Impact of the lack of ports in a DS-Lite architecture

Isabelle Kraemer   Frédéric Perrin
Télécom Bretagne

March 18, 2011

Abstract

As the supply of IPv4 addresses for end users is shrinking, while IPv6 is not as wildly available as one would like, ISPs need to deploy new architectures that enable them to reduce their consumption of public IPv4 addresses. One such architecture is DS-Lite. However, with this new network design come new risks for the end-user online experience. In particular, the limited supply of ports in the public interface of the CGN means that under heavy use, outgoing connections from end-users may fail.

Keywords DS-Lite, IPv6 transition, CGN, NAT, port consumption, port starvation, TCP congestion.

Contents

1 The DS-Lite architecture

1.1 Terminology

DS-Lite For *Dual-Stack Lite.* ISP network architecture for handling the transition from IPv4 to IPv6[4], in which IPv4 traffic from the end-user is encapsulated from the B4 to the AFTR.

B4 For *Basic Bridging BroadBand.* Network element providing access to the IPv4 Internet, by establishing an IPv4-in-IPv6 tunnel to an AFTR belonging to the ISP.

AFTR For *Address Family Transition Router.* CGN installed in the ISP network. It works like a classical NAT except that it keeps an additional element in the NAT context: the IPv6 address of the tunnel from which the connection originates. This address identifies the B4 (and hence the customer) initiating the connection.

CGN For *Carrier-Grade NAT*. NAT managed by the ISP and potentially used by a great number of final users.

CPE For *Customer-Premises Equipment*. A network device, usually owned by the ISP and leased to the customer, located in the end-user's premises. It combines the functions of B4, IPv6 default router, DNS cache, sometimes WiFi access point, etc.

1.2 DS-Lite mode of operation

The figure 1 shows the principle of the DS-Lite architecture. The full architecture is being standardized, currently in the Internet Draft stage[4].

The CPE, which is located at the customer's premises, is configured with the IPv6 address of the AFTR, a subnet (typically a /64, or maybe a /56) for the customer's network, plus its own routing information in the ISP's network. With the usual auto-configuration mechanisms, it distributes both a global IPv6 addresses and private IPv4 addresses to the customer's devices. It will also announce itself as the default gateway for both protocols, and as a caching DNS server.

When a customer device needs to access the IPv6 Internet, packets are routed normally in the ISP's network until it reaches the public IPv6 Internet. Both incoming and outgoing connections are possible.

When a customer's device needs to access the legacy IPv4 Internet, things are more complicated. We will take as an example a TCP connection initiated by a customer device to a remote IPv4 server. The customer's device is not aware of the presence of an AFTR between it and the public IPv4 Internet (or, for that matter, that it is using a private IPv4 address). It sends a TCP SYN segment, using its own IPv4 address @S as the source and the remote server's IPv4 address @D as the destination. The destination port PD is imposed by the application protocol (port 80 for the WWW, port 22 for SSH, etc.), while the source port PS is randomly chosen by the customer's device. This segment is sent to the CPE, which is the default gateway for the customer's LAN. This segment is noted ① in the figure 1.

The CPE, acting as a B4, sends this segment to the AFTR in an IPv4-in-IPv6 tunnel. It puts this TCP/IPv4 segment inside an IPv6 packet, using a *Next Header* of IPIP, its own public IPv6 address @B4 as the source address and the AFTR IPv6 address @AFTR as the destination. This IPv6 packet ② is then routed in the ISP network up to the AFTR.

The AFTR receives this IPv4-in-IPv6 packet. It notes the source IPv6 address @B4 and opens the TCP/IPv4 segment inside. As this segment doesn't belong to a known TCP connection, the AFTR creates a new context, defined with ⟨@B4, @S, PS | @D, PD, @AFTR$_4$, PAFTR$_4$⟩. The port PAFTR$_4$ is chosen randomly by the AFTR amongst its free ports. The AFTR replaces the source address with its own (@AFTR$_4$) and the source port with PAFTR$_4$. This partially rewritten TCP/IPv4 segment ④ is finally sent to the public IPv4 Internet, and reaches the remote server.

Assuming the remote server accepts this new connection, it replies with a SYN+ACK segment. It uses ⟨@D, PD⟩ as the source, and ⟨@AFTR$_4$, PAFTR$_4$⟩ as the destination; it is not aware that the customer is behind an AFTR, and considers that the connection has been initiated by the AFTR.

When the AFTR receives the answer from the remote server, it matches the segment against the known contexts. The AFTR recognizes the tuple ⟨@D, PD, @AFTR$_4$, PAFTR$_4$⟩. It replaces the destination ⟨@AFTR$_4$, PAFTR$_4$⟩ with the customer information ⟨@S, PS⟩, and encapsulates this partially rewritten TCP/IPv4 segment inside an IPv6 address, using as the source its IPv6 address @AFTR and as the destination the B4 address @B4.

Further segments belonging to the same TCP connection follow the same path. The only difference between the initial SYN segment and the following ones sent by the end-user is, at the AFTR, that the tuple ⟨@B4, @S, PS, @D, PD⟩ is already known, and the same port PAFTR$_4$ is used when rewriting the source address of the TCP segment sent over the public Internet.

When one endpoint wishes to close the connection, it sends a FIN segment (or an RST in some TCP implementations). The segment itself is transported

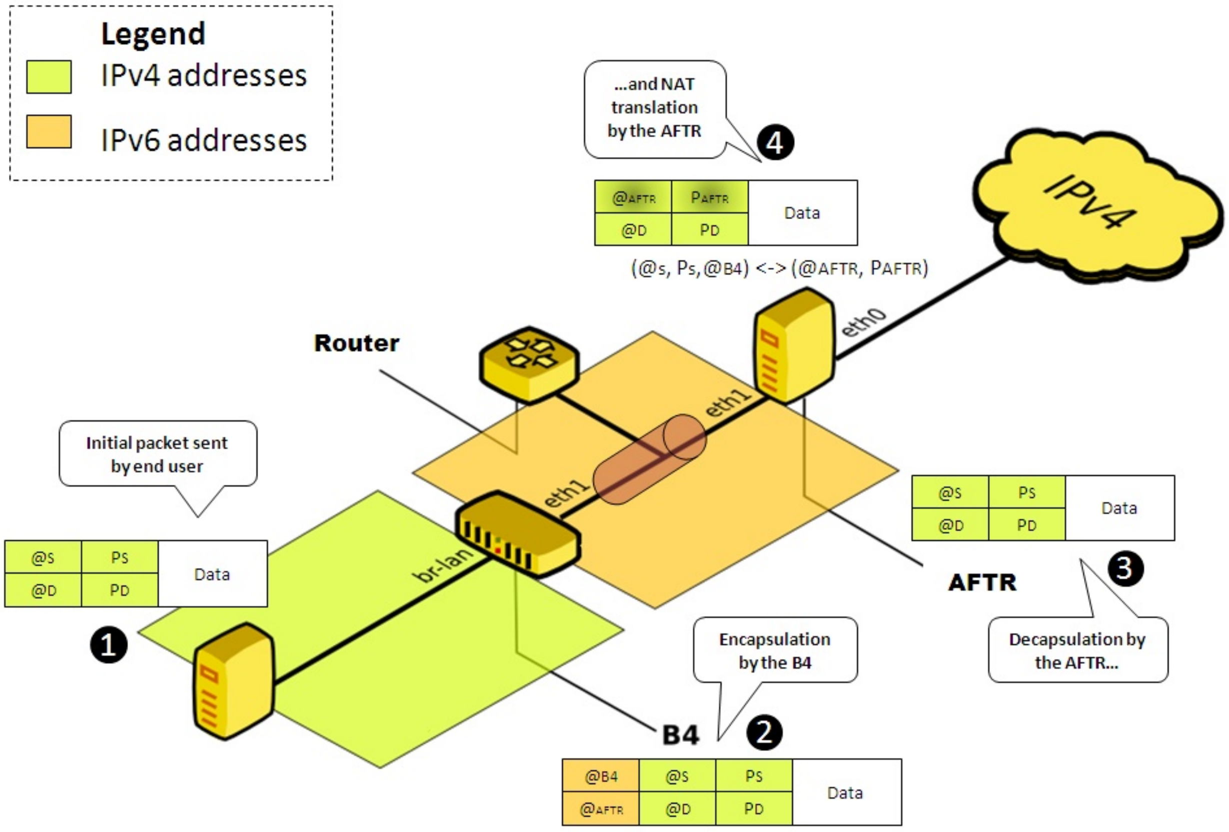
Legend
IPv4 addresses
IPv6 addresses
...and NAT translation by the AFTR
4
@AFTR PAFTR @D PD Data
(@S, PS, @B4) <-> (@AFTR, PAFTR)
IPv4
eth0
Router
eth1
eth1
Initial packet sent by end user
@S PS @D PD Data
br-lan
1
@S PS @D PD Data
AFTR
3
Decapsulation by the AFTR...
Encapsulation by the B4
B4
2
@B4 @S PS @AFTR @D PD Data

Figure 1: DS-Lite architecture: emission of a TCP segment from an end-user to the IPv4 Internet

in the same way as the data segments. The context of this connection is marked for deletion by the AFTR. After some time, the context is actually deleted and the port $\textsc{Paftr}_4$ marked as free and reusable.

It should be noted that neither the customer device nor the remote server are aware of the address rewriting made by the AFTR, nor of the fact that the packet travelled inside a tunnel between the B4 and the AFTR. As a result, this architecture is mostly transparent to both endpoints, and it is not necessary to upgrade the TCP/IPv4 stack of either endpoint.

UDP sessions initiated by an customer work the same way. A difference is that UDP sessions don't have state; it particular, they are not explicitely closed. This means that the AFTR need to track UDP session based only on $\langle$@B4, @S, PS, @D, PD$\rangle$ tuples, whithout being able to rely on flags inside the datagrams. UDP contexts may be marked for deletion when no traffic is seen during a configurated time, but there is no way to differentiate between an idle UDP session and a session where both endpoints are done.

It is not possible for remote machines on the public IPv4 Internet to initiate connections to customer systems.

1.3 Known issues and advantages

From the customer's point of view, the DS-Lite architecture has the same issues as usual NAT, plus some more:

- Only outgoing connections are possible. As the AFTR is not under the customer's control, she can't setup port redirection. Hosting servers at home is made much more difficult.

- Special handling is needed for each layer 4 protocol such as TCP, UDP and ICMP. Protocols like FTP that include layer 3 / 4 information in the application layer need further special care; VoIP or P2P applications need to include NAT traversal techniques. Many protocols which assume end-to-end communication are likely to remain broken.

- Several customers are seen, from the point of view of remote IPv4 servers, as using the same IPv4 address. This is an issue in on-line gaming, for instance, where one disruptive player has the potential of banning several households at once.

Of course, by reducing the incentive of a quick IPv6 deployment for all customers, it may be argued that the mere possibility of accessing the IPv4 Internet is in itself a drawback.

In order to alleviate the first point above, the ISP may setup a protocol like PCP. In PCP, the customer may require that all connections from the IPv4 Internet to a specific TCP or UDP port on the AFTR be forwarded to one of her devices. However, the same public interface is shared by several customers, and as a result ports over the public Internet are a scarce resource. She must be prepared to be given a port outside the well-known port range, and possibly to have to change ports regularly.

From the point of view of the ISP, however, this scheme has several advantages:

- All traffic in the ISP network is IPv6. There is no need to maintain a dual-stack network.

- One single public IPv4 address is usable by several customers, and there is no need to use IPv4 at all for the ISP infrastructure. This saves a lot of relatively rare public IPv4 addresses.

- As all IPv4 communications go through one device, it is easy for the ISP to monitor, filter and prioritize traffic. It is at least debatable whether this is in the customer's best interests.

Nevertheless, even from the point of view of the ISP, there are several drawbacks to this architecture:

- All the IPv4 traffic from several customers is concentrated in one single place. As a result, the AFTR becomes a single point of failure, and also risks being overloaded with traffic.

- In order to comply with various log retention laws, it is likely necessary to keep track of each port used by each customers. However, such a pervasive logging may be seen as an invasion of customers' privacy.

- The IPv4-in-IPv6 tunnel introduces overhead. In particular, as most network equipment has an MTU of 1500 bytes, the effective MTU seen by end-users is 1460 bytes only. This is slightly less efficient for customers, of course; but it may also introduce subtle, hard to debug problems. More generally, if there is an issue in the IPv4-in-IPv6 tunnel, ICMP messages can't be sent to the customer's machine, leading to silent failures.

In spite of all these weaknesses, DS-Lite is still one of the least bad options for dealing with the shortage of IPv4 addresses. Other options include NAT444[9] (where customers are behind two levels of NAT, once at the CPE level, once at the ISP level), DNS64 with NAT64[1, 2] (where the ISP's DNS servers return AAAA records for IPv4-only servers, embedding the IPv4 address in the IPv6 address given to the client). These solutions make it even harder for applications to traverse the NAT.

1.4 Test-bed

We use a B4 element Netgear, installed in the students residence (WNDR3700 model). The AFTR is a Fedora GNU/Linux 13 (2.6.34 kernel) machine, installed in the RSM laboratory, in the College. The AFTR dæmon used was provided by the ISC and written by Francis Dupont[3]. The network between the B4 and the AFTR is successively under the authority of the "Réseau des Élèves" (Students Network), the IT department of Télécom Bretagne and the RSM department of the School.

Only a few machines are behind the B4. The one used for the tests described below is a Sun UltraSparc IIi running Debian GNU/Linux Lenny (2.6.26 kernel). In order to experiment a situation were customers ports range is undersized, we artificially reduce the range of ports the AFTR may use to initiate connections to the IPv4 Internet.

2 Simulating the user experience

2.1 Simulating a web browsing session

From previous work[5], we know that the most port-consuming applications are web browsing and peer-to-peer downloading, before instant messaging and gaming. We simulated end-users browsing the web, and measured the response time as a function of the severity of port starvation.

2.1.1 Principle

In order to simulate an end-user browsing the web, we followed the following procedure:

1. choose an arbitrary web page as a starting point, and initialize an empty pool of addresses;
2. download the page, and all `img`, `script` and `link` elements;
3. add the target of all links to the pool of addresses;
4. sleep for some time;
5. select at random a new address from the pool;
6. go back to point 2.

We created in this manner several surf sessions, with the starting point set to `planet.debian.org`, `twitter.com` and `del.icio.us`. These starting points were chosen because these pages have a lot of external links. In order to get repeatable results, we launched a one-hour long browsing session for each starting point, recording all the pages that were seen. We then replayed the session, in different environment, with point 5 above replaced with "select the next page from the recorded session".

We create 15 threads to download all the elements for a page. If several elements have the same address, only one request is made. No HTTP pipe-lining is used. The same page is never visited twice. A simple caching mecanism is implemented, in order to avoid re-downloading an element if it is amonsgt the last

400 elements seen. This figure of 400 elements comes from Firefox' default cache size of 40 MB, and an averable size for cached elements of 100 kB, notice on one of the author's system.

The pause between two pages follows an exponential law, with a mean time of 15 seconds.

As our concern is the response time of remote servers, for each element we measure the time it takes to establish a TCP connection to port 80. This enable us to alleviate the influence of the limited bandwidth in our test lab, and to measure only the impact of the limitation of TCP ports.

The scripts used in the experiments, along with the resulting data, are available on one of the author's website[6].

2.1.2 Preliminary results

We reported in table 1 some characteristics of the one-hour long browsing sessions. These sessions were done in an IPv4-only network that was not in a TCP starvation situation.

The small number of external links in the Delicious case was rather surprising; this may be explained by the large number of internal links, which flooded the address pool with `delicious.com` addresses.

2.2 Qualifying the user experience

Once a surfing session has been recorded, it may be replayed by a *client*. A *client* is a Perl program that fetches each element seen in the session, and respects the pauses between two pages. We measure the time it takes to start retrieving each element. As we are not interested in testing the bandwidth of our test lab or the responsiveness of our DNS server, all domain names are resolved in advance to put them in the cache, and we stop the timing when receiving the first byte from the page.

Our test lab has enough hardware to simulate 3 simultaneous clients. The HTTP library used (Perl's LWP) has a timeout of 180 seconds before giving up on fetching an elements.

3 Experimental results

3.1 What happens when ports are missing

Figure 2 shows how long it takes for each client to start downloading a file. The x axis is the simulation time, the y axis is the time it takes to receive the first byte of the element. In this figure, we used three clients, each with one different starting point as explained in 2.2. Each client was using 15 threads to download pages. The AFTR was configured to use 500 ports, dynamically allocated to the clients.

Each connection from a client to a remote server uses one port on the AFTR. However, when the connection is closed, the port on the AFTR is not immediately marked as free and reusable. This is the normal behaviour of TCP: some segments may be late, in the wrong order, and thus arrive after the FIN packet that closed the connection. Therefore, even after the connection is closed, the AFTR must be prepared to handle segments belonging to this connection. As a result, even if more there are never more than $3 \times 15 = 45$ connections opened simultaneously, there are many more than 45 ports used of the AFTR. This is why, with 500 ports, we can clearly see a lot of requests that can not be satisfied, leading to a very poor user experience.

When the AFTR can't give out a port to a client, it has no way to warn it that it can't let the connection through. A *Source Quench* ICMP message would mean to slow down the current connection, not the rate of new TCP connections. Maybe an RST response or a *Port Unreachable* ICMP message forged by the AFTR would be the closest matches, but would force the customer's computer to give up connecting to the remote host altogether. As a result, the only indication for the client that the initial SYN was dropped is the lack of a SYN+ACK response. From the point of view of the end-user, all she can see is a long delay in contacting the server, without any feedback.

We see on figure 2 that a lot of requests are satisfied in 3 (resp. 9, 21, 45, etc.) seconds. These requests correspond to connections where the 1st (resp. the 2nd, 3rd, etc.) SYN segment from the client was

Starting point	Planet Debian	Twitter	del.icio.us
Number of pages visited	150	162	162
Total number of HTTP objects to fetch w/o a cache	2835	7522	1856
Total number of HTTP objects effectively fetched	2257	3489	598
Average number of elements per page	18.9	46.4	11.45
Different domain names seen	208	124	69
Number of pages under the initial domain name	4 (incl. `*.debian.org`)	38	140 (incl. `delicious.com`)
Requests made over HTTPS	106	114	23

Table 1: Characteristics of a browsing session

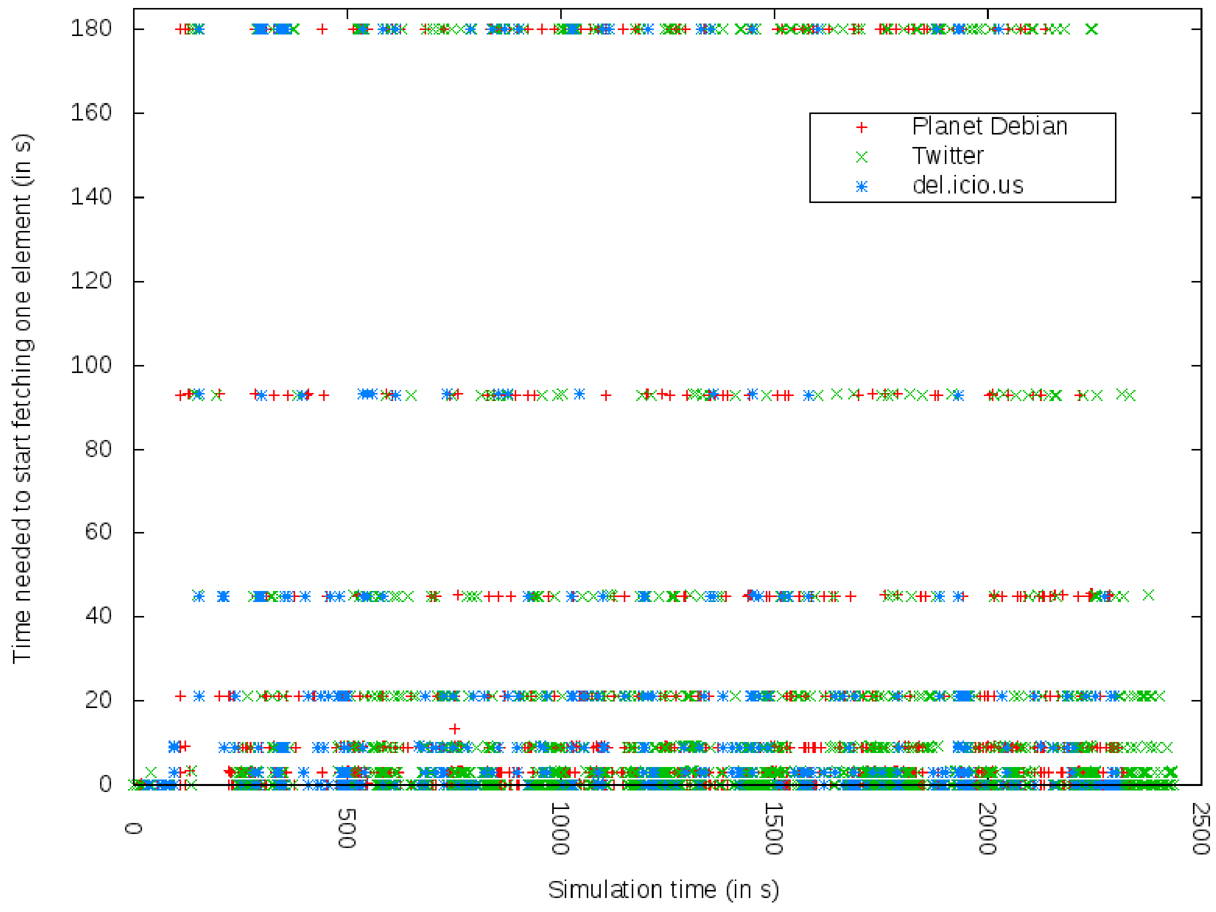

Figure 2: Time needed to fetch elements —500 ports available, 3 simulated clients

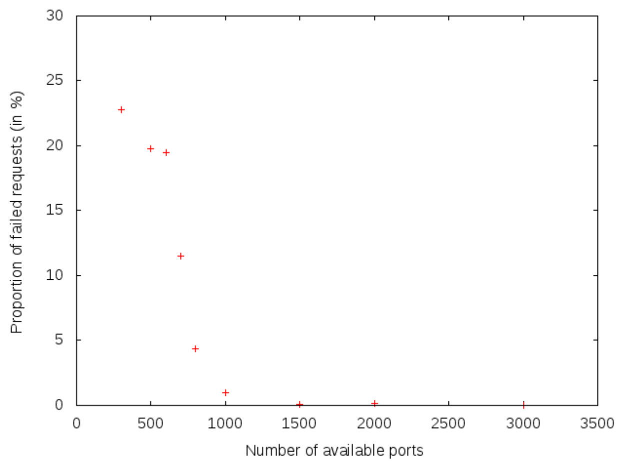

Figure 3: Proportion of failed connections ($\tau = 5$ seconds)

dropped by the AFTR, but the 2nd attempt (resp. the 3rd, 4th, etc.) went through, and the remote server answered quickly (in less than 100 ms). Linux' strategy for retransmitting the initial SYN packet is an exponential back-off[7]. The exact formula used for the retransmission time T_n of the nth packet is $T_n = (2^n - 1) \times 3$ seconds. The first elements of this sequence are 3, 9, 21, 45, 93 seconds; indeed, we can observe clearly marked stages at these periods, with an upper bound at 180 seconds, where we give up trying to connect to the remote host.

3.2 Number of failed attempts

When a web page cannot be accessed, the end-user can't distinguish between an AFTR starving for TCP ports and a slow or dead remote server. Her web surfing is disrupted, and this makes for an unhappy customer. We decide that a web object (the HTML page itself, or an embedded image) can't be fetched based on the impatience of the user. In the following, we will set the threshold after which a request is considered a failure at $\tau = 5$ seconds. This is half the time users can be expected to wait, and well above the comfort zone[8].

Figure 3 shows the number of elements who took

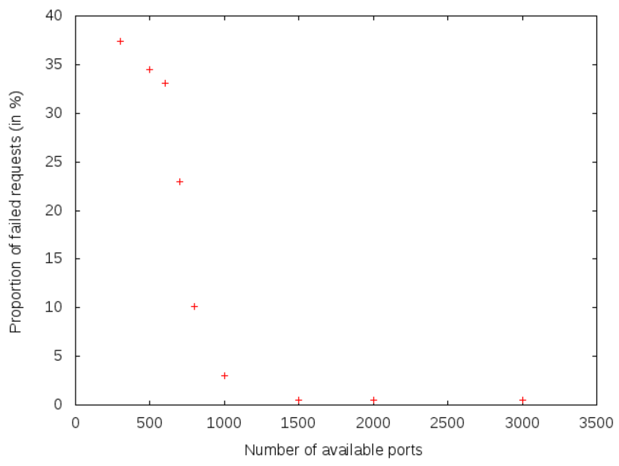

Figure 4: Proportion of failed connections ($\tau = 2.9$ seconds)

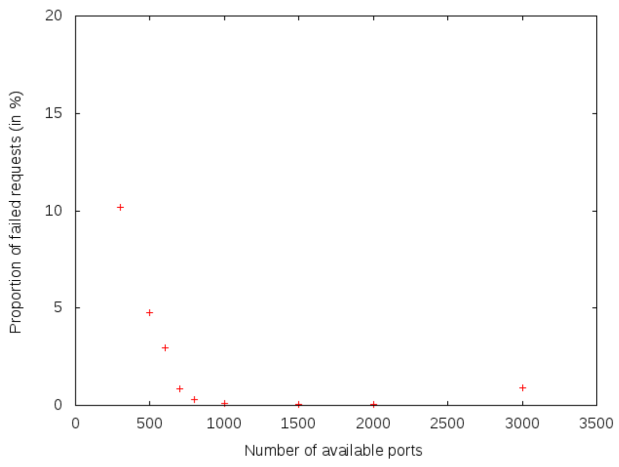

Figure 5: Proportion of failed connections ($\tau = 170$ seconds)

more than 5 seconds to load (or rather, the proportion of TCP connections who took longer than this delay to be established). For comparison, figure 4 shows the same thing, but with a failure threshold set at 2.9 seconds, just shy of Linux' initial TCP timeout. Lastly, figure 5 uses a threshold of 170 seconds; in other words, it counts the number of elements that we completely gave up trying to fetch.

We see that the three curves have a very similar shape (but pay attention to the scale of the y-axis). In all cases, the proportion of failed connections start to increase sharply as soon as the number of available ports goes under 1000 ports per 3 clients. It means that as soon as the pressure for the attribution of ports is strong enough to cause the AFTR to drop even a few SYN segments, any small increase in the demand of ports will have a strong impact on the user experience and on the failure rate. We therefore reccomend to size the system so that no SYN segment is ever dropped by the AFTR.

3.3 Number of retries for establishing a TCP connection

As seen above, Linux uses deterministic, discrete timeouts when waiting for a SYN+ACK answer to its first SYN. This enables us to easily determine the number of attempts the client had to made to open a TCP connection, by matching the time needed against discrete stages. When a connection can be established in less than 3 seconds, it means the first SYN segment went through the AFTR and reached the remote server; when it took between 3 seconds and 9 seconds, it means the first SYN segment was dropped, but the second try was successful; and so forth.

As before, the client gave up after 180 seconds. In other words, it tried to send at most 6 SYN segments before considering the remote server unreachable.

Figure 6 shows the proportion of TCP connections which needed a certain number of attempts to be established. For instance, we can see that when more than 1000 ports are available, 95% of TCP connections can be established on the first try, and about 2% could not be established at all.

4 Conclusions

The Dual Stack Lite architecture doesn't handle gracefully port starvation. Any connection attempt that can't be satisfied by the AFTR results in a very poor user experience, with at least a 3-second long delay without any feedback. Such a delay leads to unresponsiveness of web-based interfaces, which are becoming prevalent.

As we saw in section *Number of failed attempts*, the number of failures before a successful connections rises very quickly when the number of available ports goes below a certain threshold. The key to providing a pleasant user experience is therefore to size the ISP network so that an AFTR always has enough ports to satisfy all connection requests on the first try.

With less than 700 ports available on the AFTR and three clients simultaneously browsing the Web, a significant proportion of the connections need at least 2 SYN segments before reaching the remote server. At 1000 ports and more, with three active clients, the system reaches an equilibrium, where the AFTR is not flooded with connection attempts. We recommend to keep at minimum 500 ports per clients.

From our experiments, we can say that a DS-Lite architecture needs to have about 500 free ports per active client at a minimum. With 65 000 TCP ports available, this means that a single AFTR can handle up to 130 customers simultaneously browsing the web.

5 Acknowledgements

The authors would like to express their gratitude to Laurent Toutain and Tanguy Ropitault for their mentoring, as well as Francis Dupont for his support around the AFTR dæmon.

References

[1] M. Bagnulo, P. Matthews, and I. van Beijnum, *Stateful NAT64: Network address and protocol translation from IPv6 clients to IPv4 servers*, February 2011, Internet Standards Track. `http://www.rfc-editor.org/authors/rfc6146.txt`.

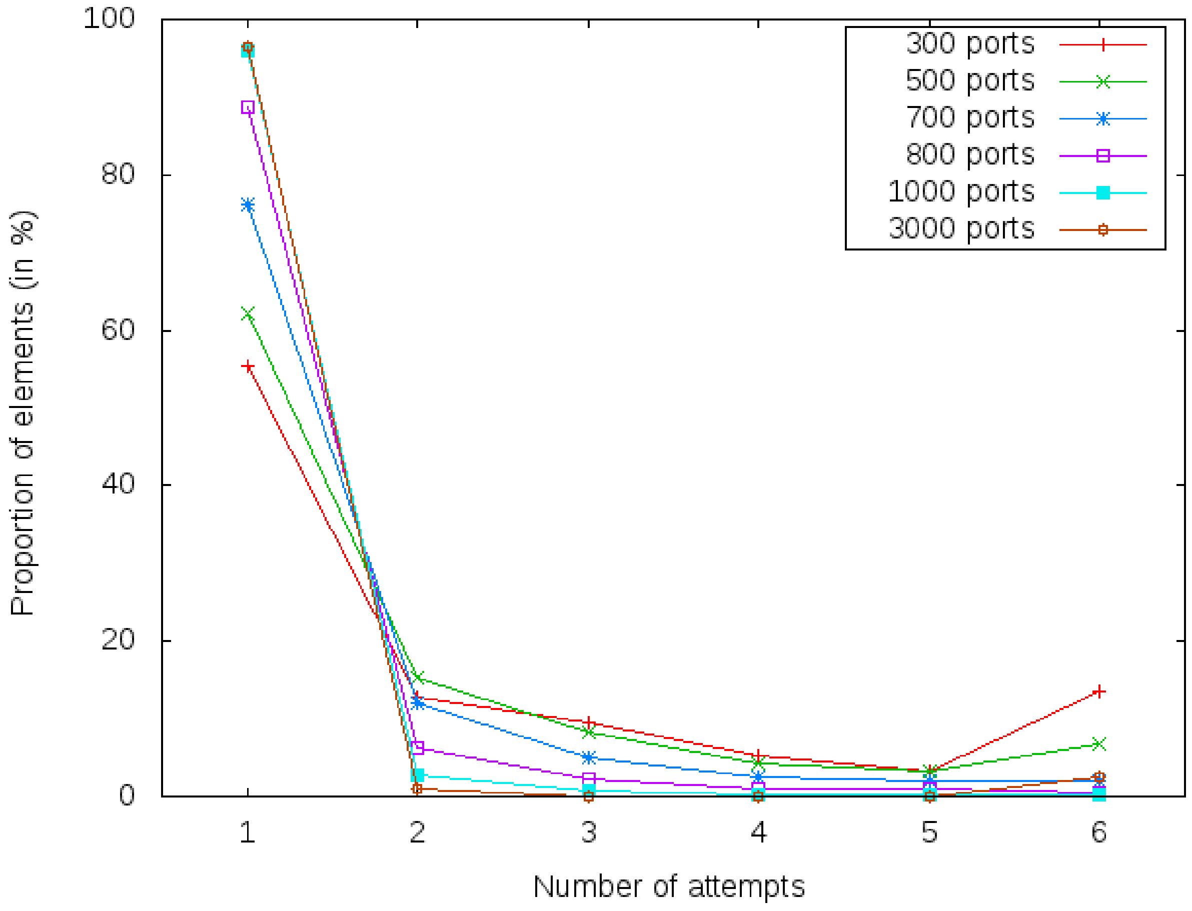

Figure 6: Number of attempts before a successfull connection (3 clients)

[2] M. Bagnulo, A. Sullivan, P. Matthews, and I. van Beijnum, *DNS64: DNS extensions for network address translation from IPv6 clients to IPv4 servers*, March 2011, Internet Standards Track. `http://www.rfc-editor.org/authors/rfc6147.txt`.

[3] F. Dupont, *The AFTR dæmon*, `http://www.isc.org/software/aftr`.

[4] A. Durand, R. Droms, J. Woodyatt, and Y. Lee, *Dual-Stack Lite broadband deployments following IPv4 exhaustion*, March 2011, Internet-Draft. `http://tools.ietf.org/html/draft-ietf-softwire-dual-stack-lite-07`.

[5] B. Grelot and F. Fourcot, *Migrating to IPv6 with Address+Port translation*, Rapport de projet SLR, Télécom Bretagne, March 2010.

[6] I. Kraemer and F. Perrin, `surf.pl` *and* `resurf-socket.pl`, 2011, `http://svn.fperrin.net/v6fication/crawler`.

[7] D. Lukowski et al., `net/tcp_timer.c`*:* `retransmits_timed_out()`, August 2009, Linux kernel. `http://kernel.org`.

[8] J. Nielsen, *Response time limits*, 1993, `http://www.useit.com/papers/responsetime.html`.

[9] J. Yamaguchi, Y. Shirasaki, A. Nakagawa, J. Yamaguchi, and H. Ashida, *NAT444*, January 2011, Internet-Draft. `http://tools.ietf.org/html/draft-shirasaki-nat444-03`.