SNIP: TLS based IPv6 transition proxy

Yorin Anne de Jong*, Otto Jonassen Wittner†
UNINETT
{yorn*, wittner†}@uninett.no
+47 95361017*, +47 99550566†,

***Index Terms*—IPv6 transition, Proxy, HTTPS, DNS, IPv6 only servers**

Extended abstract

Four years has passed since the last blocks of IPv4 address where handed out by the Internet Assigned Numbers Authority (IANA). Still only less than 12% of web sites in the world support IPv6 [1] and the amount of IPv6 enable users is below 10%. Hence a full transition from "legacy" IPv4 to modern IPv6 addressing cannot be claimed to yet have taken place.

Since the introduction of IPv6 in 1995 [2] IPv4-to-IPv6 transition technologies have been a research and development area. Considering the still low penetration of native IPv6 servers and clients, transition technologies have indeed still relevance.

In a client environment, there are multiple transition solutions to address the IPv4 address shortage. NAT44 (or just NAT), i.e. network address translation from one IPv4 address to another IPv4 address, is by far the most typical technology to "save" IPv4 address space. Usage scale varies from small (a few nodes) home networks to WANs applying carrier grade NAT. However alternatives to NAT44 has been around for a while, NAT64 [3] and CLAT (based on 464XLAT [4]) being two well known options.

On the server side, however, there has not been as much development. The use of VHosts (running multiple similar services on a single server, e.g. [5]) has successfully kept the need for IPv4 server addresses low, similar to what NAT44 has done for clients. In modern models, where services run in containers, there is usually a proxy server which exposes all services on a single IP address, similar to the way VHosts work.

In an future "IPv6-only" world, VHosts and container proxies may become obsolete as every service can simply claim its own IPv6 address. One immediate advantages of this is the improved ability to build service specific access control lists (ACLs) based on addresses in lower layers of the network stack, for example in a router. It also simplifies the stack, because application level routing is no longer required, sparing the administrator from configuring a VHost and proxy servers as well as maintaining the related required software.

SNIP

This extended abstract presents an alternative server side IPv4 to IPv6 transition technology named *Server Name Indication Proxy* (SNIP). The core concept is still based on placing a proxy server between the application server and the clients. But to keep the complexity down and avoid the need for a traditional address translation table, SNIP utilizes the fact that a significant number web based services are available over HTTPS, i.e. encrypted HTTP connections. According to [6] already in 2011 more than 30% of the 1 million most visited web sites applied HTTPS. Embedded in the underlying Transport Layer Security connections (which provide the S in modern HTTPS connections) is a Server Name Indication (SNI) value, i.e. the domain name of the destination application server. Hence the mapping required to perform address translation can be configured in the Domain Name System (DNS).

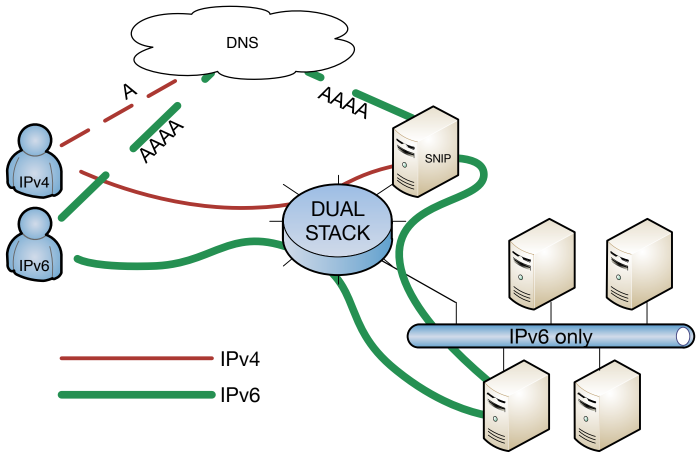

Figure 1. The SNIP architecture including a SNIP session connection setup.

Figure 1 illustrate the SNIP architecture and two session examples, one Ipv6 only and one with IPv4-to-IPv6 translation. Traffic from IPv6 clients are routed (via dual stack routing components) directly to the relevant IPv6 only server as DNS reports the native IPv6 address of the server (AAAA DNS entry). However when IPv4 clients enquired DNS for server addresses (A entries) the IPv4 address of the SNIP proxy is returned. Traffic is hence routed to the proxy which is ready listening to port 443 (HTTPS). The proxy enquires the DNS service for IPv6 address information (AAAA entries) based on the SNI value extracted from the TLS header. The returned IPv6 address as well as the incoming IPv4 based client connection is applied by the SNIP proxy to establish two TCP connections, one towards client and one towards server. All payload of TCP data packets are forwarded (in both directions) by the SNIP proxy until one of the TCP connections are teared down.

Core assumptions made by the SNIP proxy is that all traffic

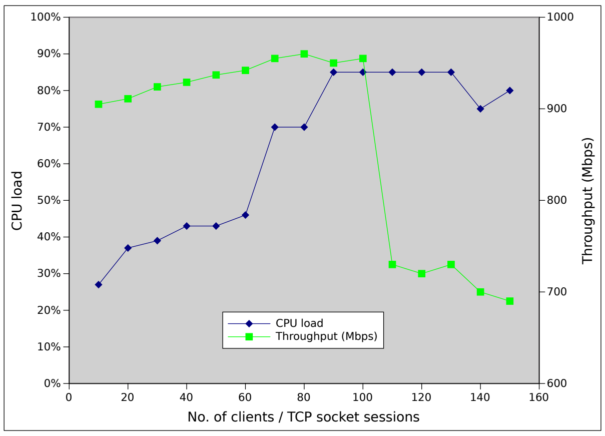

Figure 2. Performance on a single SNIP instance running on single CPU virtualized server.

applies HTTPS with TLS and that clients (browsers) support SNI values. A browser not supporting SNI but still with a relevant market share is Internet Explorer 6 (IE6). However IE6 is typically only applied in intranets (due to security challenges), hence browsers on the public internet may be assumed to support SNI. Today any well configured HTTP-server will support HTTPS. Recent initiatives to offer free certificates (required by TLS), e.g. [7] will enable any server to support trusted TLS session at no or limited cost.

The remaining fraction of none-HTTPS traffic may be handle by either dual stacks in servers or by an TCP proxy, e.g. through HAProxy [8].

Some none-HTTPS services will only exist as single instances in an organisation, for example mail (IMAP, SMTP), chat (XMPP) and VPN (OpenVPN). These can still share the same IPv4 address as they run on different ports. A port based mapping to different IPv6 addresses can be applied.

A security concern with SNIP may be that the proxy can be exploited for source address anonymization. To avoid this a SNIP proxy may be delegated a /32 IPv6 address range which enables embedding of original IPv4 source address in the IPv6 source address applied to contact the HTTPS server.

A proof of concept of SNIP is implemented in the NodeJS language. This language allows fairly complex operations to be done with small amounts of code.

Evaluation

Figure 2 shows the results of some preliminary performance measurements of SNIP. A single Ubuntu 14.04 configured Intel Core i7 based laptop with gigabit network interface was applied to generate client request with the *Curl* command line browser. A SNIP proxy running in an virtual machine with a single 2.0 Ghz CPU and 768MB of memory was prepared. A 16 core 2.4Ghz server with 32GB memory running *Apache2* was applied as HTTPS server. The bottleneck link in the setup supported 1Gbps. A number of concurrently running Curl clients where executed to download a 1GB file from the Apache server. The maximum number of client where chosen to match the number of parallel sessions handled (before they are queued) by Apache with a default configuration, i.e. 150.

As is readily seen in Figure 2, a single instance of SNIP coupes with around 100 clients. As CPU load reaches 85% the VM running SNIP struggles to offer enough resources and througput drops. At 140 clients the VM also starts swapping memory, hence throughput drops further.

Related Work

HAProxy [8] is an open source high availability HTTP, HTTPS and TCP proxy. For HTTP and HTTPS, it uses the incoming socket (IP address and port) and host-name provided in the HTTP header to provide VHost-capabilities. In the case of HTTPS, this means that it needs to decrypt the traffic and therefore needs a private key. The TCP proxy uses only the incoming socket. There is currently no support for utilizing SNI values.

SNI Proxy [9] is an open source TCP proxy that uses the SNI value to provide VHost-capabilities. It uses a configuration file which maps the SNI value to an IP address. This is very similar to the behaviour of SNIP, except that SNIP uses DNS instead of a config file.

NAT64 [3] is a transition technology in order to make external IPv4-only servers available to local IPv6 clients. It is typically employed by a network administrator that has IPv6-only clients. The goal is the counterpart of SNIP, where SNIP makes local IPv6 servers available to external IPv4-only clients. In a setup with IPv6-only servers, it is possible to use both SNIP and NAT64 in tandem, in order to make the servers available for IPv4-only hosts, and to allow the servers to use external IPv4-only APIs.

Conclusion and Future work

SNIP provides a proof of concept for a NodeJS implemented IPv4-to-IPv6 transition mechanism enabling IPv6 only servers to be reached by IPv4 clients. The current implementation has stability issues when it needs to handle a lot of connections at the same time. A framework is under development to control maximum load on a single instance of SNIP, i.e. a SNIP proxy load balancer, and hence improve stability significantly. Despite its current limitations SNIP seems to coupe well with respect to a typical default web server configuration, and may be viewed as an valuable step towards the IPv6-only server park.

References

[1] E. Vyncke, “Ipv6 deployment aggregated status.” https://www.vyncke.org/ipv6status/. Visited 2015-11-11.

[2] S. Deering and R. Hinden, “Internet protocol, version 6 (ipv6) specification.” https://tools.ietf.org/html/rfc1883, Dec 1995.

[3] M. Bagnulo, P. Matthews, and I. van Beijnum, “Stateful nat64: Network address and protocol translation from ipv6 clients to ipv4 servers.” https://tools.ietf.org/html/rfc6146, April 2011.

[4] M. Mawatari, M. Kawashima, and C. Byrne, “464xlat: Combination of stateful and stateless translation.” http://tools.ietf.org/html/rfc6877, April 2013.

[5] T. A. S. Foundation, “Apache virtual host documentation.” https://httpd.apache.org/docs/2.2/vhosts/. Visited 2015-11-11.

[6] I. Ristic and M. Small, “A study of what really breaks ssl.” Presentation at HITB Sec Conf 2011, Amsterdam, May 2011.

[7] “Startssl.” https://www.startssl.com/. Visited 2015-11-30.

[8] W. Tarreau, “Haproxy - the reliable, high performance tcp/http load balancer.” http://www.haproxy.org/, 2000. Visited 2015-11-30.

[9] D. Lundquist, “Sni proxy.” https://github.com/dlundquist/sniproxy. Visited 2015-11-30.

VITAE

Yorin Anne de Jong

has a Msc in telematic from the Norwegian University of Science and Technology (NTNU). He has a full time position at UNINETT's systems department, and works with software development in many different activities.

Otto J Wittner

has a PhD in Network Management from Norwegian University of Science and Technology (NTNU). He is currently engaged full time in UNINETT's innovation program where he explores upcoming internet networking and multimedia technologies. He also holds a part time adjunct professorship at NTNU.