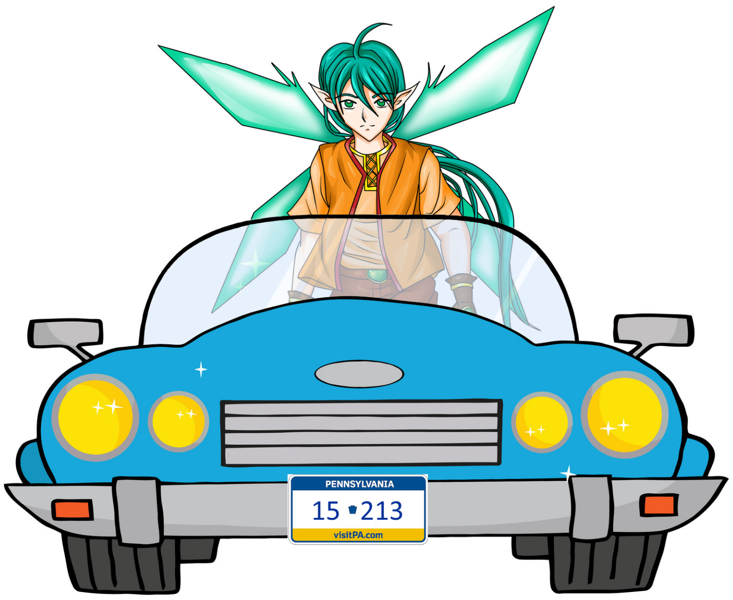

PXYDRIVE: A Proxy Testing Framework[*]

Randal E. Bryant
Carnegie Mellon University

July 25, 2018

1 Overview

PXYDRIVE (pronounced "pixie drive") provides a comprehensive testing environment for web proxy programs. It enables testing and debugging of the basic operations of a proxy, as well as the ability to support concurrent requests and to provide caching of retrieved files. An associated program PXYREGRESS runs a series of standard tests, using PXYDRIVE to perform each test. These two programs are designed to help students implement their proxies and to help graders perform automated testing. PXYDRIVE was developed to support the ProxyLab assignment, but it can be used on other proxies, as well.

[*]Copyright © 2018, R. E. Bryant. All rights reserved.

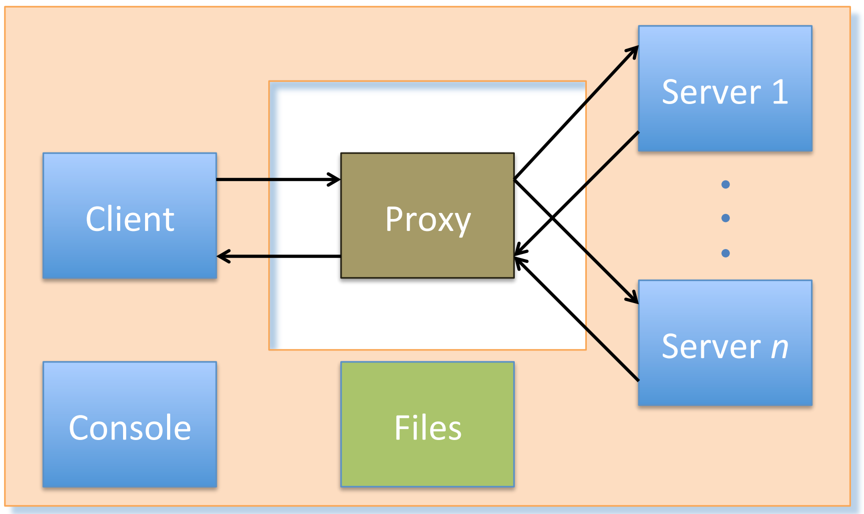

Figure 1: **Structure of PXYDRIVE** The program acts as both the client and the servers for a proxy program.

Figure 1 illustrates the overall structure of PXYDRIVE. As shown, it includes a number of components:

- An external executable *proxy* program. It is executed as child process of PXYDRIVE.
- A web *client* that can issue requests for files and then compare the responses to original copies of the files.
- One or more *servers* that can respond to requests for files
- A set of *files* shared by the client and the servers
- A *console* providing a simple command-line interpreter to perform tests and to observe results.

This structure provides a number useful features for testing a proxy program:

- By including all of these components in a single program, PXYDRIVE has very detailed control over the sequencing of interactions with the proxy.
- By having files shared between the client and the servers, the client can check that the results it receives match those sent by the servers.
- By observing a transaction from both the client's and the server's side, it can provide useful information to the user about all of the communications that take place.
- The ability to observe both server and client communications enables PXYDRIVE to check that the proxy complies with any formatting requirements for HTTP messages.

2 Running PxyDrive

PxyDrive is written in Python, and so is invoked as `./pxydrive.py` from its local directory. (You can also run it from another directory by giving an appropriate path.) PxyDrive has been tested on both Linux and Mac OS X. Its ability to run on other platforms is unknown.

Command-Line Arguments

PxyDrive supports the following command-line arguments:

`-h`
: Print usage information

`-p` *PROXY*
: Run specified proxy program. For more details, consult the documentation of the `proxy` command.

`-f` *FILE*
: Read commands from *FILE*. For more details, consult the documentation of the `source` command.

`-l` *FILE*
: Write inputs and outputs to *FILE*. For more details, consult the documentation of the `log` command.

`-S`
: Enforce strict formatting rules for HTTP request message headers. For more details, consult the documentation of the `strict` option.

PxyDrive creates two subdirectories of the directory in which it is invoked: `source_files`, in which generated files are stored, and `response_files`, in which files received by the client are stored. These files can be examined when some transaction generates an error. Old versions of these files are deleted every time PxyDrive starts.

Once started, the user directs PxyDrive to perform a series of tests, either by typing commands directly to the program or by reading the commands from a file.

Proxy Requirements

PxyDrive is designed to tests proxy programs that meet the requirements specified in ProxyLab. The lab description specifies several important aspects of the request headers sent by the proxy to a server:

- There must be a `Host` header
- There must be `User-Agent`, `Connection` and `Proxy-Connection` headers with specific values.
- Any other request headers that are sent by the client should be forwarded unchanged.

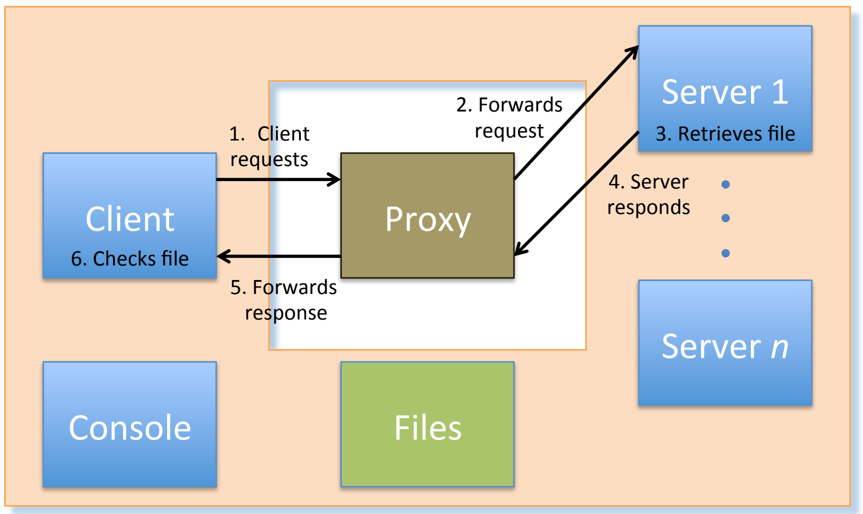

Figure 2: **Steps in a transaction** A transaction involves a *request* (steps 1–3) and a *response* (steps 4–6)

The latter requirement is especially important, because PXYDRIVE uses information passed in headers to track the interactions between clients and servers.

PXYDRIVE can operate in two different modes. In *strict* mode (set with either the command-line argument `-S` or the `strict` option), these requirements are strictly enforced. Error messages will be printed for request messages that do not comply. In *non-strict* mode (the default), the program doesn't require these headers. Instead, it does its best to track the interactions heuristically. Even in non-strict mode, it works best when the proxy forwards information placed in the request headers to the server.

3 PXYDRIVE Operation

A *transaction* by PXYDRIVE consists of all of the steps required for the client to get a file from a server via the proxy and ascertain its correctness. A transaction has two different phases: the *request* by the client, followed by the *response* by the server. Each transaction can be divided into six steps—three for each phase, as is illustrated in Figure 2:

Request

1. The client sends a `GET` request to the proxy. This request includes a URL, identifying the host server and the file name.
2. The proxy forwards the request to the designated server.
3. The server receives the request, locates the file, and formulates a response message.

Response

```
1 serve s1                          # Set up server
2 generate random-text.txt 10K      # Generate text file
3 request r1 random-text.txt s1     # Initiate request r1
4 delay 100                         # Wait for request to propagate
5 respond r1                        # Allow server to respond
6 delay 100                         # Wait for response to propagate
7 check r1                          # Compare retrieved file to original
```

Figure 3: **Using PxyDrive to perform transaction with separate request and response.** Delays must be included to allow the proxy, the server, and the network to operate.

4. The server sends the response message to the proxy.
5. The proxy forwards the response to the client.
6. The client receives the requested file and compares it to the original one.

PxyDrive gives fine-grained control over transactions, providing separate commands to trigger the request phase and the response phase. This feature makes it possible to test important features of the proxy, including its ability to handle concurrent transactions and to cache previously retrieved files.

The following simple examples illustrate how the user can control PxyDrive operation. These examples all consist of commands that can be supplied to the command-line interface. Observe that '`#`' is a comment character—it and anything to the right on the same line are ignored.

Figure 3 shows a session having a single transaction, with separate commands to trigger the request and the response. (For this session, the proxy was specified as a command-line option.) The lines are as follows:

1. A server is set up and given name `s1`. (It is possible to have multiple servers, each with a unique name.)

2. A file, consisting of 10,000 random ASCII characters is generated. (The extension `.txt` indicates that the file should contain only ASCII characters.) The suffix '`K`' indicates that the number of bytes should be multplied by 1,000.

3. The client makes a request for this file from the server. The transaction is given the identifier '`r1`.' Transaction identifiers must be unique, because they provide the mechanism by which PxyDrive tracks the progress of transactions.

4. The program waits for 100 milliseconds, giving time for steps 1–3 of the transaction to occur. Explicit delays must be included whenever the client, proxy, or server must complete some operations before the next command can be performed.

5. The server is instructed to respond to the request. Note the use of the transaction identifier to indicate the specific request.

6. The program waits for 100 milliseconds, giving time for steps 4–6 of the transaction to occur.

```
1 serve s1                          # Set up server
2 generate random-data.bin 10K      # Generate binary file
3 fetch f1 random-data.bin s1       # Fetch file
4 delay 100                         # Wait for request and response
5 check f1                          # Compare retrieved file to original
```

Figure 4: **Using PxyDrive to perform transaction with immediate response.** Only one delay is required.

```
1 serve s1                          # Set up server
2 fetch f1 nonexistent.file s1      # Fetch file
3 delay 100                         # Wait for request and response
4 check f1 404                      # Check for not-found status
```

Figure 5: **Using PxyDrive to perform transaction with missing file.** The check confirms status code 404.

7. The program checks that the file was received correctly by the client. An error message will be printed if either the server could not find the file, the file was corrupted, or the client did not receive the reponse.

Figure 4 also performs a single transaction, but it illustrates several additional features of PxyDrive. In line 2, the generated file is given the extension `.bin`. This causes the generated file to contain arbitrary bytes. Such a file will test the ability of the proxy to properly handle binary data. In addition the `fetch` command of line 3 indicates that PxyDrive should perform a complete transaction, with the server responding immediately to the request. Only one delay command (line 4) is required, since all six steps of the transaction will take place. Running a session consisting of a mix of fetches and requests requires the proxy to handle a variety of event timings.

Figure 5 illustrates how PxyDrive can be used to test the ability of the proxy to handle transactions involving exceptional conditions. It shows an attempt to retrieve a file that does not exist. The server will respond to this request with a message having status code 404, indicating that the file was not found. The proxy should forward this message to the client. The optional argument `404` on line 4 indicates that the program should check for this status code in the response to the client.

These three examples illustrate some of the capabilities of PxyDrive to run tests of a proxy. Section 5 provides additional examples showing how to test different features of a proxy.

4 Commands

The following documents all of the commands supported by PxyDrive. These are divided into different categories, according to when and how they get used.

In the following, square brackets enclose optional arguments, while the notation *ARG*+ means one or more repetitions of argument *ARG*. The notation (*ARGA*|*ARGB*) indicates a choice between arguments *ARGA* and *ARGB*.

Set Up

The following commands are used at the beginning of a session to establish the environment for a set of transactions.

`proxy` *PATH* **[*ARG*+]**

Execute proxy program stored as executable file *PATH*. Any remaining arguments are passed to the proxy as command-line arguments. (There is no need to pass a port number—this is supplied by PXYDRIVE.) Any results printed by the proxy to either standard output or standard error will be echoed to the PXYDRIVE standard output and log files. If some proxy is already running, it will be terminated before starting the specified proxy.

`serve` ***SID*+**

Set up server(s) with named identifier(s). These will be assigned ports chosen by PXYDRIVE. Server identifiers must be unique.

`generate` ***NAME*`.`(`txt`|`bin`) *BYTES***

Generate file with random content. Files with names of the form *NAME*`.txt` will contain random ASCII characters, formatted into lines of length 80. Files with names of the form *NAME*`.bin` will contain random, unformatted bytes. Argument *BYTES* indicates the number of bytes in the file. As a shortcut, suffix ‘`K`’ multiplies the number of bytes by 1,000, and suffix ‘`M`’ multiplies the number of bytes by 1,000,000.

Transactions

The following commands control the progress of transactions and observe their results.

`fetch` ***ID FILE SID***

Fetch file from server (i.e., the request will be passed to server, and the server will be allowed to respond immediately.) *ID* is an identifying name for the transaction. It must be unique. Unless testing for a “not found” response, *FILE* should be one of the files accessible to the server (e.g., one generated with the `generate` command). *SID* identifies which server to use.

`request` ***ID FILE SID***

Request file from server (i.e., the request will be passed to server, but the server will be not be allowed to respond.) *ID* is an identifying name for the transaction. It must be unique. Unless testing for a “not found” response, *FILE* should be one of the files accessible to the server (e.g., one generated with the `generate` command). *SID* identifies which server to use.

`respond` ***ID*+**

Allow the server(s) to respond to request(s) specified by the identifier(s). An error message will be printed if the server has not yet received this request.

`check` *ID* [*CODE*]
: Check that transaction *ID* generated the proper status code. The default code is 200, indicating that the file was successfully retrieved. Additional status codes include 400 (bad request), 404 (not found), 501 (not implemented), and 505 (unsupported HTTP version), as are documented in Figure 11.25 of CS:APP.

`trace` *ID*+
: Trace the histories of the specified transactions. This will print information about the six transaction steps illustrated in Figure 2, including copies of the HTTP headers. This command can be very useful when debugging a proxy or a test sequence.

Runtime Control

The following commands control the operation of the program.

`delay` *MS*
: Delay for *MS* milliseconds.

`log` *FILE*
: Copy outputs to *FILE*. Inputs are also echoed to the file if the `echo` option is set. (This is the default.)

`source` *FILE*
: Read commands from *FILE*.

`option` *OPTION VALUE*
: Set a runtime option. Supported options are documented below.

`delete` *FILE*
: Delete file *FILE*. Files generated during previous runs of PXYDRIVE are automatically deleted every time the program starts, and so explicit deletion of files is only required for running special tests, or to avoid leaving large files around after a test has completed.

`help`
: Print a summary of the commands and options.

`quit`
: Exit the program. This may take awhile, since multiple threads must be terminated. If the program hangs up indefinitely or causes an unexpected exception, it may be necessary to explicitly kill the PXYDRIVE process.

Options

The following options are supported. For Boolean options, 1 indicates true, and 0 indicates false.

autotrace (Boolean, default = 0)
: Trace every request for which a check fails. This mode is helpful for debugging, because it causes detailed information about each failed check to be printed.

```
1 serve s1
2 generate random-text1.txt 2K      # File 1
3 generate random-text2.txt 4K      # File 2
4 request r1 random-text1.txt s1    # Request r1
5 request r2 random-text2.txt s1    # Request r2
6 delay 100
7 respond r2                        # Respond out of order
8 delay 100
9 check r2
```

Figure 6: **Using PXYDRIVE to perform transaction requiring concurrent proxy.** A serial proxy would be held up waiting for the server to respond to request `r1`.

echo (Boolean, default = 1)
: Echo inputs to log files. This makes it possible to keep track of which commands were being executed.

error (Integer, default = 5)
: The maximum number of errors that can occur before the program aborts.

stretch (Integer, default = 100):
: Adjusts the length of each delay. Given a command "**delay** d," and given a stretch value of s, the actual delay will be $d \cdot s/100$ milliseconds. Setting the stretch to a value greater than 100 lengthens all delays, while setting it to a value less than 100 shortens them. This provides a way to adjust all of the delays in a test sequence without editting all of the delay commands in a file. It can be useful in detecting cases where tests are just on a border between success and failure due to event timings.

strict (Boolean, default = 0):
: Apply strict formatting rules to HTTP request headers. As described in the ProxyLab writeup, request messages are required to include headers with settings for `Host`, `Connection`, and `Proxy-Connection`. In addition, the proxy should pass along any other headers defined in the request message from the client.

5 Testing Proxy Capabilities

The following are some examples demonstrating how the fine-grained control provided by PXYDRIVE allows testing of different features of a proxy. Additional examples can be found in the standard tests described in Section 6.

Figure 6 illustrates the ability of PXYDRIVE to ascertain that a proxy supports concurrent transactions. In this test, two requests—`r1` and `r2`—are made (lines 4–5), but the server only responds to `r2` (line 7). A sequential proxy will hang up waiting for the response for `r1`, never forwarding `r2` to the server. The check on line 9 will therefore fail, because transaction `r2` does not complete. A concurrent proxy can handle multiple transactions simultaneously, and so it will forward `r2` to the server and send the response back to the client, even while waiting for the server to respond to `r1`

```
serve s1
generate random-text.txt 10K
fetch f1 random-text.txt s1    # Handled by server
delay 100
check f1
request r1 random-text.txt s1  # Handled by proxy
delay 100
check r1                       # No action required by server
```

Figure 7: **Using PxyDrive to perform transaction requiring caching.** A caching proxy will respond to `r1` without any action by the server.

Figure 7 illustrates the ability of PxyDrive to ascertain that the proxy caches previous results. It first performs transaction `f1`, retrieving file `random-text.txt`. It then issues a request for the same file (transaction `r1`). If the proxy has the file it its cache, it will shortcircuit the transaction and return the file directly to the client, skipping steps 2–4 of Figure 2. The server never receives the request and therefore need not be instructed to respond to it.

6 Standard Tests

The directory `tests` contains a set of 32 command files testing many aspects of a proxy. These are divided into three *series*, named “A”, “B”, and “C” based on the first letter of their file names.

The A series (12 files) tests attributes required of all proxies, including ones that operate sequentially. These include the ability to handle both text and binary files, to correctly handle exceptional cases, and to support sequences of transactions involving multiple servers.

The B series (10 files) tests attributes required of a concurrent proxy. These include cases where the reponses occur in a different order than the requests, and ones with many transactions, having the potential to cause race conditions.

The C series (10 files) tests attributes required for a caching proxy. These include the ability to cache text and binary files, as well as responses for nonexistent files. It tests the cache policies on maximum file size, maximum total cache size, and eviction rules. These tests require a concurrent proxy.

Of course, even passing these 32 tests does not ensure complete correctness of a proxy program. Errors involving incorrect synchronization are especially difficult to detect in any testing regime.

Regression Testing with PxyRegress

The program PxyRegress automates the process of running a subset of the standard tests using PxyDrive. From its local directory, it is invoked as: `./pxyregress.py -p` *PROXY*, where *PROXY* is the path to the file containing the executable proxy program. Command-line options include

`-h`

Print usage information

`-s` *SERIES*
Specify which series of tests to perform. By default, all three series (A, B, and C) are tested. This argument allows finer control, specified as a string consisting of some combination of the characters '`A`', '`B`', and '`C`'. For example, the command-line option `-s A` will run only the A-series tests.

`-t` *SECS*
Set a timeout limit of *SECS* seconds for each test. Any test exceeding that limit will be marked as having failed. The default limit is 60 seconds. Setting the limit to 0 indicates that the test should be allowed to run indefinitely.

`-l` *FILE*
Copy program outputs to *FILE*.

`-S`
Enforce strict formatting rules for HTTP request message headers.

PXYREGRESS stores the outputs generated by each of the tests in the `logs` subdirectory.

7 Additional Notes

- The results of PXYDRIVE (and therefore PXYREGRESS) are nondeterministic, due to the variable delays of network communications and proxy actions. From one run to another, a test may pass one time and fail another. When running one of the standard tests, such an inconsistency most likely indicates a synchronization error in the proxy, since the delays included in the regression tests should provide ample time for activities to take place. Keep in mind that a correct proxy should pass all tests every time. If tests succeed in some runs but fail in others, it is highly likely that the proxy is incorrect, and it is only by chance that the bugs are not always triggered.
- PXYDRIVE uses randomly generated port numbers for the proxy and the servers. There is a small chance that its chosen numbers will conflict with ones already in use. If this happens, try rerunning the program, so that it will select a new set of ports.
- PXYDRIVE is still in early stages of development. It therefore has the potential to hang up when an error occurs, making it impossible to exit with the `quit` command or via Control-C. If this occurs, you can kill the program by (in a separate window), using the `ps` command to determine the process number and the `kill` command to kill the process.