IETF 123 Hackathon

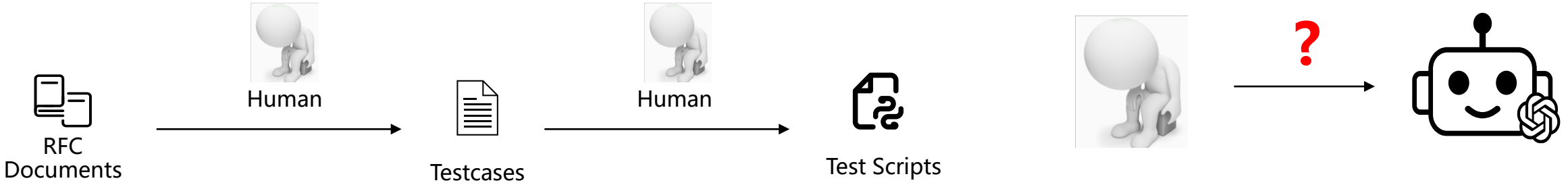
Large Language Model Driven Automated Network Protocol Testing

19-20 July 2025

Yong Cui, Xiaohui Xie, Yunze Wei and **Kaiwen Chi**

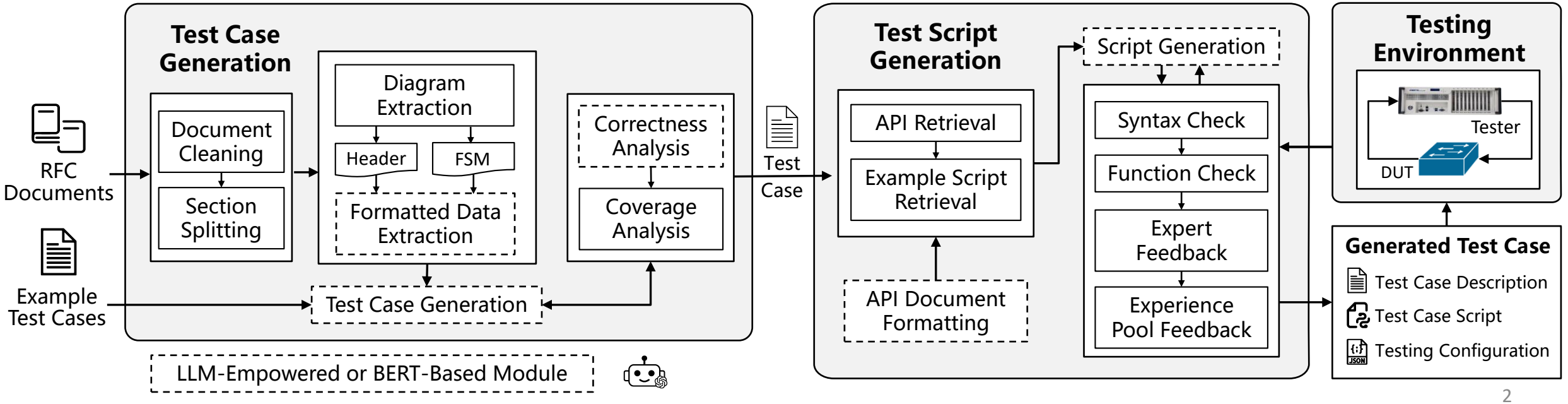
Background & Motivation

- Traditional network protocol testing faces critical challenges in adapting to rapid protocol evolution, primarily due to:
  - Complex and verbose protocol specifications (e.g., RFC documents) that are hard to parse.
  - Labor-intensive customized test code development, relying on manual API calls and device documentation.
  - Underutilization of accumulated expert knowledge.



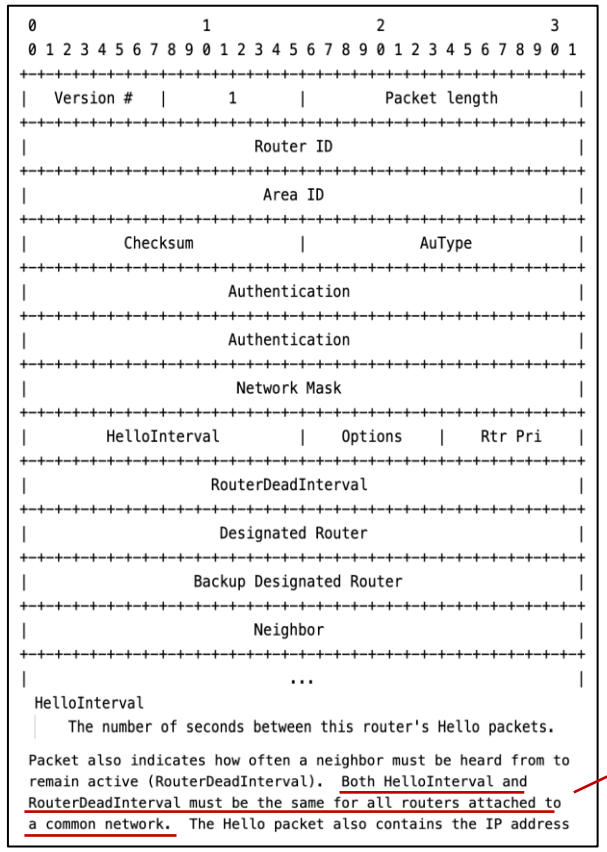
Design

- To address this, we proposed NeTestLLM, a Large Language Model(LLM)-powered End-to-End framework that automates protocol testing containing:
  - Hybrid Test Case Generation
  - Retrieval-Feedback Code Generation

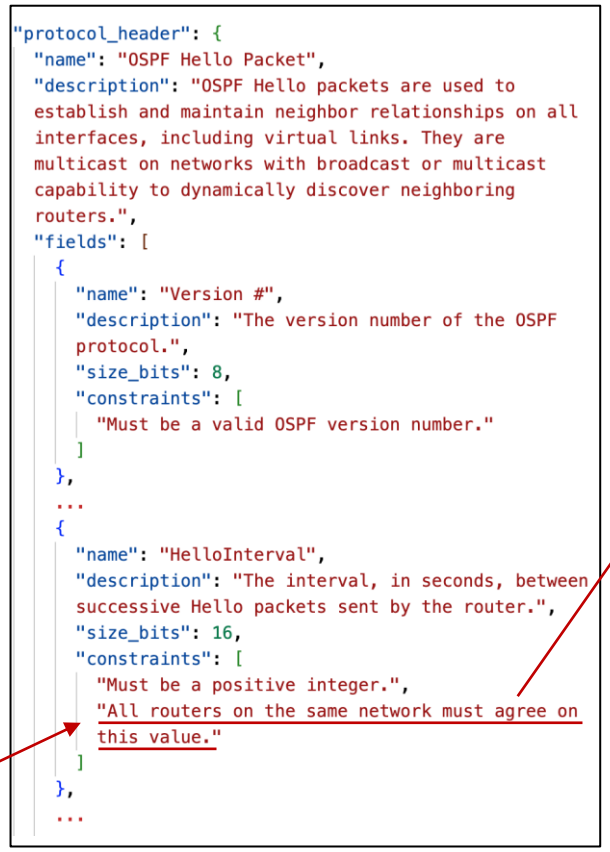


Example Testcase

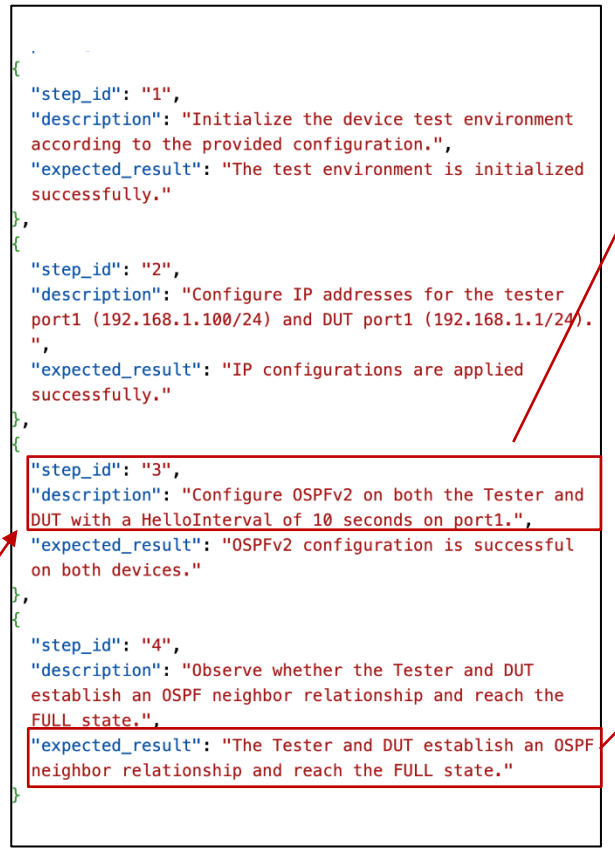
Test Case: OSPF Hello Interval Field



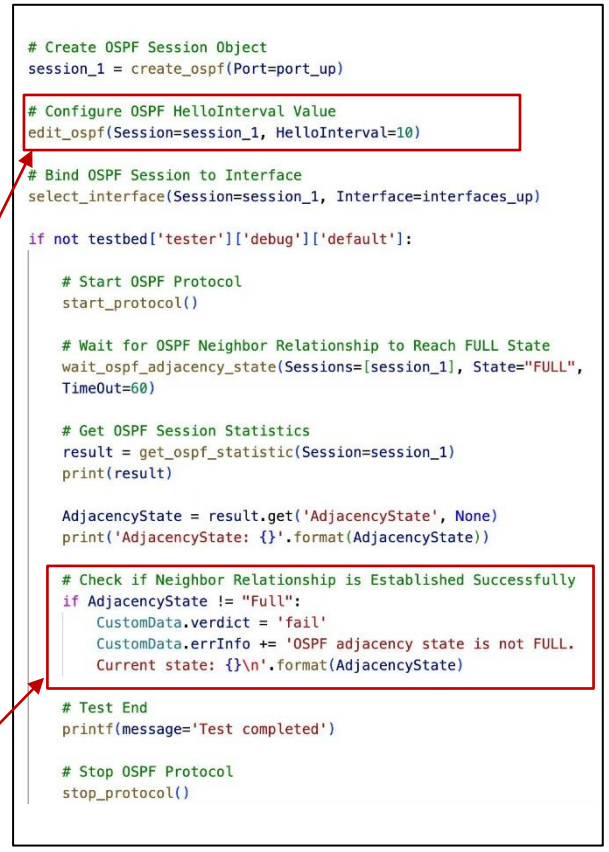
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Version #   |       1       |         Packet length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Router ID                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Area ID                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Checksum             |             AuType            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Authentication                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Authentication                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Network Mask                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        HelloInterval          |    Options    |    Rtr Pri    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     RouterDeadInterval                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Designated Router                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Backup Designated Router                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Neighbor                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            ...                                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

HelloInterval
    The number of seconds between this router's Hello packets.

Packet also indicates how often a neighbor must be heard from to
remain active (RouterDeadInterval).  Both HelloInterval and
RouterDeadInterval must be the same for all routers attached to
a common network.  The Hello packet also contains the IP address
```



```
"protocol_header": {
  "name": "OSPF Hello Packet",
  "description": "OSPF Hello packets are used to
  establish and maintain neighbor relationships on all
  interfaces, including virtual links. They are
  multicast on networks with broadcast or multicast
  capability to dynamically discover neighboring
  routers.",
  "fields": [
    {
      "name": "Version #",
      "description": "The version number of the OSPF
      protocol.",
      "size_bits": 8,
      "constraints": [
        "Must be a valid OSPF version number."
      ]
    },
    ...
    {
      "name": "HelloInterval",
      "description": "The interval, in seconds, between
      successive Hello packets sent by the router.",
      "size_bits": 16,
      "constraints": [
        "Must be a positive integer.",
        "All routers on the same network must agree on
        this value."
      ]
    },
    ...
```



```
{
  "step_id": "1",
  "description": "Initialize the device test environment
  according to the provided configuration.",
  "expected_result": "The test environment is initialized
  successfully."
},
{
  "step_id": "2",
  "description": "Configure IP addresses for the tester
  port1 (192.168.1.100/24) and DUT port1 (192.168.1.1/24).
  ",
  "expected_result": "IP configurations are applied
  successfully."
},
{
  "step_id": "3",
  "description": "Configure OSPFv2 on both the Tester and
  DUT with a HelloInterval of 10 seconds on port1.",
  "expected_result": "OSPFv2 configuration is successful
  on both devices."
},
{
  "step_id": "4",
  "description": "Observe whether the Tester and DUT
  establish an OSPF neighbor relationship and reach the
  FULL state.",
  "expected_result": "The Tester and DUT establish an OSPF
  neighbor relationship and reach the FULL state."
}
```



```
# Create OSPF Session Object
session_1 = create_ospf(Port=port_up)

# Configure OSPF HelloInterval Value
edit_ospf(Session=session_1, HelloInterval=10)

# Bind OSPF Session to Interface
select_interface(Session=session_1, Interface=interfaces_up)

if not testbed['tester']['debug']['default']:

    # Start OSPF Protocol
    start_protocol()

    # Wait for OSPF Neighbor Relationship to Reach FULL State
    wait_ospf_adjacency_state(Sessions=[session_1], State="FULL",
    TimeOut=60)

    # Get OSPF Session Statistics
    result = get_ospf_statistic(Session=session_1)
    print(result)

    AdjacencyState = result.get('AdjacencyState', None)
    print('AdjacencyState: {}'.format(AdjacencyState))

    # Check if Neighbor Relationship is Established Successfully
    if AdjacencyState != "Full":
        CustomData.verdict = 'fail'
        CustomData.errInfo += 'OSPF adjacency state is not FULL.
        Current state: {}\n'.format(AdjacencyState)

    # Test End
    printf(message='Test completed')

    # Stop OSPF Protocol
    stop_protocol()
```

OSPF RFC Protocol Fields → Extracted Formatting Constraints (61 Fields Total) → Generated Test Cases (146 Cases Total) → Generated Test Code

Future work

- Better evaluation on generated testcases
- New RFC protocol emulation implementation
- Vender-customized automated protocol testing
- …

Next Steps

- HotRFC: Sunday
  - Large Language Model Driven Automated Network Protocol Testing
- BMWG: Monday
  - draft-cui-bmwg-testcase-spec
- ANRW: Tuesday
  - Large Language Model Driven Automated Network Protocol Testing
- NMRG: Friday
  - draft-cui-nmrg-auto-test

Wrap Up

- Champions:

  Yong Cui (cuiyong@tsinghua.edu.cn)
  Xiaohui Xie (xiexiaohui@tsinghua.edu.cn)
  Yunze Wei (wyz23@mails.tsinghua.edu.cn)
  Kaiwen Chi (ckw24@mails.tsinghua.edu.cn)

- First timers @ IETF/Hackathon ☺:
  - Yunze Wei and Kaiwen Chi
- Internet Drafts:
  - draft-cui-bmwg-testcase-spec
  - draft-cui-nmrg-auto-test
- Repositories:
  - https://github.com/ckw20/NetAutoTest_Hackathon