mod_kaPoW: Mitigating Denial-of-Service with Transparent Proof-of-Work

Ed Kaiser & Wu-chang Feng

The Problem

Unwanted traffic like ***Denial-of-Service attacks*** remain a problem for networked systems.

Proof-of-Work is a defense that prioritizes service requests based on the clients' willingness to solve computational challenges.

Existing Proof-of-Work schemes have not made much progress towards deployment because in order to work they require the wide-scale use of **special client software**.

1

The Challenges

Transparency so that clients do not need to download and install special software.

Backwards-compatibility so clients that cannot solve challenges may still participate.

Bind work functions to client, server, and time.

Efficiency to minimize overhead.

Tailor challenges with client-specific difficulty to prioritize clients based on their past behavior.

2

The Solution

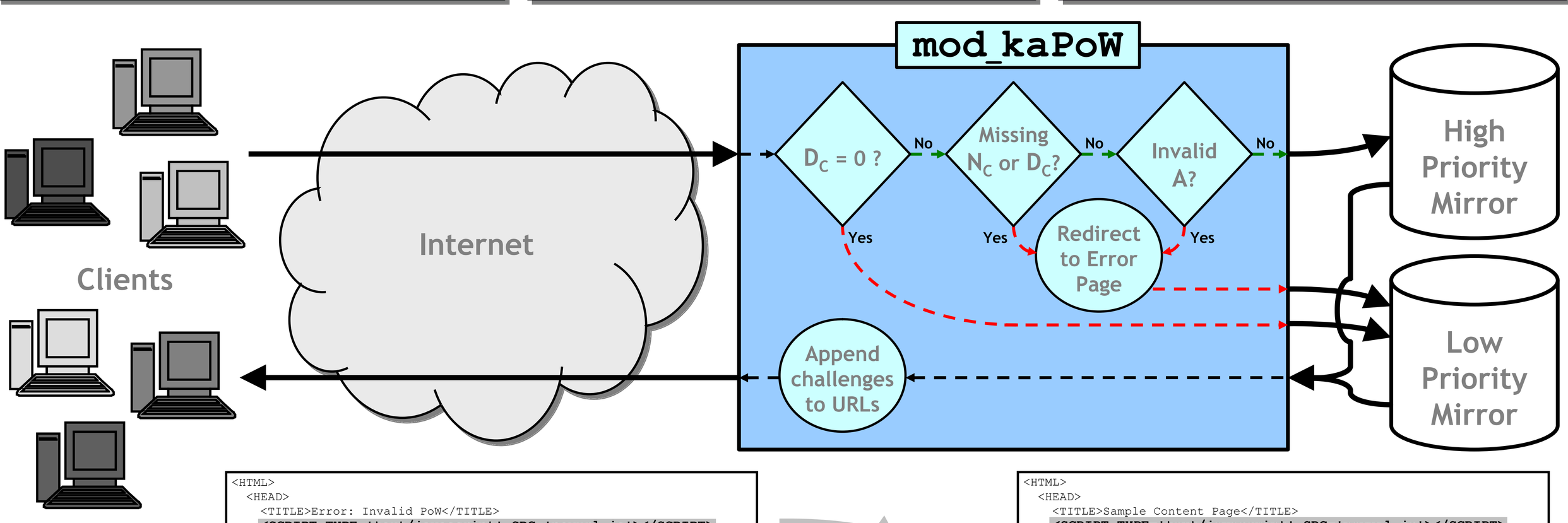
Embed the Proof-of-Work challenges and responses ***within the URLs*** of protected web content.

Clients use ***JavaScript*** to solve the work functions.

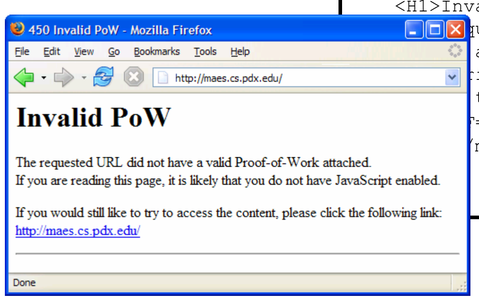
The server uses an ***Apache*** module to prioritize HTTP requests based on the solution in the URL;

valid solution	→	high priority
missing solution	→	low priority error
invalid solution	→	low priority error
zero difficulty solution	→	low priority

3

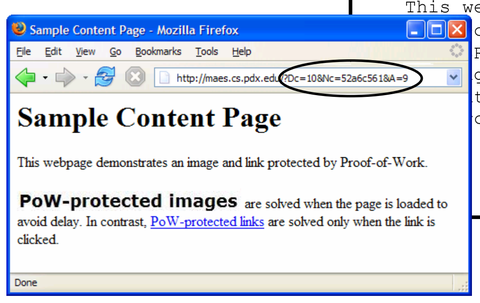



```
<HTML>
  <HEAD>
    <TITLE>Error: Invalid PoW</TITLE>
    <SCRIPT TYPE='text/javascript' SRC='powurl.js'></SCRIPT>
  </HEAD>
  <BODY ONLOAD='Solve(document.links[0]);
                window.location.replace(document.links[0].href)'>
    <H1>Invalid PoW</H1>
    quested URL did not have a valid Proof-of-Work attached.<BR>
     are reading this page, it is likely that you do not have
    ipt enabled.<BR><BR> If you would still like to try to
     the content, please click the following link:
    ='http://maes.cs.pdx.edu/?Dc=0' Nc=52a6c561 Dc=10>
    /maes.cs.pdx.edu/</A><BR><BR><HR>
```



The Error Page

```
<HTML>
  <HEAD>
    <TITLE>Sample Content Page</TITLE>
    <SCRIPT TYPE='text/javascript' SRC='powurl.js'></SCRIPT>
  </HEAD>
  <BODY>
    <H1>Sample Content Page</H1>
    This webpage demonstrates an image and link protected by
    of-Work.<BR><BR>
    RC='test.jpg?Dc=0' Nc=a53b6145 Dc=10> are solved when
    ge is loaded to avoid delay.<BR>
    trast, <A HREF='/?Dc=0' Nc=52a6c561 Dc=10>
    otected links</A> are solved only when the link is clicked.
```



A Modified Content Page

The Work Function

Find an answer **A** that satisfies:

$$H(D_C, N_C, A) \circ 0 \bmod D_C \quad (1)$$

where;

H is a one-way hash function (i.e. SHA1) with uniformly distributed output

D_C is a client-specific server-assigned difficulty

N_C is a client-specific server-generated nonce, generated by:

$$N_C = E_K(IP_C, URL, D_C) \quad (2)$$

using;

E an efficient encryption algorithm (i.e. the XTEA block cipher)

K the secret key held by the server

IP_C the client's network identity

URL the resource descriptor contained in the request

D_C the same client-specific server-assigned difficulty as above

4

Transparency

Client browsers use the `Solve()` script as needed; image URLs are solved as the DOM is loaded but hyperlinks are only solved when clicked. This is driven through scripts; ***user input is not needed***.

The error page's script automatically solves the work function and refreshes using the correct URL; ***the error page is not seen by users***.

Webpages are modified only upon egress from the module; ***content servers operate as normal***.

5

Backwards Compatibility

Modified URLs default to difficulty zero ($D_C = 0$) so that legacy clients without *JavaScript* enabled can access the content on the low priority mirror; ***all clients have a method to access content***.

The module operates independent to content production and does not require any changes to the format or content of webpages, whether they are static or dynamically generated; ***the module flexibly modifies outgoing webpages***.

6

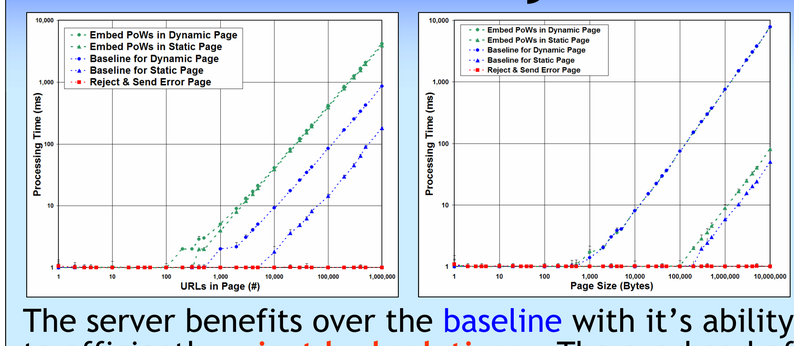
Challenge Difficulty

The client-specific difficulty D_C is assigned by the server based upon the maximum of either the client's contribution to the current aggregate load or the client's slowly decaying load history.

The history is stored efficiently using a counting Bloom Filter indexed by the client's identity IP_C. Each entry measures a client's cumulative load from ***successful*** requests (i.e. those that had valid solutions).

7

Efficiency



The server benefits over the baseline with it's ability to efficiently **reject bad solutions**. The overhead of **appending challenges** to URLs is only significant for large files containing hundreds of URLs.

8

PORTLAND STATE UNIVERSITY

Project Webpage: http://maes.cs.pdx.edu

Funded by: intel NATIONAL SCIENCE FOUNDATION