SMART PROXY

CS370: Project Report

Marc Aldrich
marcaldrich@gmail.com

Project Introduction

Users are often frustrated by delays and unexpected blocking of content when browsing the Internet through commercial Internet Service Providers (ISP). ISPs maintain large portions of the Internet and provide abstraction of network details for the user. Generally, abstraction of technical details is desirable, but it does leave many users unaware of the policies of ISPs. Businesses that operate partitions of this interconnected system often set and enforce policies for the data traversing their subset of the Internet. These policies, which are expressed by limiting bandwidth for a data stream or blocking the data stream entirely, are heavily influenced by the organizations' political or financial interests. Often such policies do not align with the best interest of the end user. A highly visible example of business interest causing an unfavorable user experience is the use of geolocation to control content availability. This project presents Smart Proxy, a non-invasive solution to circumvent these unfavorable routing policies and achieve an improved web browsing experience.

Design Constraints

Smart Proxy was designed under the following constraints:

- The end user must not be hindered from a typical web browsing experience.
- Users should need minimal to no experience, training or configuration to use the service.
- Traffic should be rerouted as little as possible to minimize network delay and buffering.
- To minimize bandwidth costs, targeted domains must be the only traffic to be routed over Smart Proxy service.
- Proxy nodes should be created and destroyed without affecting network or service performance.

Technology Background

This project is concerned with several protocols that enable the web browsing experience that users have come to enjoy and expect: Internet Protocol (IP), Domain Name System (DNS), Hypertext Transfer Protocol (HTTP) and Transport Layer Security protocol (TLS). The base of all internet technology is IP, which is used to interconnect the systems of networks that make up the Internet. IP's main feature is the standardization of naming for each device connected to a network. These names, which are unique for each device, are called *IP addresses*. IP also abstracts data transfer such that the user need not know, or care about, the path that their data will take to arrive at a desired destination. Abstraction of the routing was a design choice intended to ensure the survivability of network communications in the real world, where routes can be unpredictable due to hardware failures or software crashes.

As a result of the flexibility provided by abstract routing, IP facilitates better routing decisions by sharing metadata with any routing equipment that touches the data. Among this metadata is information about the originator of the traffic, the next node along the data route, and the final destination of the traffic. Each of these network points are named by an IP address. The metadata, which accompanies an IP transmission, is also what enables traffic shaping, such as geolocation by ISPs and other network providers. Geolocation allows an IP address to be approximately located on a standard map[1], and it is implemented by combining an IP address with the publically available registration data associated with that address.

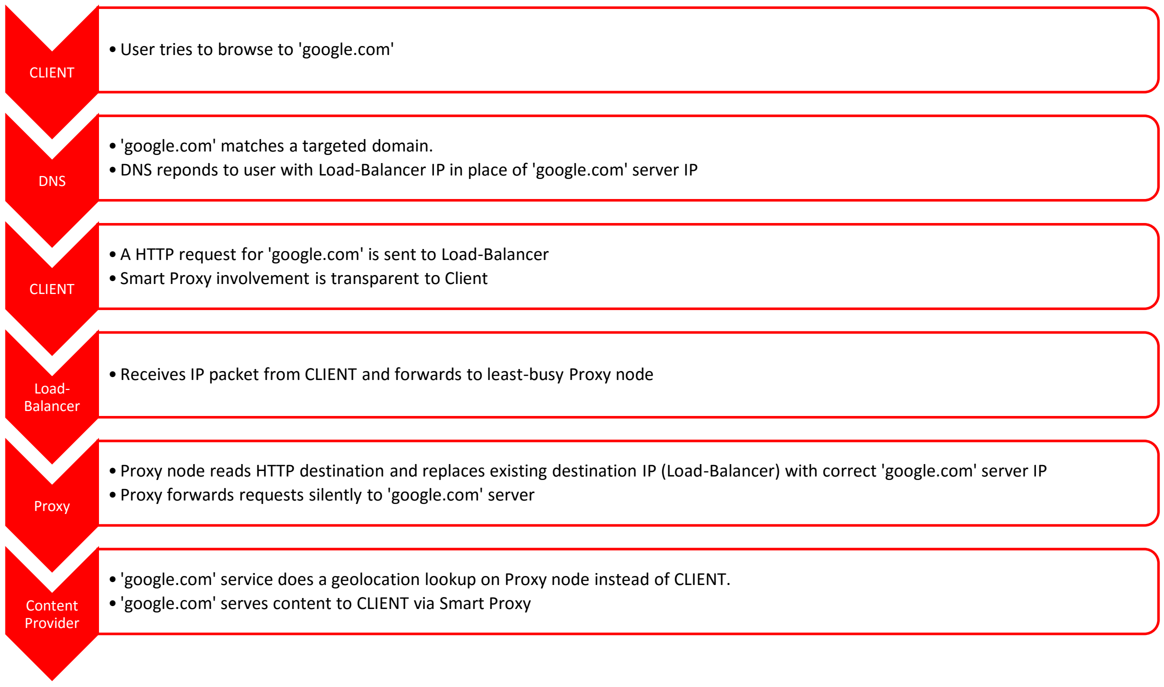
[1] Maxmind. (2015, October 29). How to Protect Your Streaming Content from VPN & Proxy Traffic. Retrieved April 28, 2016, from https://blog.maxmind.com/2015/10/29/how-to-protect-your-streaming-content-from-vpn-proxy-traffic/

IP addresses are represented by strings of numbers or hexadecimals, depending on display formatting. In either form IP addresses are difficult to either a) remember or b) read and easily decipher the content of the system. Making the network more usable by humans is a protocol specified by the Domain Name System[2] and implemented by a Domain Name Service (DNS), which provides a translation between the machine friendly IP address and common human readable identifiers. While DNS allows for users to easily remember server names, it also allows for filtering technologies to easily make coarse determinations about the content of a server. DNS provides a hierarchical system of translation between domains and IP addresses. At a high level, DNS looks like the following:

1. A device asks a DNS server for a translation of a domain and it will return the IP address.
2. If the first contacted DNS server does not know the domain to IP translation, then it will forward the request to a higher tier DNS server.
3. This is repeated until an answer is found or a top tier server returns “Non-existent domain”.

The protocol that utilizes DNS within the context of Smart Proxy is the Hypertext Transfer Protocol[3] (HTTP). HTTP provides a standard messaging and data encapsulation format for a client and server to communicate. The general operation of HTTP is not important to the Smart Proxy project other than to note:

1. HTTP is the layer where a client specifies a data request’s destination domain, and
2. HTTP’s lack of security led to the design and implementation of Hypertext Transfer Protocol Secure (HTTPS) .


CLIENT
• User tries to browse to 'google.com'
DNS
• 'google.com' matches a targeted domain.
• DNS reponds to user with Load-Balancer IP in place of 'google.com' server IP
CLIENT
• A HTTP request for 'google.com' is sent to Load-Balancer
• Smart Proxy involvement is transparent to Client
Load-Balancer
• Receives IP packet from CLIENT and forwards to least-busy Proxy node
Proxy
• Proxy node reads HTTP destination and replaces existing destination IP (Load-Balancer) with correct 'google.com' server IP
• Proxy forwards requests silently to 'google.com' server
Content Provider
• 'google.com' service does a geolocation lookup on Proxy node instead of CLIENT.
• 'google.com' serves content to CLIENT via Smart Proxy

[2] IETF. (1987, November). DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION. Retrieved April 28, 2016, from https://www.ietf.org/rfc/rfc1035.txt
[3] IETF. (2014, June). RFC 7230 - Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. Retrieved April 28, 2016, from https://tools.ietf.org/html/rfc7230

Implementation

Smart Proxy is comprised of modified FOSS projects and needed system-level configurations that enable desired traffic, and only the desired traffic, to be routed outside an affected network exit node to a load-balanced geo-located proxy. If Smart Proxy is deployed as a standalone product, then the user can enable it by just updating the DNS server address on the devices desired. When deployed as an augmentation to an existing network infrastructure, the end user is not required to make any changes at all. The system can be logically broken down into three pieces: DNS server, Load-Balancer and Proxy.

The DNS server is implemented as a set of modified Bind scripts that ensure the local DNS server claims to know the IP address of targeted domains. Bind is a very common DNS server implementation that is highly configurable and free and open source. To ensure that minimal set of domains is redirected, a network administrator must observe the flow of DNS and HTTP traffic to a targeted domain and note which domains are serving content and web scripts for a given web site. The DNS response for the target domains is the address of the geographically specific load-balancer that keeps a list of proxy servers co-located within the country of the VPN exit node.

The Load-Balancer is implemented using HAProxy[4]. HAProxy provides TCP-layer routing of any traffic directed to it. When the user's request for a web page is received, HAProxy considers a list of known exit nodes and forwards traffic to one of them, which is chosen by an algorithm that favors the server with the fewest active connections. In a deployment environment, the load-balancer is co-located with the DNS server to minimize redirection delay and decrease bandwidth cost.

The proxy node is located in a data center near the geographic location of the content provider's region of restriction. The exit node utilizes a modified FOSS project called Squid[5]. Required code modification was small, but it effectively forces the exit node to do a DNS resolution of the domain in the HTTP or HTTPS packet. The new logic replaces the IP data packet destination with the IP address of the actual host received from an unmodified DNS resolution. An unmodified DNS resolution is required because the initial Smart Proxy based DNS returned the address of the Load-Balancer to transparently make the client route data across the Smart Proxy system. In the graphic above the first three steps depict this process. Once the packet is received by the Proxy server it must do an IP lookup to find the actual IP address of the server for the domain requested, so that it can forward the data stream to the content provider, which is depicted in the last two steps of the graphic.

Finally, the flow of network traffic proceeds as usual, with the small exception that traffic to the targeted domain is redirected through an unblocked proxy server near the geo-location of the current VPN exit node. The use of the load-balancer allows proxy servers to be swapped into and out of the system with minimal effort. Updating the load-balancer's list of known proxy servers and reloading its configuration is sufficient to replace proxy servers that get blocked or otherwise need to be brought offline. Load-balancing also allows the system to balance the cost of bandwidth through different data hosting providers.

Implementation Challenges

Many parts of the Smart Proxy implementation are straightforward, but some were not. The first and most significant was that the unmodified Squid proxy would not overwrite the destination address of a

[4] HAProxy. (2016, April 13). HAProxy. Retrieved April 28, 2016, from http://www.haproxy.org
[5] Dawson, A., & Chadd, A. (2013, October 05). Squid-cache.org. Retrieved April 28, 2016, from http://www.squid-cache.org/

received packet. Several configuration options were used to try to force this behavior, but none of the default configuration options included covered this use case. For example, 'ssl-bump' was tried, which works by decrypting the received traffic and resetting the destination IP. This was not used as it would've required extra configuration for the user as well as not having any effect on non-encrypted traffic. So the code was modified as described above. Before, Squid only did a DNS lookup and forward if it received a request not contained in its cache. Now Squid does the DNS lookup and forwarding on any traffic the proxy server receives. Since DNS resolutions are cached by the operating system, doing a resolution for each new stream does not affect performance.

Another challenge was that HTTPS traffic would fail to forward through the proxy since Squid was not inspecting the TLS:SNI field for HTTPS connections. When TLS encrypts the HTTP message, the destination domain is obscured and only the IP packet's destination can be used for forwarding. Obscuring the destination domain increases security, unfortunately it also breaks the expected behavior of most web traffic. The forwarding broke because the destination domain contained in the HTTP message is encrypted so Squid forwarding logic could not perform a DNS resolution to properly forward to the correct content server. After noticing this problem, TLS was extended by a specification extension[6]. RFC-3546 defined the creation of an 'SNI' data field to be included in the TLS handshake to solve this issue. The desired destination domain is listed in the client metadata when a secure connection is attempted, so that forwarding nodes between the client and server can correctly route the traffic. Support for SNI based forwarding was added to Squid-Proxy through a community-supplied patch, so now the forwarding of encrypted traffic to target domains is supported. This change enables Smart Proxy to handle encrypted traffic without the need to decrypt and inspect client traffic.

Conclusions and Future Work

Smart Proxy mitigates the issue of VPN exit nodes being blocked while adhering to design constraints. Smart Proxy nodes are already deployed, and they are able to circumvent data shaping based on the use of VPN and Proxy technologies. Ultimately, maintaining this system is a function of detecting and replacing exit nodes as bandwidth limits are reached or the Proxy IP address is blocked.

This project could be further developed by implementing a web interface for the load-balancer to enable the support teams to create, destroy, and otherwise manage the proxy nodes. Currently, servers must be brought up manually and a bash script must be activated. From the same web interface, the targeted domains could be updated within the DNS server. Health-monitoring is a limitation of the current system that needs further research and development. HAProxy, natively supports real-time health monitoring, but it does not retain a history of the health status making troubleshooting difficult to do. Extending HAProxy to support more robust health and statistic tracking would greatly enhance the system. Possibly this will lead to the ability of the system to automate the replacement of exit nodes without intervention of a support team.

Smart Proxy could be used to combat censorship in countries like Iran and China where blacklisting of servers and interception and modification of DNS traffic is aggressive. Through a slight modification of the code, the server could ignore TCP reset packets and support DNSCrypt, which would stop DNS packet attacks. By inverting the topology of Smart Proxy, 'exit-nodes' would become 'input-nodes', users would access the unfiltered Internet by switching Internet gateways often and nearly seamlessly. Smart Proxy provides a good foundation for that scenario.

[6] Blake-Wilson et al, S. (2003, June). RFC 3546 - Transport Layer Security (TLS) Extensions. Retrieved April 28, 2016, from https://tools.ietf.org/html/rfc3546