Testing online services

Philip Withnall

August 3, 2013

The GNOME Conference
GUADEC

Online services

- Google
- OpenStreetMap
- Facebook
- ...

Testing what, exactly?

Client conforms to some past behaviour of the server?

Or to the current behaviour?

Ghost of behaviour past

- Offline testing: reliable
- Not what users care about

Ghost of behaviour present

- What users care about
- Requires a network connection to run tests
- Unreliable; frequent breakages

Ghost of behaviour future

Web services change: updating tests must be easy

Distributed systems

Web services are often distributed systems

Distributed systems

Web services are often ~~distributed systems~~ fragile

A solution?

1. Record sample network traces between client and server
2. Test against the traces
3. Periodically compare and update traces

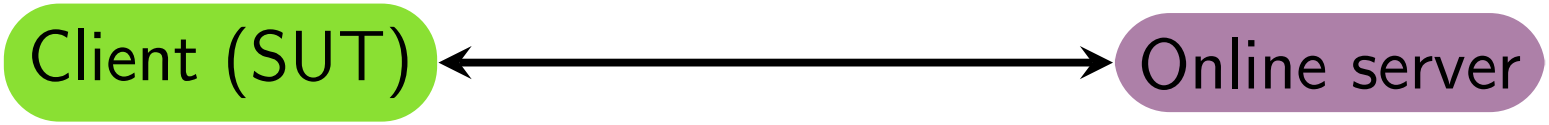
Existing solutions

WireMock	Java-only, separate process (`http://wiremock.org/`)
REST-driver	Java-only, REST-only (`https://github.com/rest-driver/rest-driver`)
Betamax	Java-only (`http://freeside.co/betamax/`)
python-dbusmock	D-Bus (`https://launchpad.net/python-dbusmock`)
umockdev	Hardware (`https://launchpad.net/umockdev`)

Approach

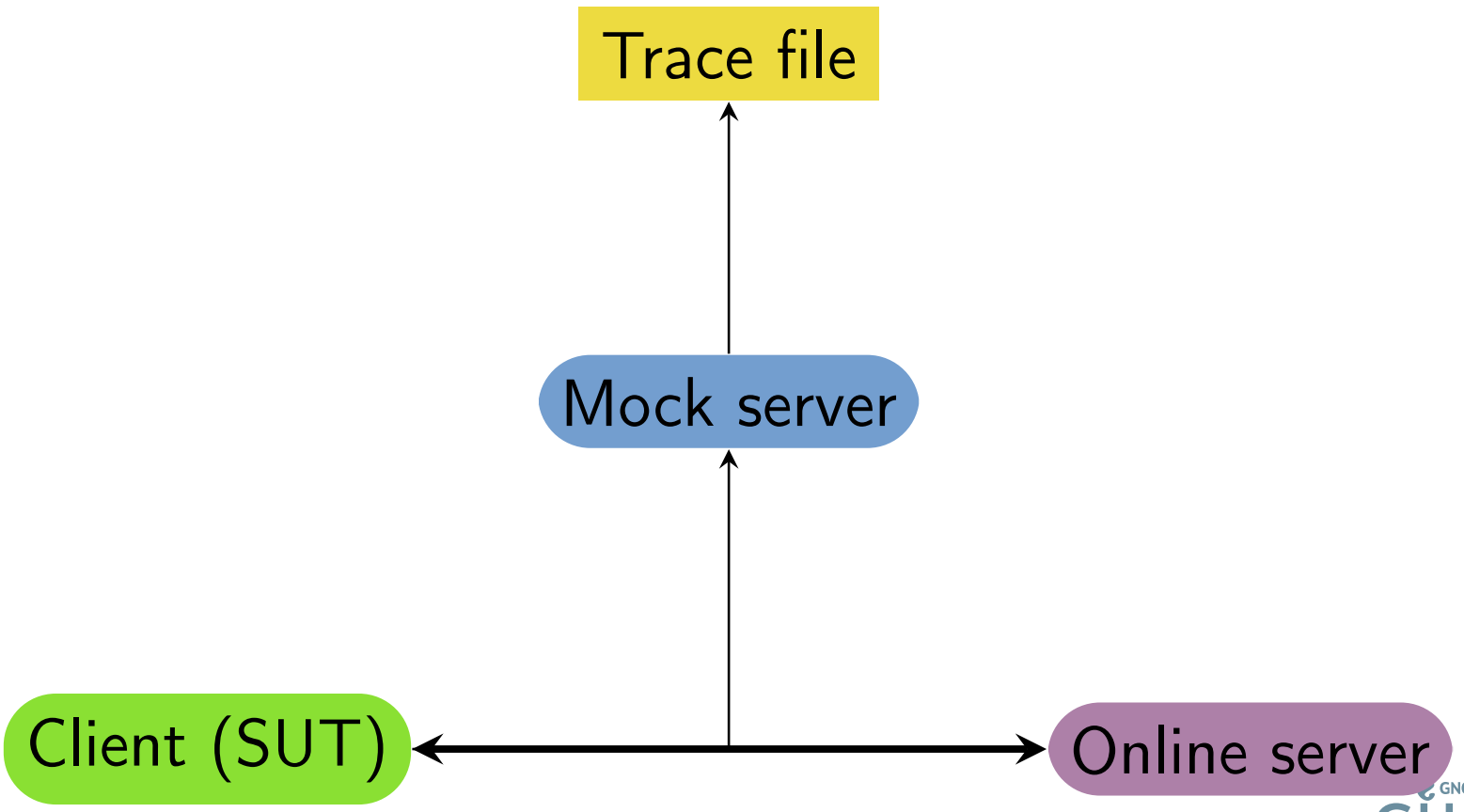
- Override client's DNS resolver and redirect all requests to a loopback server
- Replay a network trace from the loopback server and catch client requests which don't match the trace
- Or, don't override the DNS and test against the actual server

Shiny diagrams



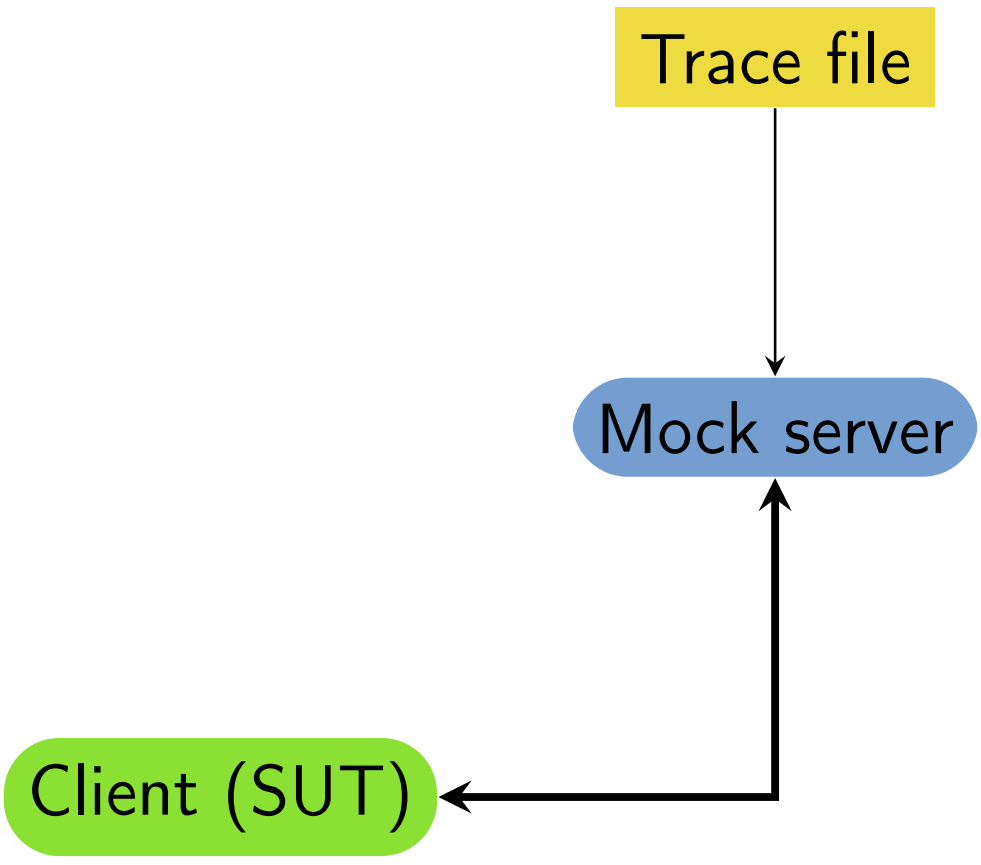
Shiny diagrams

Logging



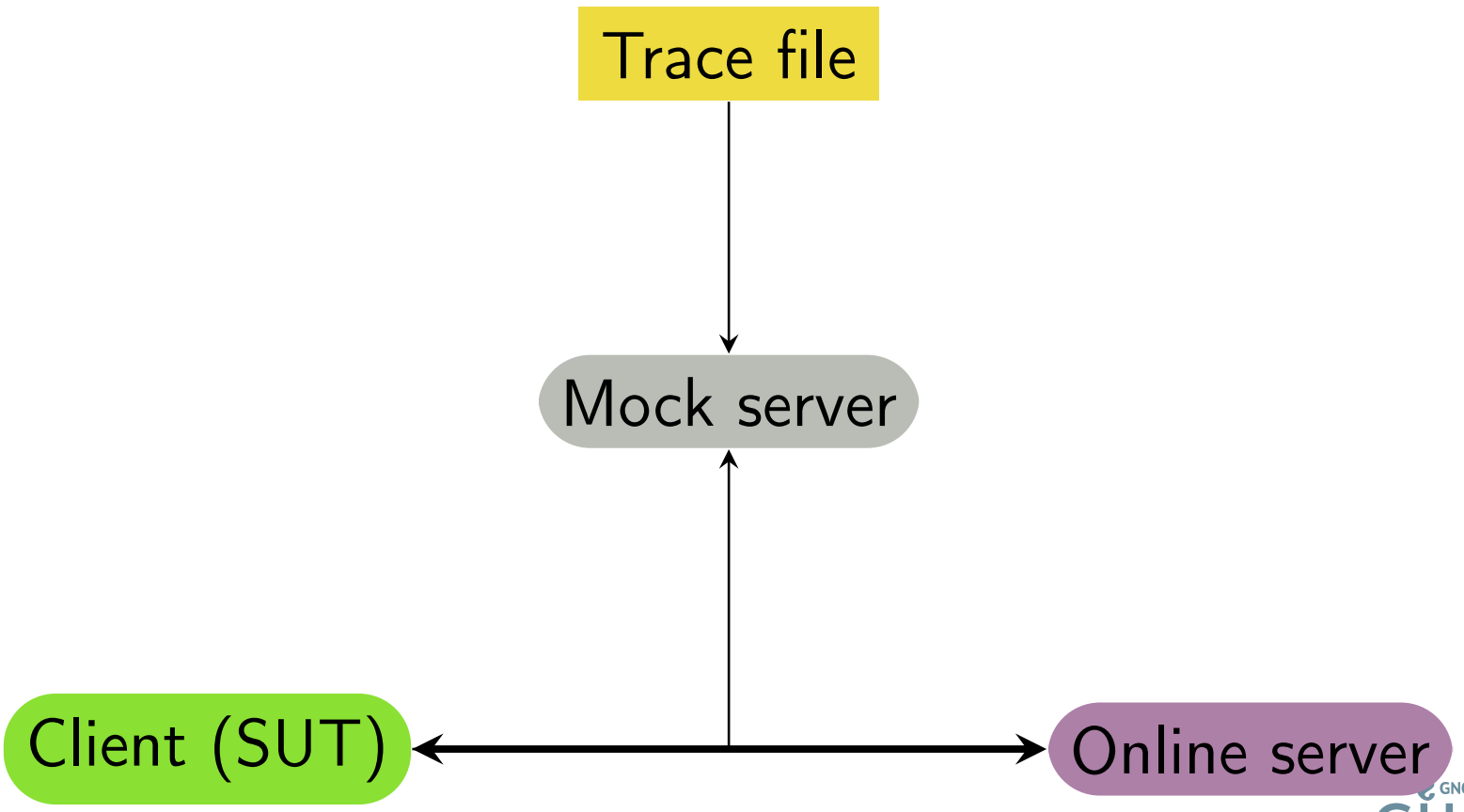
Shiny diagrams

Testing



Shiny diagrams

Comparing



Unit test example

```
mock_server_set_trace_directory (server, dir);
mock_server_start_trace (server, "my-test");
my_app_set_port (mock_server_get_port (server));

if (!mock_server_get_enable_online (server)) {
ip = mock_server_get_address (server);
res = mock_server_get_resolver (server);
mock_resolver_add_A (res, "google.com", ip);
}

/* Test code goes here. */

mock_server_end_trace (server);
```

Results: libgdata

- libgdata `make check`: 137 s (was $>$ 10 minutes)
- libgdata code coverage: 81% of lines (slightly increased)

Problems

- Reuse of setup/teardown functions
- IDs and timestamps
- Cancellation of messages needs work

To-do

- Split out into a separate project
- Improve support for ‘standard’ error response testing
- Apply to other libraries

Miscellany

uhttpmock `https://gitorious.org/uhttpmock`

libgdata `https://git.gnome.org/browse/libgdata`

Beamer theme: `https://github.com/kittykat/guadec-theme-2013`