TAP JUGGLING
Test Anything Protocol - everywhere

Steffen Schwigon, Operating System Research Center
August 15, 2011
Public
INTRODUCTION
INTRODUCTION | Abstract

The Operating System Research Center (OSRC), a global AMD Research organisation headquartered in Dresden, Germany, acts as a bridge between the OS development community and the worldwide AMD processor design community.

At the OSRC we develop and run a test infrastructure called "Tapper". One of its used essential technologies is the Test Anything Protocol, or TAP.

By that postulate we use, write, and combine TAP tools in occasionally unorthodox ways to achieve our goals. In this talk I will give an overview of available TAP tools and related topics.
**INTRODUCTION | Context**

- Operating System Research Center (OSRC)
- We developed and run a test infrastructure called *Tapper*
  - Automated testing of operating systems and virtualization (Xen/KVM)
  - Published as open source in 2011
    - [http://github.com/amd](http://github.com/amd)
- Central idea: Test Anything Protocol - TAP
  - *Tapper* is a **TAP database**
  - We “**lazy evaluate**” TAP (in contrast to produce TAP)
  - TAP is our **daily business**
- Overview of available TAP tools
  - not restricted to *Tapper*
INTRODUCTION | Motivation

- TAP is a data format
- Primarily for expressing test results
- Easier than XML, YAML, JSON, etc.
- Yet allows embedded YAML (simple subset)
- Errors are forgiven (non-TAP lines are ignored)
- Migrates all the complexity to the toolchain developer
- User “whipuptitute” + developer “manipulexity” \(\rightarrow\) low usage barrier
INTRODUCTION | Agenda

- Escalate in complexity
  - Tier 1: TAP basics
  - Tier 2: TAP v13 and formatting
  - Tier 3: Transport meta information
  - Tier 4: TAP as document object model
  - Tier 5: Write TAP applications
  - Tier 6: Test waivers
  - Final Tier: Nested TAP
**TIER 1 | TAP Basics**

- Line-based protocol
- Starts with a “plan” – how many test lines expected
- Some “ok” test lines
- Some “not ok” test lines
- Directives # TODO / # SKIP on test lines
- Comment lines starting with “#”
- Unrecognized lines are ignored
TIER 1 | TAP Basics | Synopsis

1..3

ok
ok
not ok

- Plan and ok/not ok lines
1...3

- **ok** - established connection
- **ok** - checksum
- **not ok** - transfer completed

- Plan and ok/not ok lines
- Test line descriptions
1..3

ok - established connection
ok - checksum
not ok - transfer completed

# got error message "Bummer!"

- Plan and ok/not ok lines
- Test line descriptions
- Comment lines
1..3

**ok** - established connection

**ok** - checksum

**not ok** - transfer completed # TODO we know it fails

# got error message "Bummer!"

- Plan and ok/not ok lines
- Test line descriptions
- Comment lines
- Directives # TODO
TIER 1 | TAP Basics | Synopsis

1..3

ok - established connection
ok - checksum # SKIP no md5sum available
not ok - transfer completed # TODO we know it fails
# got error message "Bummer!"

- Plan and ok/not ok lines
- Test line descriptions
- Comment lines
- Directives # TODO / # SKIP
TIER 1 | TAP Basics | Synopsis

1..3

ok - established connection
ok - checksum # SKIP no md5sum available
not ok - transfer completed # TODO we know it fails
# got error message "Bummer!"

Hello? I am a statement lost in code, help me out!

- Plan and ok/not ok lines
- Test line descriptions
- Comment lines
- Directives # TODO / # SKIP
- Unrecognized lines are ignored
**TIER 1 | TAP Basics | Synopsis**

1..3

- **ok 1** - established connection
- **ok 2** - checksum # SKIP no md5sum available
- **not ok 3** - transfer completed # TODO we know it fails

# got error message "Bummer!"

Hello? I am a statement lost in code, help me out!

- Plan and ok/not ok lines – optionally numbered
- Test line descriptions
- Comment lines
- Directives # TODO / # SKIP
- Unrecognized lines are ignored
1..3

ok 1 - established connection

ok 2 - checksum # SKIP no md5sum available

not ok 3 - transfer completed # TODO we know it fails

# got error message "Bummer!"

Hello? I am a statement lost in code, help me out!

→ TAP is like Perl
1..3

ok 1 - established connection
ok 2 - checksum # SKIP no md5sum available
not ok 3 - transfer completed # TODO we know it fails
# got error message "Bummer!"

Hello? I am a statement lost in code, help me out!

→ TAP is like Perl (without sigils)
TIER 1 | Run and evaluate TAP emitters

$ prove foo.t  # run + evaluate
$ prove -m foo.t  # run + evaluate + merge STDIN/OUT
$ prove -e cat static_tap_results.tap  # just evaluate
TIER 2 | Skilled
1..3

ok - established connection

ok - checksum # SKIP no md5sum available

not ok - transfer completed # TODO we know it fails

# got error message "Bummer!"

Hello? I am a statement lost in code, help me out!
1..4

ok  - established connection

ok  - checksum # SKIP no md5sum available

not ok - transfer completed # TODO we know it fails

# got error message "Bummer!"

Hello? I am a statement lost in code, help me out!

ok  - transfer benchmarks
OK - established connection
OK - checksum # SKIP no md5sum available
NOT OK - transfer completed # TODO we know it fails

# got error message "Bummer!"

Hello? I am a statement lost in code, help me out!

OK - transfer benchmarks

```yaml
---
benchmarks:
  pass1:
    snd: 1234.56
    rcv: 999.99
  pass2:
    snd: 1123.56
    rcv: 888.88
...```
TAP Version 13

1..4

ok - established connection
ok - checksum # SKIP no md5sum available
not ok - transfer completed # TODO we know it fails
# got error message "Bummer!"
Hello? I am a statement lost in code, help me out!
ok - transfer benchmarks

---

benchmarks:
  pass1:
    snd: 1234.56
    rcv: 999.99
  pass2:
    snd: 1123.56
    rcv: 888.88
...
TIER 2 | TAP v13 | Enforce version

- “TAP Version 13” - annoying detail for TAP producing end users
- Configure on consumer side
  
  $ prove --tapversion=13
TIER 2 | TAP Formatting

- HTML formatter
  
  $ prove -Q --formatter=TAP::Formatter::HTML > out.html

- Same via plugin
  
  $ prove -Q -P HTML:outfile:out.html
TIER 2 | TAP Formatting

- HTML formatter
  
  $ prove -Q --formatter=TAP::Formatter::HTML > out.html

- Same via plugin
  
  $ prove -Q -P HTML:outfile:out.html

- Used in Tapper...
TIER 2 | TAP Formatting

209083: AutoTest-hackbench
Test Execution Context
Test results

Test file
hackbench/keyval.tap

TAP Version 13
1.2
ok 1 - results

version 1

ok 2 - results

hackbench/results/keyval.tap 100.0%

hackbench/status.tap 100.0%

status ten 100.0%
TIER 3 | Combat
“Hot comments“ - meta information in comment lines

*Tapper*-specific extension to evaluate them

Example: “t/00-tapper-meta.t”

```perl
use Tapper::Test;
tapper_suite_meta;
```

Result TAP

```
1..1
 ok 1 - tapper-suite-meta

# Tapper-suite-name: Some-Library
# Tapper-suite-version: 3.000010
# Tapper-machine-name: bascha
# Tapper- uname: Linux bascha 2.6.35-30-generic #54-Ubuntu SMP x86_64 GNU/Linux
# Tapper-osname: Ubuntu 10.10
# Tapper-cpuinfo: 2 cores [AMD Athlon(tm) 64 X2 Dual Core Processor 6000+]
# Tapper-ram: 2007MB
```
TAP::DOM – Synopsis

```perl
use TAP::DOM;
$tapdata = TAP::DOM->new( tap => $tapstr ); # same options as TAP::Parser
print Dumper($tapdata);
```
**TIER 4 | TAP as document object model**

- Resulting data structure

```perl
bless (
    'tests_planned' => 6,
    'tests_run' => 8,
    # [...]
    'summary' => {
        'status' => 'FAIL',
        'total' => 8,
        'passed' => 6,
        'failed' => 2,
        'skipped' => 1,
        'todo' => 4,
        'todo_passed' => 2,
        # [...]}
),
'lines' => [
    { 'number' => '1',
        'is_ok' => 1,
        'description' => '- use Data::DPath;',
        '_children' => [
            # subsequent comments/yaml
            { 'is_yaml' => 1,
                'data' => [
                    { 'name' => 'Hash one', 'value' => '1' },
                    { 'name' => 'Hash two', 'value' => '2' }
                ] } ]
    # [... lines ...]
], 'TAP::DOM')
```
**TIER 4 | TAP as document object model**

- Uh, oh, complexity!
- Data::DPath to the rescue
  - **fuzzy** paths through data structures

  - `$tapdom ~~ dpath '//summary/passed'`
  - `$tapdom ~~ dpath '//description//foo'`
Example: find succeeding TODO tests

$ tap-emitting-test.sh | dpath -i tap '//'has_todo[value==1]//..//is_actual_ok[value==1]//..'

---

- as_string:    "ok 149 - ANYWHERE + NOSTEP # TODO deferred"
  description:  "- ANYWHERE + NOSTEP"
  directive:    TODO
  explanation:  deferred
  has_todo:     1
  is_actual_ok: 1
  is_ok:        1
  is_test:      1
  number:       149
  type:         test
Example: extract benchmarks

```bash
$ perl xt/large_data.t | dpath -i tap //wallclock
```
TIER 4 | TAP::DOM and Data::DPath

- **Tapper use case: “TAP pass-through”**
  - Subscribe to dedicated data blocks in TAP and forward them
  - Test suite → *Tapper* "reports receiver" → "level 2 receivers"
  - Subscribe to subsets of test results
  - Extract and forward to appropriate 3rd party applications
  - E.g., benchmark values to codespeed application


**Tapper use case: “TAP pass-through”**

```perl
# anywhere inside big TAP report...
ok - benchmark example data
---
codespeed:
  
  benchmark: example1
  commitid: 1b1a3d2a
  environment: myhost
  executable: perl-5.12.1
  project: perl
  result_value: 12.345

  benchmark: example2
  commitid: 1b1a3d2a
  environment: myhost
  executable: perl-5.12.1
  project: perl
  result_value: 9.876

# some more TAP
```
**Tapper use case: “TAP pass-through”**

- TAP pass-through implemented in 2 lines

```perl
method forward_to_codespeed ($tap_dom) {
    # step 1 - fuzzy subscription path
    $chunks = $tap_dom ~~ dpath("//data/codespeed");
    # step 2 - pass-through
    $lwp_useragent->post("http://CODESPEED/result/add/", $_) foreach @$chunks;
}
```
TIER 5 / Veteran
TIER 5 | Writing TAP applications

- Emacs mode
- `prove` plugins
- TAP transformers
TIER 5 | Emacs tap-mode

- Use Emacs tap-mode to edit TAP
TIER 5 | Writing prove plugins

- Carve out TAP from prove
Carve out TAP from \texttt{prove}

- "\texttt{prove -v}" reprints TAP
- Idea: branch off TAP immediately during test run, but...

```
$ prove -vl t/foo.t
t/foo.t ..
1..1
ok 1 - use FOO
ok
All tests successful.
Files=1, Tests=1, 0 wallclock secs ( 0.10 usr 0.00 sys + 0.26 cusr 0.03 csys = 0.39 CPU)
Result: PASS
```
Carve out TAP from **prove**
- "**prove -v**" reprints TAP
- Idea: branch off TAP immediately during test run, but...

```
$ prove -vl t/foo.t
t/foo.t ..
1..1
ok 1 - use FOO
ok
 <-- extra "ok" line - Bummer!
```

All tests successful.
Files=1, Tests=1, 0 wallclock secs ( 0.10 usr 0.00 sys + 0.26 cusr 0.03 csys = 0.39 CPU)
Result: PASS
Carve out TAP from `prove`

- "`prove -v`" reprints TAP
- Idea: branch off TAP immediately during test run, but...

```
$ prove -vl t/foo.t
t/foo.t ..
1..1
ok 1 - use FOO
ok
```

All tests successful.
Files=1, Tests=1, 0 wallclock secs (0.10 usr 0.00 sys + 0.26 cusr 0.03 csys = 0.39 CPU)
Result: PASS

- `prove` output is not “idempotent”
Carve out TAP from `prove`

- "`prove -v`" reprints TAP
- Idea: branch off TAP immediately during test run, but...

```
$ prove -vl t/foo.t
t/foo.t ..
1..1
ok 1 - use FOO
ok                  <-- extra "ok" line - Bummer!
```

All tests successful.
Files=1, Tests=1, 0 wallclock secs (0.10 usr 0.00 sys + 0.26 cusr 0.03 csys = 0.39 CPU)
Result: PASS

- `prove` output is not “idempotent”
- Since `TAP::Harness v3.24` the ok line can be overwritten
TIER 5 | Writing `prove` plugins

- Carve out TAP from `prove`
  - "`prove -v`" reprints TAP
  - Idea: branch off TAP immediately during test run, but...

```
$ prove -vl t/foo.t
   t/foo.t ..
   1..1
   ok 1 - use FOO
   ok                  <-- extra "ok" line - Bummer!
All tests successful.
Files=1, Tests=1, 0 wallclock secs ( 0.10 usr  0.00 sys +  0.26 cusr  0.03 csys =  0.39 CPU)
Result: PASS
```

- `prove` output is not "idempotent"
- Since TAP::Harness v3.24 the ok line can be overwritten
- Module `App::Prove::Plugin::Idempotent` to the rescue
Carve out TAP from `prove`

- "`prove -v`" reprints TAP
- Idea: branch off TAP immediately during test run, but...

```
$ prove -vl -P Idempotent  t/foo.t
  t/foo.t ..
  1..1
  ok 1 - use FOO
All tests successful.
Files=1, Tests=1, 0 wallclock secs (0.10 usr 0.00 sys + 0.26 cusr 0.03 csys = 0.39 CPU)
Result: PASS
```

- `prove` output is not “idempotent”
- Since TAP::Harness v3.24 the ok line can be overwritten
- Module `App::Prove::Plugin::Idempotent` to the rescue
TIER 5 | TAP transformers

- De-concatenate multiple TAP blocks from \textit{prove}
TIER 5 | TAP transformers

- De-concatenate multiple TAP blocks from `prove`
  - Problem: "`prove -v`" reprints TAP from many files concatenated
De-concatenate multiple TAP blocks from `prove`

- Problem: "`prove -v`" reprints TAP from many files concatenated

```
$ prove -vl -P Idempotent t/foo.t t/bar.t
t/foo.t ..
1..1
ok 1 - use FOO
t/bar.t ..
1..1
ok 1 - use BAR
All tests successful.
Files=1, Tests=1, 0 wallclock secs ( 0.10 usr 0.00 sys + 0.26 cusr 0.03 csys = 0.39 CPU)
Result: PASS
```
TIER 5 | TAP transformers

- De-concatenate multiple TAP blocks from \texttt{prove}
  - Solution: Module \texttt{TAP::Splitter} recognizes TAP borders
    - TAP version line
    - Plan line (1..3)
    - \texttt{prove}'s filename line

```perl
use TAP::Snipper;
$tap = slurp("$temp/too_much_tap_in_one_go.tap");
$snipper = TAP::Snipper->new( tap => $tap );
$snipper->_parse_tap_into_sections;
# ARRAY of TAP blocks:
# $snipper->parsed_report->{tap_sections}[*] {raw}
```
TIER 6 / Master
TIER 6 | Test waivers

- “Test waiver” == “ignore known issue for a reason”
- The test has already been run, the result **is** there, we just don’t like it
- We need “lazy evaluated” exceptions to test results
- That’s in contrast to marking tests “# TODO” in advance

Example:
- Software project might not run with IPv6
- But you want to see a big SUCCESS or NO SUCCESS in an IPv4-only context
- Statically marking tests with “# TODO” requires changing back and forth
- Dynamically marking tests depending on environment conflicts with debugging the problem

Solution:
- append “# TODO explanation” to dedicated NOT OK lines, **after** you run the tests

How to patch?
- Change the **TAP::DOM**, regenerate raw TAP
TIER 6 | Test waivers

- TAP::DOM::Waivers -- match DPath, apply TAP::DOM hash merge

```perl
use TAP::DOM;
use TAP::DOM::Waivers 'waiver';
$dom = TAP::DOM->new( tap => "somefile.tap" );
$waivers = [
    # a description of what the waiver is trying to achieve
    comment => "Force all IPv6 stuff to true",

    # a DPath that matches the records to patch:
    match_dpath => [ 
        
        # apply changes to the matched records,
        # here a TODO with an explanation:
        patch => { 
            is_ok => 1,
            has_todo => 1,
            is_actual_ok => 0,
            explanation => 'waiver for context xyz',
            directive => 'TODO',
        },
    ],

    $patched_dom = waiver($dom, $waivers);
print NEWFILE $patched_dom->to_tap;
```
**TIER 6 | Test waivers**

- **TAP::DOM::Waivers** -- meta patches, common cases like "TODO" or "SKIP"

```perl
$waivers = [  
    {  
        comment => "Force all IPv6 stuff to true",
        match_dpath => [ "//lines//description[value =~ 'IPv6']/.." ],
        metapatch => { TODO => 'waiver for context xyz' },
    },
];
```
TIER 6 | Test waivers

- TAP::DOM::Waivers -- meta patches, very often you match by description

```
$waivers = [
    [
        'comment' => "Force all IPv6 stuff to true",
        'match_description' => [ "IPv6" ],
        'metapatch' => [ 'TODO' => 'waiver for context xyz' ],
    ],
];
```
FINAL TIER
FINAL TIER | Nested TAP

- Nested TAP

  TAP Version 13
  1..2
  ok 1 - established connection
    1..3
      ok 1 - step 2-1
      ok 2 - step 2-2
      ok 3 - step 2-3
    ok 2 - transfer completed (summary of nested 2-x lines --> backwards compatible)

- Rerun Tier 1 to 6 with nested TAP
- Left as an exercise for the audience
FINAL TIER | Nested TAP

- Nested TAP

  TAP Version 13
  1..2
  ok 1 - established connection
    1..3
    ok 1 - step 2-1
    ok 2 - step 2-2
    ok 3 - step 2-3
  ok 2 - transfer completed (summary of nested 2-x lines --> backwards compatible)

- Rerun Tier 1 to 6 with nested TAP
- Left as an exercise for the audience

- The End.
The End.
Trademark Attribution

AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.

©2011 Advanced Micro Devices, Inc. All rights reserved.