expect-lite

Automation for the rest of us

Craig Miller
Ciena
cvmiller@gmail.com

Abstract

Developers can always use a tool that will save money and keep the boss happy. Automation boosts efficiency while skipping drudgery. But how to implement automation for the rest of us without slowing down the real work of developing software? Introducing expect-lite. Written in expect, it is designed to directly map an interactive terminal session into an automation script. As easy as cutting and pasting text from a terminal window into a script, and adding '>' and '<' characters to the beginning of each line with advanced features to take you further. No knowledge of expect is required!

In this paper, you'll get an introduction to expect-lite, including applications where complex testing environments can be solved with just a few lines of expect-lite code. Although expect-lite is targeted at the software verification testing environment, its use is not limited to this environment, and it has been used world-wide for several years in router configuration, Macintosh application development, and FPGA development. expect-lite can be found at: `http://expect-lite.sf.net/`

1 Introduction

expect-lite was born out of the need to boost efficiency and skip drudge work. While written in Expect, no knowledge of expect is required. In fact, expect-lite was created to avoid scripting in Expect.

Because Expect is an interpretive language, quite often there are lines which are not executed, except when something goes wrong. There is a chance that a typo, or some other error may cause the Expect script *crash*, printing a call trace. If that script is part of a larger framework, it is possible the overnight regression, for example, will come to a screeching halt.

expect-lite avoids this problem in its simplicity. By using simple characters such as > and <, it is near impossible to have a typo, or syntax error.

2 Problem

The problem is that developers need to test software, without having to think about an entire automation framework and yet another specialized language. expect-lite was developed out of a need to automate testing, and make it as easy as possible.

2.1 History

expect-lite is based on the industry standard Expect language, a TCL extension written by Don Libes of NIST (National Institute of Standards and Technology in the U.S.). It was used and improved internally from 2005 to 2007 when Freescale Semiconductor graciously allowed it to become an open source project. In 2008, over 24,000 lines of expect-lite code were running daily as part of a nightly regression.

2.2 Licensing

Because of the open nature of the foundation software TCL (BSD-Style License) and Expect (Public Domain), this open source project is licensed under the BSD-Style License. Any modifications need not be re-integrated into the open source project. Even commercial products can be created using this software, as long as the copyright header remains intact.

This seemed like the right thing to do, since the underlying software is licensed under less restrictive terms.

3 Features

expect-lite has been expanded over its five years of use, starting with the simple **send** and **expect** constructs ('>' and '<' respectively), and adding additional functionality, such as if statements, looping, multi-session support, and instant debugging.

The following subsections will high-light some of the more important features which give expect-lite the power to solve complex automation problems with minimal lines of script.

3.1 Remote Login

expect-lite was born into a multi-host environment, which created an early requirement to remote login to a host (usually a Linux machine). The remote host was allocated by a sharing facility such as the commercial package, LSF. Additionally, the remote host must be specified on the command line where the script will wake up and begin executing. Three methods of remote login are supported:

1. telnet
2. ssh with password
3. ssh with keys (no password)

ssh with keys is the preferred method, since no password is required, however this requires some environment setup before hand.

Even when no remote host is required, it is best to log into the localhost (shown in yellow in Figure 1) since the underlying Expect has problems with synchronization (between send and expect strings) when a remote host is not specified. Therefore it is possible to setup the localhost with ssh keys using the provided shell script *setup_local_ssh.sh*.

3.2 Special Character Sequences

expect-lite interprets special characters at the beginning of each line in the script as shown in Table 1.

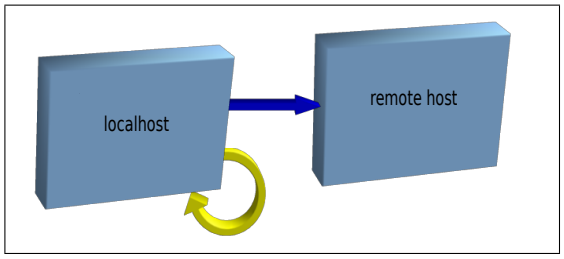

Figure 1: Connection to remote host (blue) and localhost (yellow) .

Table 1: Special Characters

Char	Action
>	send string to the remote host
>>	send string to remote host, without waiting for prompt
<	string/regex MUST be received from the remote host in the alloted timeout or the script will FAIL!
<<	literal string MUST be received (similar to 'lessthan' without regex evaluation)
-<	if string/regex IS received from the remote host the script will FAIL!
#	used to indicate comment lines, and have no effect
;	are also used to indicate comment lines, but are printed to stdout (for logging)
;;	similar to above, but no extra newlines are printed (useful for printing script help)
@num	changes the expect timeout to number of seconds
$var=	static variable assignment at script invocation
+$var=	dynamic variable assignment
+$var	increment value of $var by 1 decimal
-$var	decrement value of $var by 1 decimal
=$var	math functions, perform bitwise and arithmetic operations: << >> & | ^ * / % + -
!	indicates an embedded expect line
?	c-style if/then/else in the format ?cond?action::else_action
%	label - used for jumping to labels
~filename	includes a expect-lite automation file, useful for creation of common variable files, or 'subprograms/subroutines'

3.3 Regular Expression (RegEx) Evaluation

The full power of TCL/Expect RegEx is available to expect-lite. Thus permitting complex expect statements,

with the following limitations:

1. Support of RegEx in standard expect (e.g. including anchors, char-classes and repeats)
2. Support of RegEx meta characters (e.g. \t \n are supported, \d is not)
3. expect-lite only evaluates lines using RegEx which begin with '<' '-<' and '+' (dynamic variables)

3.4 Static and Dynamic Variables

Variables are always prefaced with the $. Variables bound at script invocation are considered static, and those which are bound during execution of the script are dynamic. Static variables are assigned with the $ as the first character of a line, such as:

```
$five=5
```

Dynamic variables are assigned using RegEx capture and begin with +$. For example, text output from a command may captured into a variable, such as:

```
>hostname
+$host=\n([a-z]+)
```

If, however, the dynamic var is not successfully captured, expect-lite will print:

```
Assigned Var: \
   somevar=__NO_STRING_CAPTURED__
```

3.5 Constants

Constants are a very powerful feature of expect-lite. Constants permit writing a generalized script and overriding internal variables in the script with command line specified constants. They are are immutable, and cannot be changed. For example the constant local_eth can be defined on the command line as:

```
expect-lite remote_host=remote-host-018 \
   cmd_file=basic_ping_test.elt \
   local_eth=eth2
```

All references to $local_eth in the script will be set to eth2. This allows one to change the behaviour of the script without requiring a script change.

3.6 Plays well with Bash

Although not limited to working with bash, bash is invoked upon logging into the remote host, and therefore will be discussed more here.

Since the bash shell is well documented, and supported, it can be leveraged to assist in expect-lite's limitations such as looping and branching. A simple bash loop inside expect-lite can be created for example:

```
$set=1 2 3 4 5
>for i in $set #expanded by expect-lite
>{
>echo $i #expanded by bash
>}
```

In the above example, $set is an expect-lite variable, not a bash variable. An expect-lite variable is always declared before its use (e.g. $set = 1 2 3 4 5). If a variable cannot be dereference by expect-lite it is passed to the shell. The loop will execute after the final "}" line is sent to the remote-host.

Using Bash to create executable expect-lite scripts, it is possible to make expect-lite scripts executable, with the help of a small embedded bash helper script. Insert the following at the top of the expect-lite script:

```
#!/usr/bin/env bash
# make this auto-runnable!
# Little bootstrap bash script to run \
#  kick start expect-lite script
# Convert --parms to expect-lite \
#  param=value format
PARAMS=`echo $* | /bin/sed -r \
's;--([a-z]+) ([0-9a-zA-Z]+);\1=\2;g'`

echo $PARAMS
expect-lite r=none c=$0 $PARAMS
exit $?
\
```

The bash scriptlet does the following:

- Converts command line arguments from –arg value to arg=value format, making the script more linux-like

- Call the expect-lite script with default arguments (in this example that is r=none) and converted arguments
- Exit with the same exit code of the expect-lite script (0 success, 1 failure)

expect-lite ignores the bash scriptlet when invoked, because none of the lines begin with expect-lite special characters.

3.7 Include Files

Include files are a quick way to develop script snippets which can be included into larger scripts or to include a common variable file. When an include file is executed, it is as if the file were just pasted into the script file, and therefore has access to the variable space of the main script, and can modify that variable space as well. In this example, a common variable file is "sourced":

```
# Source common variable file
~asic_vars.inc
```

Common functions, such as telnet'ing to the DUT, are a good use of include files:

```
; === Connect to DUT
~dut_connect.inc
```

Include filenames can also be assigned in a variable, such that the file names can be declared at the top of the script but used later within the script. For example:

```
# Source Var file to be used
$asic_include=asic_vars.inc
...
~$asic_include
```

3.8 Not Expect

A test can also fail should text appear that is unexpected (such as an error). This does not clear the expect input buffer, and should be used before a valid expect. How long should the script wait for the un-expected? In order to reduce delays in script running time, the Not Expect feature only waits for 100ms. This is usually enough time to detect quick error responses such as "file not found".

For example: Fail device doesn't exist

```
>ls -l /dev/linux
-<No such file
```

3.9 If Statement

Conditional (if/then/else) statements are natively supported in expect-lite. The conditional uses a c-style syntax with a question mark (?) at the beginning of the line, and double colon to indicate the else statement, using the format ?cond?action::else_action

Although spaces are not required around the conditional characters (? and ::), it is recommended for ease of reading. The comparison operators are: '==', '!=' ,'>=', '<=', '>' and '<'. If the compared values can be evaluated as a numbers, then larger and less than will yield expected results. A simple conditional example:

```
$age=56
?if  $age >= 55 ? \
  >echo "freedom at 55!" :: \
  > echo "keep working!"
```

In the above example if $age is larger than or equal to 55 then the action 'echo "freedom at 55!". If $age is less than 55 then the action 'echo "keep working!"' will be sent.

3.10 Looping with Conditionals & Labels

Conditionals are limited to a single line. Sometimes this is too limiting, as it would be nice to have several commands be executed based on the success of a conditional. To support this, the concept of labels has been introduced. A label is defined as having the first character a '%'. Although the label line itself does nothing, it provides a location to the conditional to Jump To Label.

Simple looping is now supported by allowing jump to label backwards. The Repeat Loop is the easiest loop to implement:

```
; ======== Incrementing Loop ========
$max=5
$count=3
%REPEAT_INC_LOOP
```

```
    >echo $count
    # increment variable
    +$count
?if $count <= $max ?%REPEAT_INC_LOOP
```

Because of jump to label can jump backwards, it is important to assign unique looping labels, such as %REPEAT_INC_LOOP. Unexpected results will occur if non-unique loop label names are used. Non-looping labels, as illustrated in the previous section, are not required to be unique.

Also included in the above example is incrementing an expect-lite variable: +$count This will add 1 to the value of $count. If $count is not an integer, the value of $count will remain unchanged (can't add 1 to a string).

As part of the looping enhancement, there is infinite loop protection. The maximum amount of looping is defined in expect-lite itself with the variable _el_infinite_loop. This value is decremented with each iteration of all loops for the entire script. Typically this would be in the range of 100 to 1000 to be safe. For example, if a complex expect-lite script had 4 loops each with 100 iterations, the _el_infinite_loop should be set larger than 400.

3.11 User Defined Prompt */prompt/

With each '>' command an implied "wait for prompt" occurs. The predefined prompts are based on standard shell prompts (>%$#). However, it is quite possible that expect-lite will not be interacting with a shell, but another application (such as gdb) or device which does not issue a shell-like prompt.

As of version 3.1.5, the user may define a custom prompt using the following command:

```
*/my_prompt /
```

The succeeding '>' lines will interpret a prompt as the standard shell prompts and 'my_prompt '. The user defined prompt definition between the slashes is interpreted as regex, and therefore regex rules such as escaping applies. For example, to set a user defined prompt for gdb the following would be used:

```
*/\(gdb\) /
```

Only one (1) user defined prompt can be active at a time, however multiple prompts can be represented using the regex OR '|', for example, creating a gdb prompt and my_prompt:

```
*/\(gdb\) |my_prompt /
```

Clearing a user defined prompt. There may be times where a previously user defined prompt is causing problems by output which falsely triggers the user defined prompt. It is possible to clear the user defined prompt by:

```
*//
```

The scope of the user defined prompt is global, that is, it extends to the main script as well as all include files referenced.

3.12 Multiple Sessions *FORK

Until version release 3.5.0, expect-lite has been intentionally limited to a single session keeping it simple. However, there are certain environments where it might be advantageous for a single script to control/monitor both a client (e.g wget) and a server (httpd log). Without multiple session support this type of environment would be difficult to automate with expect-lite.

What is a new session? A new session starts with a new shell on the remote host. All commands '>' and received text '<' are constrained to that session. It is possible to use multiple sessions on the localhost using r=none, however r=none support is limted due to timing issues. If multiple sessions on the localhost (the same host where expect-lite is running), it is better to ssh to the localhost by specifying r=localhost. This loop back method ensures that the timing between commands and received text stay in sync.

With the version of 3.5.0, expect-lite supports nearly limitless multiple sessions. However, it is a good guideline to use as few sessions as needed. For backward compatibility, the first session is the "default" session. Additional sessions are assigned a name at invocation with the *FORK <session_name> directive:

```
*FORK Server
INFO: FORK session is: Server
```

expect-lite will print an "INFO" line stating the current session name Session names may not contain spaces, and must be unique for each session. To switch back to a previously started session, re-use the session name. The session name "default" (without quotes) is reserved for the first session.

```
*FORK default
INFO: FORK session is: default
```

Print the current session name by using the *FORK with no session name:

```
*FORK
INFO: FORK session is: Server
```

*FORK and variables. It is possible to assign a session name to a variable, and then create a new session with that name. For example:

```
$my_session=Client
*FORK $my_session
INFO: FORK session is: Client
```

All sessions will be automatically closed when the script terminates.

4 Debugging your scripts

Although expect-lite is designed to implement automation as easily and quickly as possible, it is possible that by using one of the more complex features, some script debugging may be required.

4.1 Interact

Interact may be the quickest, easiest, and overall best debugging aid. Interact is a mode which turns control of the keyboard over to the user, so that one may type directly to the process on the remote host With version 3.5.0 there are two methods to invoke Interact: programmatic, and instant-interact.

Programmatic Interact is called in the script with the following command:

```
*INTERACT
```

expect-lite will pause at this point in the script, and connect the keyboard to the remote session (which may be at a prompt). Any command may be entered and responses observed. Typing '+++' (3 pluses) will return control to the script, and it will continue. This is very helpful for automation assist, allowing the script to perform complicated setup commands, before turning control over to the user for an interactive session.

The other method is instant-interact. This feature requires a tcl package TclX to be installed, and will automatically be enabled when the package is present. With the feature enabled, the user can press '^\' (control+backslash) at anytime and enter Interact. This is the easiest and fastest way to debug a script.

4.2 Debugging with the el_shell

To make debugging of scripts even easier, both methods of interact support a limited expect-lite "shell". In this "el_shell", expect-lite script commands can be typed, it is possible to even assign variables inside the paused script, for example:

```
$MYVAR=today

*SHOW VARS
Var:0     Value:0
Var:MYVAR Value:today
Var:TEST  Value:/proc/cpuinfo

>>pwd
pwd
/home/user
```

In the previous example, $MYVAR is assigned, all variables their respective values are displayed, and the 'pwd' command is sent.

5 General Tips for writing scripts

Although expect-lite is designed to be simple, there are a few things to watch out for when writing a script. Here are some simple tips which will make script writing go more smoothly:

1. Use reasonable timeouts, if 30 seconds is needed to get a response, set the timeout at 45 or 60 seconds, not 600.

   - There is no cost to changing the timeout, timeout values can also be variables

2. Beware of expect-lite using regex, when creating lines such as: `<0.005 secs (5 micro secs)`

   - The parentheses is used by the regex engine, instead escape these characters: `<0.005 secs \(5 micro secs \)`
   - or use '<<' which does not use regex, and does not require escaping: `<<0.005 secs (5 micro secs)`

3. Use the expect character '<' or '<<' often. Check for valid results when possible. A script which expects nothing will never fail!

4. Use printable comments ';' often. Think of it as writing a note to oneself, it will make reading log files much easier.

6 Example Script

Problem: you need to write an automated script which monitors an ethernet port for incoming packets (using tcpdump) requiring sudo privilages, while generating network traffic (a web request).

Solution: generate a request with wget and use tcpdump to verify the return packet

In this script, two sessions are created with the `*FORK` command, one for the packet capture, and another for the wget request. The tcpdump output is filtered with egrep to remove packets of non-interest, and limit the hex packet output to 8 lines. Using two sessions permits the monitoring of network traffic while generating the wget request.

After setting up the monitor, and generating the wget request, the script switches back to the `tcpdump` session to verify the client and server id strings. If the strings are correct, then the script prints texttttOverall Result: PASS.

```
#
#       OLS 2010 Example Script
# Script uses sudo, and tcp dump to
# monitor network interface while
# initiating web traffic with wget
#
# set timeout to 10 seconds
@10
$SUDO_PASS=mysudopassword
$WEB_SERVICE=http://www.w3.org/

# start packet capture
*FORK tcpdump
>echo $SUDO_PASS | sudo -S tcpdump \
 -i eth0 -s 256  -e -X portrange 80 \
  | egrep -A 8 'www'
# force expect-lite to wait
# until tcpdump is ready
<\nlistening on
<link-type

# initiate web request with wget
*FORK wget
; === get page without saving a copy, \
    show HTTP headers
>wget -S --spider $WEB_SERVICE
# verify server in wget output
<<Server: Apache
>

# check packets captured
*FORK tcpdump
# verify client
<User-Agent
<Wget
# verify server
<Server
<Apache/2
>>^C

#*INTERACT
>>
>
```

Note near the `*INTERACT` command near the end of the script. It was used to initially pause the script inside the `tcpdump` session during development and is now commented out.

The output of the script appears in Figure 2.

7 Summary

Creating automation scripts is as easy as cutting and pasting text from a terminal window into a script, and adding '>' and '<' characters to the beginning of each line. Advanced features take you even further. No knowledge of expect is required!

Although expect-lite is targeted at the software verification testing environment, it is not limited to this environment, and has been used world-wide for several years in router configuration, Macintosh application development, and FPGA testing.

expect-lite really is *Automation for the rest of us*.

Figure 2: Script output

```
cvmiller@sawtoothy:~/Expect-lite$ ./expect-lite r=halaconia c=test_sudo.elt

INFO: Instant-Interact '^\' feature is enabled

spawn ssh halaconia

Setting Expect Timeout to: 10

INFO: FORK session is: tcpdump, Active sessions are: default tcpdump

cvmiller@halaconia:~$
cvmiller@halaconia:~$
cvmiller@halaconia:~$ export PS1='$ '
$ set prompt = "$ "
$ echo mysudopassword | sudo -S tcpdump -i eth0 -s 256 -e -X portrange 80 | egrep -A 8 'www'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 256 bytes


INFO: FORK session is: wget,    Active sessions are: default tcpdump wget

 === get page without saving a copy, show HTTP headers

cvmiller@halaconia:~$
cvmiller@halaconia:~$ export PS1='$ '
$ set prompt = "$ "
$ wget -S --spider http://www.w3.org/
--11:02:01--  http://www.w3.org/
           => `index.html'
Resolving www.w3.org... 128.30.52.51, 128.30.52.53, 128.30.52.54, ...
Connecting to www.w3.org|128.30.52.51|:80... connected.
HTTP request sent, awaiting response...
  HTTP/1.1 200 OK
  Date: Mon, 03 May 2010 15:02:01 GMT
  Server: Apache/2
  Content-Location: Home.html
  Vary: negotiate,accept
  TCN: choice
  Last-Modified: Mon, 03 May 2010 14:51:30 GMT
  ETag: "6dfe-485b1ba692080;89-3f26bd17a2f00"
  Accept-Ranges: bytes
  Content-Length: 28158
  Cache-Control: max-age=600
  Expires: Mon, 03 May 2010 15:12:01 GMT
  P3P: policyref="http://www.w3.org/2001/05/P3P/p3p.xml"
  Connection: close
  Content-Type: text/html; charset=utf-8
Length: 28,158 (27K) [text/html]
200 OK

$
```

```
INFO: FORK session is: tcpdump, Active sessions are: default tcpdump wget

11:02:01.093847 00:11:24:e1:db:c8 (oui Unknown) > 00:60:08:12:8f:95 (oui Unknown), ethertype
 IPv4 (0x0800), length 74: halaconia.hoomaha.net.57016 > stu.w3.org.www: S
  1982196051:1982196051(0) win 5840 <mss 1460,sackOK,timestamp 22265167 0,nop,wscale 6>
        0x0000:  4500 003c 541c 4000 4006 2740 0a01 010e  E..<T.@.@.'@....
        0x0010:  801e 3433 deb8 0050 7625 e953 0000 0000  ..43...Pv%.S....
        0x0020:  a002 16d0 74ac 0000 0204 05b4 0402 080a  ....t...........
        0x0030:  0153 bd4f 0000 0000 0103 0306            .S.O........
11:02:01.130855 00:60:08:12:8f:95 (oui Unknown) > 00:11:24:e1:db:c8 (oui Unknown), ethertype
IPv4 (0x0800), length 74: stu.w3.org.www > halaconia.hoomaha.net.57016: S
3973601193:3973601193(0) ack 1982196052 win 5792 <mss 1460,nop,nop,timestamp 3243392802 22265167>
        0x0000:  4500 0038 0000 4000 3206 8960 801e 3433  E..8..@.2..`..43
        0x0010:  0a01 010e 0050 deb8 ecd8 57a9 7625 e954  .....P....W.v%.T
        0x0020:  9012 16a0 46e2 0000 0204 05b4 0101 080a  ....F...........
        0x0030:  c152 3f22 0153 bd4f de71 7b8c            .R?".S.O.q{.
11:02:01.130944 00:11:24:e1:db:c8 (oui Unknown) > 00:60:08:12:8f:95 (oui Unknown), ethertype
IPv4 (0x0800), length 66: halaconia.hoomaha.net.57016 > stu.w3.org.www: . ack 1 win 5840
<nop,nop,timestamp 22265176 3243392802>
        0x0000:  4500 0034 541d 4000 4006 2747 0a01 010e  E..4T.@.@.'G....
        0x0010:  801e 3433 deb8 0050 7625 e954 ecd8 57aa  ..43...Pv%.T..W.
        0x0020:  8010 16d0 5e66 0000 0101 080a 0153 bd58  ....^f.......S.X
        0x0030:  c152 3f22                                .R?"
11:02:01.131988 00:11:24:e1:db:c8 (oui Unknown) > 00:60:08:12:8f:95 (oui Unknown), ethertype
IPv4 (0x0800), length 165: halaconia.hoomaha.net.57016 > stu.w3.org.www: P 1:100(99) ack 1 win
5840 <nop,nop,timestamp 22265176 3243392802>
        0x0000:  4500 0097 541e 4000 4006 26e3 0a01 010e  E...T.@.@.&.....
        0x0010:  801e 3433 deb8 0050 7625 e954 ecd8 57aa  ..43...Pv%.T..W.
        0x0020:  8018 16d0 bfe9 0000 0101 080a 0153 bd58  .............S.X
        0x0030:  c152 3f22 4845 4144 202f 2048 5454 502f  .R?"HEAD./.HTTP/
        0x0040:  312e 300d 0a55 7365 722d 4167 656e 743a  1.0..User-Agent:
        0x0050:  2057 6765 742f 312e 3130 2e32 0d0a 4163  .Wget/1.10.2..Ac
        0x0060:  6365 7074 3a20 2a2f 2a0d 0a48 6f73 743a  cept:.*/*..Host:
        0x0070:  2077 7777 2e77 332e 6f72 670d 0a43 6f6e  .www.w3.org..Con
--
11:02:01.171913 00:60:08:12:8f:95 (oui Unknown) > 00:11:24:e1:db:c8 (oui Unknown), ethertype
IPv4 (0x0800), length 70: stu.w3.org.www > halaconia.hoomaha.net.57016: . ack 100 win 5792
<nop,nop,timestamp 3243392813 22265176>
        0x0000:  4500 0034 6887 4000 3206 20dd 801e 3433  E..4h.@.2.....43
        0x0010:  0a01 010e 0050 deb8 ecd8 57aa 7625 e9b7  .....P....W.v%..
        0x0020:  8010 16a0 5e28 0000 0101 080a c152 3f2d  ....^(.......R?-
        0x0030:  0153 bd58 8acc f50f                      .S.X....
11:02:01.172825 00:60:08:12:8f:95 (oui Unknown) > 00:11:24:e1:db:c8 (oui Unknown), ethertype
IPv4 (0x0800), length 529: stu.w3.org.www > halaconia.hoomaha.net.57016: P 1:460(459) ack 100
win 5792 <nop,nop,timestamp 3243392813 22265176>
        0x0000:  4500 01ff 6888 4000 3206 1f11 801e 3433  E...h.@.2.....43
        0x0010:  0a01 010e 0050 deb8 ecd8 57aa 7625 e9b7  .....P....W.v%..
        0x0020:  8018 16a0 b148 0000 0101 080a c152 3f2d  .....H.......R?-
        0x0030:  0153 bd58 4854 5450 2f31 2e31 2032 3030  .S.XHTTP/1.1.200
        0x0040:  204f 4b0d 0a44 6174 653a 204d 6f6e 2c20  .OK..Date:.Mon,.
        0x0050:  3033 204d 6179 2032 3031 3020 3135 3a30  03.May.2010.15:0
        0x0060:  323a 3031 2047 4d54 0d0a 5365 7276 6572  2:01.GMT..Server
        0x0070:  3a20 4170 6163 6865 2f32 0d0a 436f 6e74  :.Apache/2..Cont
--
11:02:01.172857 00:11:24:e1:db:c8 (oui Unknown) > 00:60:08:12:8f:95 (oui Unknown), ethertype
IPv4 (0x0800), length 66: halaconia.hoomaha.net.57016 > stu.w3.org.www: . ack 460 win 6432
<nop,nop,timestamp 22265186 3243392813>
        0x0000:  4500 0034 541f 4000 4006 2745 0a01 010e  E..4T.@.@.'E....
        0x0010:  801e 3433 deb8 0050 7625 e9b7 ecd8 5975  ..43...Pv%....Yu
        0x0020:  8010 1920 59d3 0000 0101 080a 0153 bd62  ....Y........S.b
        0x0030:  c152 3f2d                                .R?-
sending ^C

9 packets captured
9 packets received by filter
0 packets dropped by kernel
$
$
##Overall Result: PASS
```

Proceedings of the Linux Symposium

July 13th–16th, 2010
Ottawa, Ontario
Canada

Conference Organizers

Programme Committee

Proceedings Committee