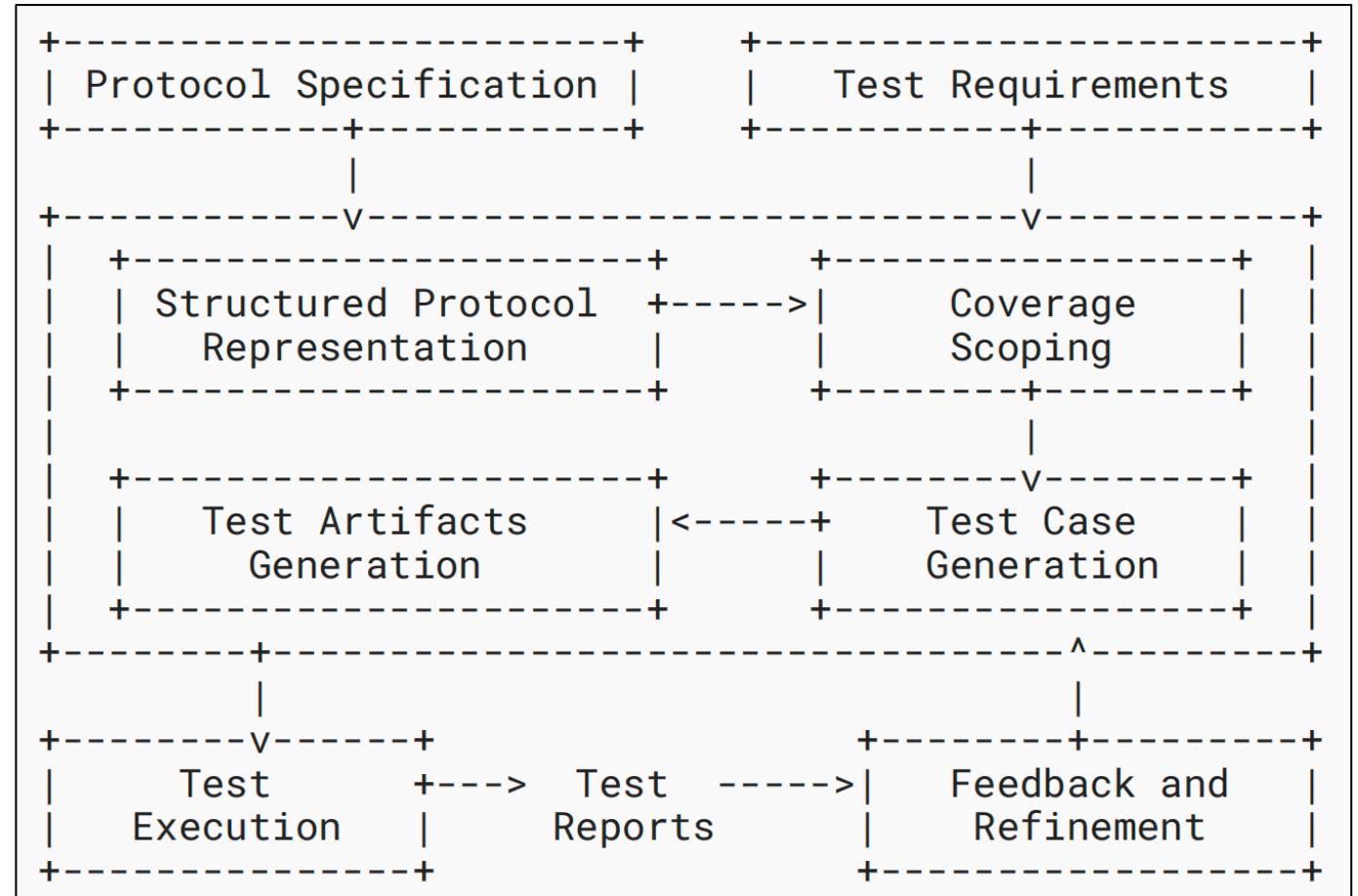
- Translate abstract test cases into coordinated tester scripts and DUT configurations
- Preserve test intent across APIs, CLIs, data models, device platforms, and testbeds

Key challenges: preserving specification semantics, generating discriminating tests, and executing them across heterogeneous environments

Framework

- **Inputs:** protocol specification and test requirements
- **Stages:** representation -> coverage -> cases -> artifacts -> execution -> refinement
- **Forward cycle:** carry test-relevant information toward execution
- **Backward cycle:** use feedback to revisit artifacts, cases, scope, or representation

Explicit stage boundaries make intermediate outputs reviewable and traceable



Structured Protocol Representation and Coverage Scoping

Structured Protocol Representation

- **Purpose:** preserve test-relevant semantics without requiring a complete formal model
- **Content:** message formats, state transitions, algorithms, and error handling
- **Updates:** express update documents as deltas linked to base modules
- **Construction:** analysis -> module induction -> formalization

Coverage Scope

- **Input:** protocol representation and test requirements
- **Decision record:** behavior, inclusion/exclusion status, priority, and rationale
- **Output:** an explicit, reviewable coverage scope before test generation

Protocol semantics and test coverage decisions remain separate artifacts

Test Case Generation: Breadth and Depth

Test Template

- Objective and specification reference
- Topology, preconditions, and static configuration
- Procedure and expected result or oracle
- Variable placeholders for later instantiation

Parameters and Oracles

- Valid, invalid, and boundary values
- Timing, topology, and computed oracles

BGP Update-Aware

RFC 9774 AS_SET deprecation
in RFC 4271 context

RIP Bug Exposure

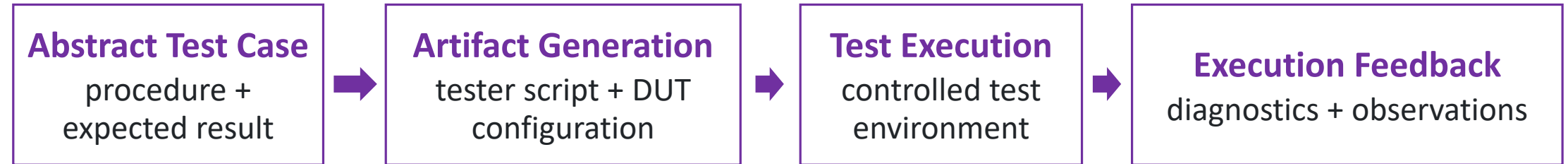
Implementation defect in RFC
2453 default-route handling

OSPF Boundary

Asymmetric priorities
and forced DR re-election

Test templates provide breadth, while parameters and oracles provide depth

Executable Artifact Generation, Execution, and Feedback



Executable Artifact Generation

- **Inputs:** tester APIs, DUT manuals, prior scripts, and testbed context
- **Output:** coordinated tester scripts and DUT configurations
- **Risks:** API misuse, invalid configuration, timing errors, or tester-DUT mismatch

Feedback and Refinement

- **Signals:** API responses, diagnostics, packet captures, and pass/fail observations
- **Likely Sources:** artifacts, test cases, test environment, or DUT
- **Refine:** artifacts, test cases, scope, protocol representation

Validate the test before treating its result as evidence about the DUT

Next Steps and Mailing-List Feedback

Next Steps

- Discuss the respective responsibilities of AI Agents and humans in protocol testing at different levels of AI capability.
- Should the draft be split into separate documents for requirements and stage-specific methodologies?

Mailing-List Feedback (From Brad): Cross-Vendor DUT Configuration

Operational gap

Vendor extensions, proprietary CLIs, YANG deviations, and custom models complicate cross-vendor configuration.

Direction: information model or ontology reconciliation

Integration: additional input and/or validation layer for DUT configuration generation

Placement: generation-time mapping, validation-time checking, or both?

Related: RFC 3444; draft-janz-nmrg-ontology-reconciliation

Thank you!

Questions and comments are welcome!

draft-cui-nmop-auto-test-00

Yong Cui, **Yunze Wei**, Kaiwen Chi, Xiaohui Xie, Shailesh Prabhu

Tsinghua University, Nokia