Testing SIP
Using XML Protocol Templates

M. Ranganathan
Olivier Deruelle
Doug Montgomery
Advanced Networking Technologies Division,
National Institute of Standards and Technology,
Gaithersburg, MD 20899, USA.

http://www-x.antd.nist.gov/index.html
Introducing SIP

- Peer-to-peer Signaling Protocol used for IP Telephony, Conferencing and Instant Messaging
- Introduced in 1999
  - 9 revisions and 2 RFCs since then!
- Widely deployed - Microsoft RTC Server, IM Client, Cisco gateways etc.
Introducing SIP

- Text based
  - unlike H.323
- “Stateless”
  - Protocol state encoded in message
- Extensible
  - Many extensions exist.
- Can run over unreliable or reliable transports
  - Out of order / dropped signaling messages.
Simple SIP Call Flow

UAC – User Agent Client

UAS – User Agent Server

There can be intermediate Signaling nodes (Proxy Servers that keep call state).
Protocol Complications

- Protocol is robust and extensible:
  - SIP keeps enough state in the Messages to deal with all these complications.
  - Correct implementation is tricky.
- Signaling may have to go through multiple hops.
- Proxy servers may go down without warning.
- Peers may go down without warning.
- Sessions can move without prior planning.
- Network can fail without warning.
SIP Testing

- **Load Testing**
  - Generate 100’s of simultaneous sessions.

- **Call Flow Testing**
  - Unit testing the SIP Protocol Implementation by generation of scenarios.
  - Primary mode of testing during SIP interoperability test events.
Generating SIP Test Cases

- Exhaustive testing generates too many test cases.
- End-to-end testing is feasible
  - protocol state and causality is encoded in the Messages/Call Flows.
- Understanding implementation complexities results in good test cases.
Layer the Test Cases like Applications/Stacks are layered
- Message Layer, Transaction Layer and Dialog Layer tests
- This prunes the number of tests and makes the tests more meaningful.
SIP Messages

- Protocol encodes all the state it needs in the message.
  - HTTP/Mail-like headers and a Request Line or a Status Line.
  - SIP Components use Messages to identify protocol abstractions.

- Protocol State is encoded in
  - Request URI, From, To, Via, CSeq, CallId, Max-Forwards
  - *Stateless* components built directly on Message Layer.
Message Layer

- Handle Incoming Requests/Responses
  - Read Raw messages from Network.
  - Output Parsed Messages to Transaction layer.

- Dispatch outgoing messages
  - Input Parsed Messages from Transaction Layer.
  - Encode Parsed Messages and send out on Network.
Grammar is context sensitive and defined using ABNF
- Grammar has changed between RFCs
- Grammar is compositional (mail, URL, HTTP)
- Parser generators have trouble with RFC grammar
  - usually hand coded parsers are used
  - Some tools are available – antlr
- Headers are Text (Body can be Binary)
SIP Applications are transaction oriented and usually interact directly with a transaction layer.

Primary duties of the Transaction Layer

- Request Response matching
- Retransmission handling for unreliable media.
- Timeout handling
Transaction Layer: Common Bugs

- Implementations do not implement the Transaction State Machine correctly
- Implementations have difficulty keeping backward compatibility
  - In RFC 3261 the branch ID of the topmost "Via" header identifies the Transaction
  - RFC 2543 used a hash over From, To, Request URI and Via headers
Transaction Layer Testing

- Simulate lost messages
  - Drop Requests/Responses
- Simulate timing variations
  - Delay Responses
  - Generate out of order responses
- Simulate stray messages
- CANCEL messages for Server Transactions that do not exist.
  - Late CANCELS
  - Late ACKs
  - Duplicate ACKs
  - Out of Sequence messages.
Transaction Layer Testing

Transaction matching / identification

-Via Header branch parameter variations
  - RFC 3261 relies on this for matching
- Variations in From / To, Request URI and CSeq Sequence Number and CSeq Method
  - RFC 2543 relies on this for matching.
Dialog Layer

- Dialog is a peer-to-peer association between communicating SIP endpoints
  - Dialogs established by Dialog creating Transactions.
  - Not all transactions create Dialogs.
  - A Transaction may belong to exactly one Dialog.
- SIP messages carry enough state to identify the Dialog directly from the message
Dialog Layer

- Manages Dialog Creation/Teardown
  - Dialogs created by transaction completion
- Manages Route Sets
  - Test agent must test for expected Route / Record-Route headers in requests
- Manages Sequence Numbers
  - Test agent must test for sequence number assignment
- Manages the Request URI
Dialogs are identified by portions of a message:

- CallID, From, To tags in RFC 3261
- CallID, From, To addresses RFC2543
- Stacks try to keep backward compatibility
- Bugs are frequently caused by tag management problems.
Testing the Dialog Layer

- Requests/Responses within and outside Dialog
- Requests/Response for Spurious Dialogs
  - Variation in From/To Headers and Tags
  - Generate Requests for Dialogs that do not exist.
- CSeq Header Sequence number variations
  - Out of sequence message arrivals
Call Flow Testing Approach

- Test the causal sequence of messages required to establish and release SIP Calls
- SIP Protocol Template – an XML pattern for a SIP Call Flow.
- XML Pattern input to a customizable user agent which can run the Call Flow (Responder)
Motivation

- XML is hierarchical
  - good way to represent SIP protocol abstractions

- Interoperability testing with control
  - Typically components are tested in call flow scenarios
  - Typically operating in an un-controlled environment
  - Reproducing complex scenarios is difficult
XML representation of SIP

- Define a set of XML tags to represent the required headers in a SIP message
- Define XML tags to express call flow state machine
- Input to Event Engine that can run the call flow
- Generate variations of the call flow by modifying the XML script
Test Scripting Architecture

Call Flow Testing Protocol Template

Event Engine

NIST-SIP Stack

Service Function

Service Container
XML Engine + Java/Jython interpreter

Service Script – a callout to a service function at transition points in the call flow.

Network
XML Tags Mirror Protocol

Structure

XML Tags

CALL_FLOW

+ ?

DIALOG

* *

CLIENT/SERVER TRANSACTION

TRANSITION

? ?

GENERATE

TRIGGER_MESSAGE

TIMER

GENERATE

TRANSACTION

MESSAGE Tags

MATCH Tags

Partial List of Attributes

CALL_FLOW

dialog
executeOnNoMatch
executeOnTransactionTimeout
executeOnDialogNotFound
executeOnTransactionNotFound
global Events

DIALOG

id

Events

CLIENT_TRANSACTION

executeOnTransactionCompletion

generateEventsOnTransactionCompletion

TIMER

type
delay
generateEventOnTimeout

TRANSITION

trigerEvent
eexecuteOnTrigger

consumedEvents
generatedEvents
setTimeout
disarmTimer
Test Script

- Pattern matching, timer events and transitions used for triggering transitions in test script.
- Test script is represented by a set of Transactions that may be nested within a dialog.
- The entire transaction state machine is exposed and defined using XML.
  - Timing can be varied and controlled errors can be created.
- Service code can be called when messages arrive, transactions are started, transactions complete, dialogs are created or dialogs complete.
Why Do It?

- Simple, clear expression of test scenarios
  - Protocol maps to XML script one to one
- Can generate multiple scenarios based on small variations for the same call flow.
- Can simulate common end-point (User Agent) behavior.
- Can generate controlled error conditions/timing variations.
Test Log File Collection

- Log file is a diagnostic tool to help debug protocol problems.
- Stack generates log files using XML format.
  - Distributed traces are collated at test proxy
- Trace viewer pairs arcs by Transactions
Visualizing the Trace

- Java Applet collects and visualizes distributed call flow trace files.
- Augmented with XML script state information.
- Enables debugging call flows & test scripts.
Related work

- TTCN testing of SIP
  - Procedural test cases
  - Not explicitly tailored to SIP
- Using our approach
  - Simplifies logical design
  - XML tools can be used for test case design.

http://www-x.antd.nist.gov/proj/iptel
Extensions and Future Work

- Standardize XML representation of the SIP protocol
- Off line protocol verification
  - Generation of Call flows based on message logs
  - Verification of traces based on message logs
- Customizable test scripts
- Extensions to service creation.
  - Integration with other distributed scripting technologies
  - JXTA, SOAP