On the Cost of Using Happy Eyeballs for Transport Protocol Selection

Giorgos Papastergiou[†], Karl-Johan Grinnemo[‡], Anna Brunstrom[‡], David Ros[†], Michael Tüxen[⋆], Naeem Khademi[*], Per Hurtig[‡]

[†]Simula Research Laboratory, [‡]Karlstad University, [⋆]Fachhochschule Münster, [*]University of Oslo

{gpapaste, dros}@simula.no, {karl-johan.grinnemo, anna.brunstrom, per.hurtig}@kau.se, tuexen@fh-muenster.de, naeemk@ifi.uio.no

ABSTRACT

Concerns have been raised in the past several years that introducing new transport protocols on the Internet has become increasingly difficult, not least because there is no agreed-upon way for a source end host to find out if a transport protocol is supported all the way to a destination peer. A solution to a similar problem—finding out support for IPv6—has been proposed and is currently being deployed: the *Happy Eyeballs* (HE) mechanism. HE has also been proposed as an efficient way for an application to select an appropriate transport protocol. Still, there are few, if any, performance evaluations of transport HE. This paper demonstrates that transport HE could indeed be a feasible solution to the transport support problem. The paper evaluates HE between TCP and SCTP using TLS encrypted and unencrypted traffic, and shows that although there is indeed a cost in terms of CPU load to introduce HE, the cost is relatively small, especially in comparison with the cost of using TLS encryption. Moreover, our results suggest that HE has a marginal impact on memory usage. Finally, by introducing caching of previous connection attempts, the additional cost of transport HE could be significantly reduced.

CCS Concepts

•**Networks** → **Transport protocols; Network performance evaluation;**

Keywords

Transport-protocol selection, Happy Eyeballs, TCP, SCTP, TLS, CPU load, memory usage.

1. INTRODUCTION

The deployment of new transport protocols on the Internet is not a trivial task. Several hurdles have to be cleared before a new transport can be used between an arbitrary pair of end hosts and see wide adoption. One of the main issues that has to be solved is, how can an end host *know* if a new protocol X is supported along the whole end-to-end path, including the remote host? In the absence of a priori knowledge or explicit signaling, the only way to know whether X works is to *try* it.

Testing a set of candidate protocols can be done serially—e.g., try first with the preferred choice X and, if the attempt fails after a suitable timeout, then fall back to a default alternative Y. Since a connection timeout can introduce a delay of up to tens of seconds, serializing attempts can incur a large latency penalty when the new protocol X is not supported, stalling the application until the subsequent connection trial succeeds.

The *Happy Eyeballs* (HE) mechanism was introduced as a means to facilitate IPv6 adoption [13]. Dual-stack client applications should be encouraged to try setting up connections over IPv6 first, and fall back to using IPv4 if IPv6 connection attempts fail. However, serializing tests for IPv6 and IPv4 connectivity can result in large connection latency. Happy Eyeballs for IPv6 minimizes the cost in delay by *parallelizing* attempts over IPv6 and IPv4.

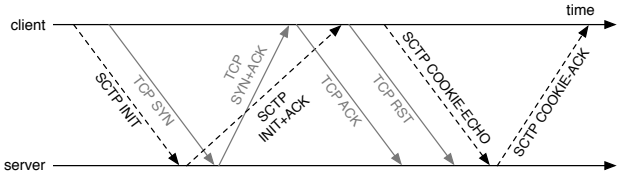


Figure 1: HE for selecting between SCTP and TCP, with SCTP being the preferred choice.

The basic idea behind Happy Eyeballs for IPv6 can be extended to discover support for transport protocols, and in particular to allow an application to use SCTP when it is available end-to-end, else revert to TCP when it is not [11,12]. Figure 1, adapted from [12], depicts how HE for transport selection may work. An end host simultaneously initiates a TCP connection and an SCTP association; it is assumed that the SYN+ACK arrives before the INIT-ACK. If SCTP (the preferred choice) is supported, the TCP connection is abandoned[1]. Similar to what is done in actual

[1]The figure assumes there is a mechanism to notify the application about the reception of the SCTP INIT-ACK, so the

ANRW '16, July 16 2016, Berlin, Germany

ISBN 978-1-4503-4443-2/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2959424.2959437

implementations of HE for IPv6 [10], a small delay may be introduced to give an advantage to SCTP over TCP [14]; else, the host can just pick the protocol for which a response (TCP SYN+ACK, SCTP INIT-ACK) arrives first.

This paper focuses on the performance penalties introduced by HE for transports. In particular, we study the impact that a HE mechanism for selecting between TCP and SCTP may have on CPU load and memory usage at a destination end-host. Our results provide empirical arguments in favor of using such a mechanism for transport selection.

The rest of the paper is organized as follows. Section 2 provides some background and motivation for introducing a transport HE mechanism. As follows from this discussion, two major concerns about transport "happy-eyeballing" are higher CPU load and memory usage. In Section 3 we assess these concerns by experimentally evaluating HE between TCP and SCTP. Finally, Section 4 concludes the paper.

2. BACKGROUND

Some thirty years back, there were two Internet transport protocols to choose from: TCP and UDP. Since these two protocols basically represent the opposites in the services they provide—TCP provides a reliable, in-order, bytestream oriented delivery service and UDP an unreliable, unordered message delivery—the selection between these two protocols from an application viewpoint was mostly straightforward. Furthermore, it could be expected that all hosts supported both TCP and UDP, and there were no middleboxes altering or blocking the traffic before it reached its final destination.

Today, the Internet looks rather different. The number of standard transport protocols and their options (and the different services they may provide) has increased, making the selection of a suitable transport less straightforward. New transports may allow to provide improved services to applications, but middleboxes such as firewalls, NATs and load balancers have become an integral part of the Internet, and there is a great diversity in how they are configured and deployed; it cannot be assumed that any transport or transport option can safely make it from sender to receiver.

As mentioned in Section 1, although HE was primarily introduced as a way to promote the use of IPv6, it has also been proposed as a way for an application to efficiently select transports [11, 12]. Wing and Yourtchenko [11] provide recommendations for HTTP clients on how to seamlessly migrate from TCP to SCTP without any adverse impact on the user experience. Moreover, they propose a way to combine an IPv6/IPv4 HE with a TCP/SCTP HE for a web browser running on a dual-stack machine [12]. Also worth mentioning in this context is the work carried out by the Transport Services (TAPS) working group of the IETF [8]. One of this working group's planned documents should "[...] explain how to select and engage an appropriate protocol and how to discover which protocols are available for the selected service between a given pair of end points [...]", something which will likely require HE between transport solutions.

Still, a HE transport-selection mechanism does raise questions about increased CPU usage and memory consumption.

application can then abort the TCP connection by sending a TCP Reset. Without such mechanism, the TCP connection can only be aborted after the full four-way handshake of SCTP is completed.

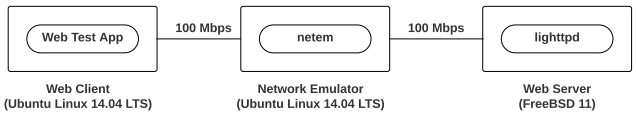


Figure 2: Experiment setup.

When HE is used, a single connection request from the application might result in several concurrent transport connection requests, i.e., not just one connection request at a time as is the case when HE is not used. Hence, the use of HE could result in an increase in both CPU and memory usage. Baker [3] provides recommendations on how to evaluate IPv4/IPv6 HE, however, metrics like CPU load and memory usage are not considered in [3]. Note that neither of these are key metrics for IP HE, whereas they are for transport HE—this is so because transport connection setup means creating state in end points. There have been a few discussions of the performance of HE for IPv6/IPv4 [1, 2, 4, 7] but, to the best of our knowledge, this paper is the first to focus on performance aspects of transport-layer HE, in terms of CPU load and memory usage.

A HE transport-selection mechanism also raises questions about increased use of network resources, a key issue for the scalability of HE. For instance, the aforementioned HE proposal by Wing et al. [12] transmits four packets for every application connection request. Still, as already pointed out by Wing et al. [11, 13, 14], HE network resource usage should be mitigated by the use of caching.

3. EVALUATION

This section evaluates HE between TCP and SCTP in terms of CPU load and memory usage. The section begins with a description of the experiment setup and the studied test scenarios. The remainder of the section presents and comments on the results from the execution of these scenarios.

3.1 Experiment Setup

In our experiment, we modeled a single wide-area network path to an upstream Web server. The laboratory network used in our experiment is shown in Figure 2. The three machines in the experiment were of type: Dell Optiplex 9020 with 3.60 GHz Intel Core i7-4790 (quad core) processors. The Web Client and Network Emulator machines ran Ubuntu Linux 14.04 LTS with kernel 3.13.0, and the Web Server machine ran FreeBSD 11 (revision r294499). All machines used the default network kernel settings, except those listed in Table 1. These changes assured that the testbed could properly support the connection rates considered in this work, and disabled all SCTP features that were not needed in the experiments. The Network Emulator machine used *netem* to emulate a propagation delay of 20 ms.

The Web Client hosted a custom-designed Web traffic generator in which two modified versions of the *httperf* [6] web traffic generator (one that supports TCP and one that supports SCTP) were combined to implement the studied test scenarios. HTTP/1.0 with the Keep-Alive option enabled was used in both httperf programs. The FreeBSD server hosted a *lighttpd* [9] server, modified to listen for both TCP and SCTP HTTP/1.0 unencrypted and TLS-encrypted requests. The lighttpd server was also modified

Table 1: Kernel Settings

Web Server	Settings
net.inet.tcp.syncache.hashsize	2048
kern.ipc.somaxconn	4096
net.inet.sctp.pr_enable	0
net.inet.sctp.ecn_enable	0
net.inet.sctp.outgoing_streams	1
net.inet.sctp.incoming_streams	1
net.inet.sctp.asconf_enable	0
net.inet.sctp.auth_enable	0
net.inet.sctp.reconfig_enable	0
net.inet.sctp.nrsack_enable	0
net.inet.sctp.pktdrop_enable	0
Web Client	**Settings**
net.ipv4.ip_local_port_range	10000 61000
net.ipv4.tcp_tw_recycle	1
Network Emulator	**Settings**
net.ipv4.ip_forward	1

Table 2: lighttpd Settings

Configuration Parameter	Settings
server.network-backend	writev
server.event-handler	kqueue
server.max-fds	4096
server.max-connections	2048
server.max-worker	7
ssl.use-sslv2	disable
ssl.use-sslv3	disable

so that the Nagle algorithm was disabled on all listen sockets to assure that there were no additional delays in total connection time. The default configuration parameters of the lighttpd server were used, except those listed in Table 2, which assured that the lighttpd server could efficiently handle the HTTP request rates considered in the experiments and that TLS was always preferred. The *OpenSSL* library v1.0.1e was used for the TLS protocol. The preferred cipher suite was ECDHE-RSA-AES128-GCM-SHA256, while Intel's AES New Instructions (AES-NI) set for hardware accelerated AES operations was utilised [5]. The lighttpd server used the FreeBSD kernel SCTP implementation, and the Web traffic generator used the Linux kernel SCTP implementation.

An experiment run lasted for 600 s, during which the Web traffic generator generated exponentially distributed HTTP requests with a fixed average intensity, and with requested Web object sizes of 1 KiB and 35 KiB. In our experiment, we considered HTTP-request intensities ranging between 100 requests/s and 1000 requests/s. We measured:

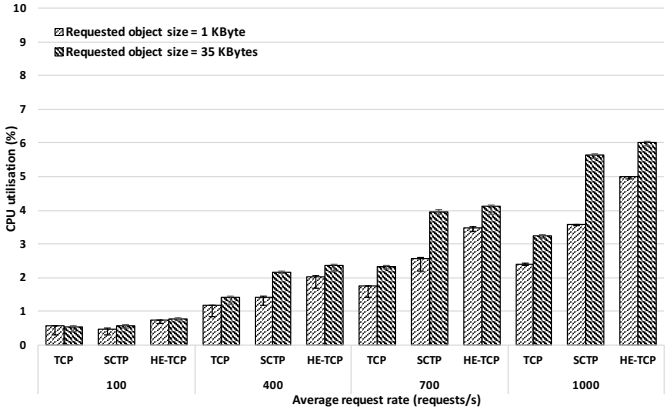
- the total CPU load on the server,
- the CPU utilisation of every process that has a substantial contribution to the total CPU time,
- the total kernel memory used for networking.

Per-process CPU utilisation was sampled every 20 s and was calculated based on the accumulated CPU time given by *procstat -r*. The total CPU load on the server was measured by measuring the accumulated CPU time of the *idle* system process (i.e., the total time that the CPU was idle) and subtracting this time from the total available CPU time during the measured interval (i.e., 160 s for an 8 parallel thread CPU). Total kernel memory utilisation was also sampled every 20 s and was calculated based on the output of *vmstat -z*, *vmstat -m*, and *netstat -m*. Table 3 outlines the malloc types and zones that were used to calculate total network-related kernel memory utilisation.

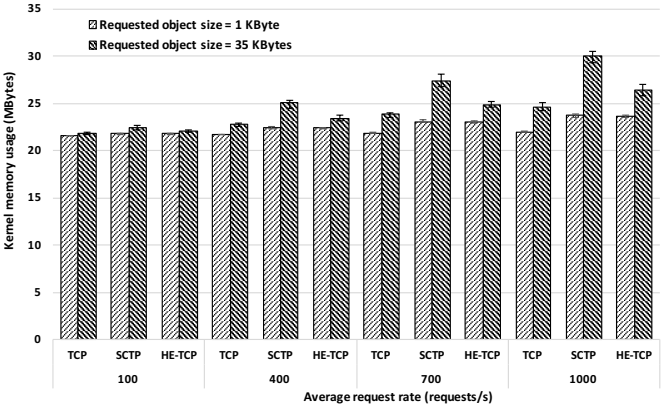
Table 3: Malloc types and zones

Command	Malloc type / Zone
vmstat -m	filedesc, kqueue, ip6opt, ip6ndp, pcb, BPF, ifnet, ifaddr, ether_multi, lltable, routetbl, igmp, in_mfilter, in_multi, ip_moptions, sctp_map, sctp_stri, sctp_stro, sctp_a_it, sctp_atcl, sctp_atky, sctp_athm, sctp_vrf, sctp_ifa, sctp_ifn, sctp_timw, sctp_iter, sctp_socko, hostcache, in6_mfilter, in6_multi, ip6_moptions, mld, inpcbpolicy, ipsecpolicy
vmstat -z	KNOTE, socket, udp_inpcb, udpcb, tcp_inpcb, tcpcb, tcptw, syncache, hostcache, sackhole, tcpreass, sctp_ep, sctp_asoc, sctp_laddr, sctp_raddr, sctp_chunk, sctp_readq, sctp_stream_msg_out, sctp_asconf, sctp_asconf_ack, selfd
netstat -m	mbufs, mbuf clusters, 4k jumbo clusters, 9k jumbo clusters and 16k jumbo clusters

Our experiment comprised three test cases. In the first case, we evaluated a naive HE mechanism that did not employ caching of the outcome of previous happy eyeball invocations and which always resulted in a TCP connection being set up. The rationale behind this case was to serve as a baseline for the remaining two cases. Next, in the second test case, we still considered the same naive HE mechanism as in the first case, however, this time we evaluated happy eyeballing between TLS-encrypted TCP and SCTP. The second test case aimed at providing an appreciation of how the increase in CPU load and memory usage due to happy eyeballing compares with that caused by the TLS encryption itself. Lastly, in the third test case, we evaluated an optimized HE mechanism that employed caching of the outcome of previous connection attempts, using TCP and SCTP both with and without TLS encryption. The purpose behind the third test case was to obtain an understanding of the extent to which HE CPU load and memory usage decrease with caching, and to get a feel for the overhead of HE with a more optimized implementation. In this test case, we considered three different outcomes of the HE mechanism: HE always results in a TCP connection being set up (HE-TCP); HE always results in an SCTP connection being set up (HE-SCTP); and, HE results in a TCP connection being set up half the time and an SCTP connection half the time (HE-50%).



(a) CPU utilisation.



(b) Kernel memory usage.

Figure 3: The results for the basic test case.

3.2 The Basic Test Case

The outcome of the basic test case is shown in Figure 3. Figure 3(a) compares the CPU load of HE between TCP and SCTP with that of single TCP and SCTP connection requests in the basic case. The figure shows how the CPU load varied as a function of the connection request rate and the size of the requested Web objects. The bar charts show the median values measured as described in Section 3.1, with error bars spanning the 10th and 90th percentiles. As follows, the CPU load of HE was quite substantial in the 1-KiB tests, roughly 40% higher CPU load than SCTP (i.e., the transport protocol considered in the study that consumed the most CPU load, in the tests with request rates 1000, 700, and 400 requests/s). However, as is evident from the 35-KiB tests, this was most likely an effect of the small amount of bytes transmitted in each Web response, and thus the small amount of bytes over which the CPU load was amortized: In the 35-KiB tests, the CPU load of HE was less than 10% higher than that of SCTP in the tests with request rates of 1000, 700, and 400 requests/s.

Figure 3(b) examines the kernel memory usage of HE compared with that of single TCP and SCTP connection requests. The bar charts show the median values with error bars spanning the 10th and 90th percentiles. Similar to Figure 3(a), the bar charts illustrate how the kernel memory varied with increasing connection request rates, and for different sizes of the requested Web object. We observe that HE had no or negligible impact on the kernel memory consumption – neither in the 1-KiB tests nor in the 35-KiB tests do we see a significant increase in kernel memory usage. In fact, the 35-KiB tests indicate that as the connection request rate increases, HE (at least in those cases where TCP wins) reduces the kernel memory usage as compared with SCTP.

3.3 Happy Eyeballing in the TLS Test Case

Figure 4 summarises the results from the TLS test case. Figure 4(a) is similar to Figure 3(a), but compares the CPU load in the case with TLS-encrypted connections. We observe that contrary to the basic case, the impact on CPU load of HE as compared with SCTP decreases significantly in the 1-KiB tests (less than 13% in all cases) and is not statistically significant in the 35-KiB tests (less than 4% in the tests with request rates 1000, 700, and 400). Similar observations also apply when compared with TCP, where the impact of HE on CPU load is significantly lower than that in the basic case. Again, the reason HE had less effect on CPU load in this scenario compared with the basic case, was an effect of the way the CPU load was amortized.

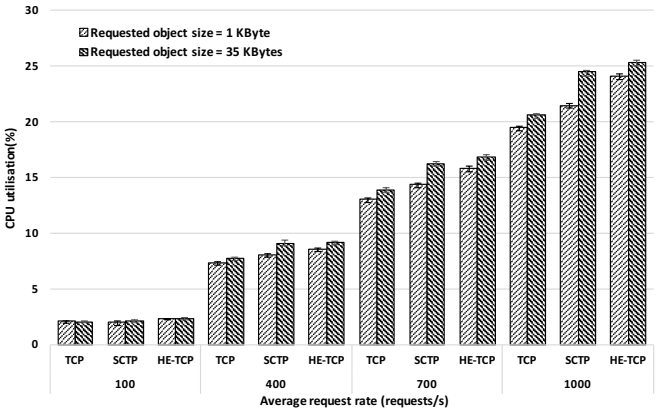
Figure 5 illustrates how the CPU was shared among the kernel (including the FreeBSD subsystem) and the lighttpd server when the HE mechanism is used in the 35-KiB tests; both the tests for the basic case, as well as those for the TLS case. We observe that since the CPU load inflicted by HE was almost the same in both test cases, the CPU load of TLS (as reflected in the increase on the user CPU time of the lighttpd processes) overshadowed that of HE. Thus, in sum, we draw the conclusion that although HE is done at the price of some extra CPU load, the price becomes marginal for larger Web object sizes, and becomes even less significant in those cases HE is done between encrypted connections.

As regards the kernel memory usage in the TLS case, it follows from Figure 4(b) that HE had a marginal impact on this factor in this test case as well: In all tests, the kernel memory usage of HE is slightly higher than that of TCP (less than 8% higher memory usage), and always less than the kernel memory usage of SCTP.

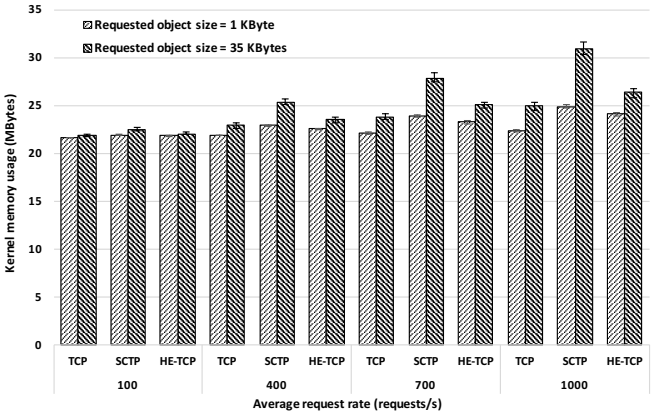
3.4 Happy Eyeballing with Cached Results

In the basic and TLS use cases, we evaluated a naive HE mechanism that always tried both TCP and SCTP. This is, however, a rather inefficient implementation of HE. A more efficient and, as we see it, more realistic implementation would cache the outcome of previous connection attempts. So, e.g., assume that we have a cache hit rate of 80%, then HE tries both TCP and SCTP in only 20% of the application connection requests; in the remaining 80% of the application connection requests, HE issues either a TCP or SCTP connection request depending on the content of the HE cache. A cache hit rate of 80% is actually not an unreasonable figure. In statistics we obtained from Mozilla, they observed a hit rate of $\approx$ 84% in the Firefox internal 'route' cache during a six-week observation period.

Figure 6 shows the median CPU load (with error bars spanning the 10th and 90th percentiles) of HE at different



(a) CPU utilisation.



(b) Kernel memory usage.

Figure 4: The results for the TLS test case.

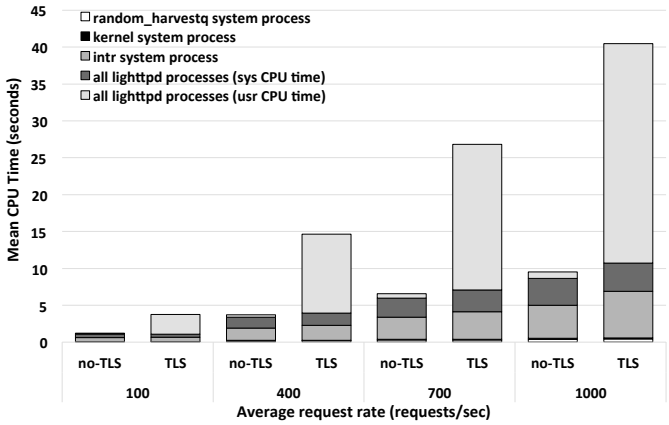


Figure 5: Breakdown of CPU utilisation for happy eyeballing between TCP and SCTP. Requested object size is 35 KiB.

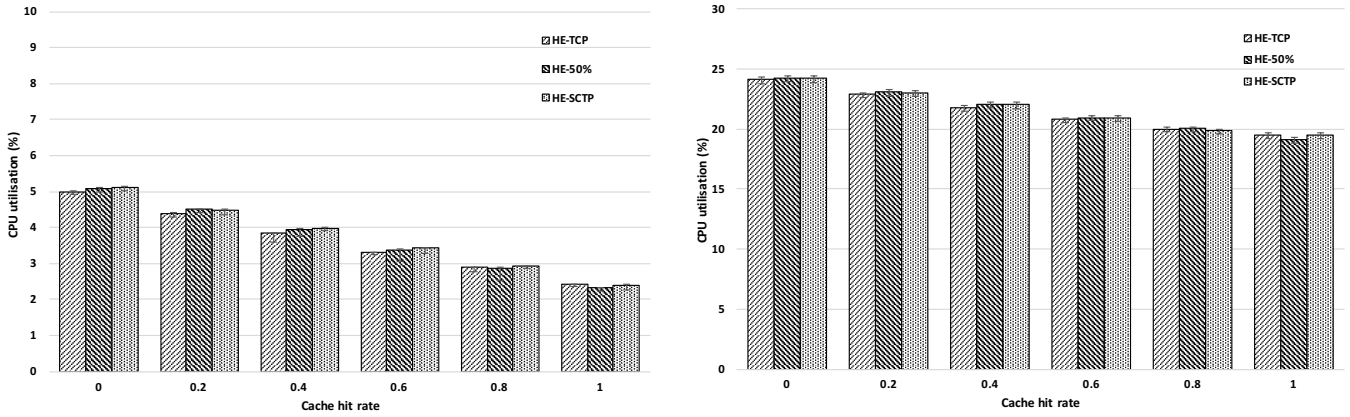
cache hit ratios between 0% (naive HE that always tries both TCP and SCTP) and 100% (single TCP flows) in the 1-KiB tests: Figure 6(a) shows the total CPU load when TLS is not used, and Figure 6(b) when TLS is used. Figure 7 shows the corresponding results for the 35-KiB tests. Since the CPU load increases with increasing connection request rates and is thus more pronounced at higher request rates, we only consider the tests with a request rate of 1000 requests/s in Figures 6 and 7. Still, it should be noted that similar results were obtained for the lower connection request rates. The HE tests considered three outcomes: HE always results in a TCP connection being setup (HE-TCP); HE always results in an SCTP connection being setup (HE-SCTP); and, HE results in a TCP connection being setup half the time and an SCTP connection half the time (HE-50%).

We observe that the CPU load of HE decreases linearly as the cache hit rate increases, and this decrease is higher, percentage-wise, when TLS is not used (in the 1-KiB tests about 43% reduction in the CPU load when the cache hit rate is 80% and TLS is not used, and 18% reduction for TLS encrypted connections and the same cache hit rate). Again, the reason caching had less effect on the decrease of the CPU load when TLS was used, was the effect of the way the CPU load was amortised. We further observe in Figure 6 that irrespective of whether TLS is being used or not, the difference in the CPU load imposed by HE-TCP, HE-50, and HE-SCTP is negligible for the 1-KiB tests. This implies that for small objects the additional cost of the http transaction is almost the same for both TCP and SCTP. For larger objects (e.g., 35 KiB), however, this cost is higher when SCTP is used, and hence significant differences between HE-TCP, HE-50, and HE-SCTP are observed at low cache hit rates in Figure 7.

We omit showing how the cache hit ratio influences the kernel memory consumption, since already the basic and TLS use cases suggest that kernel memory usage is not much of an issue for HE. Still, for completeness, we can mention that the cache hit ratio also had a positive impact on kernel memory usage. For instance, in the 1 KiB tests the kernel memory usage of HE at a cache hit rate of 80% was pretty much the same as for single TCP flows.

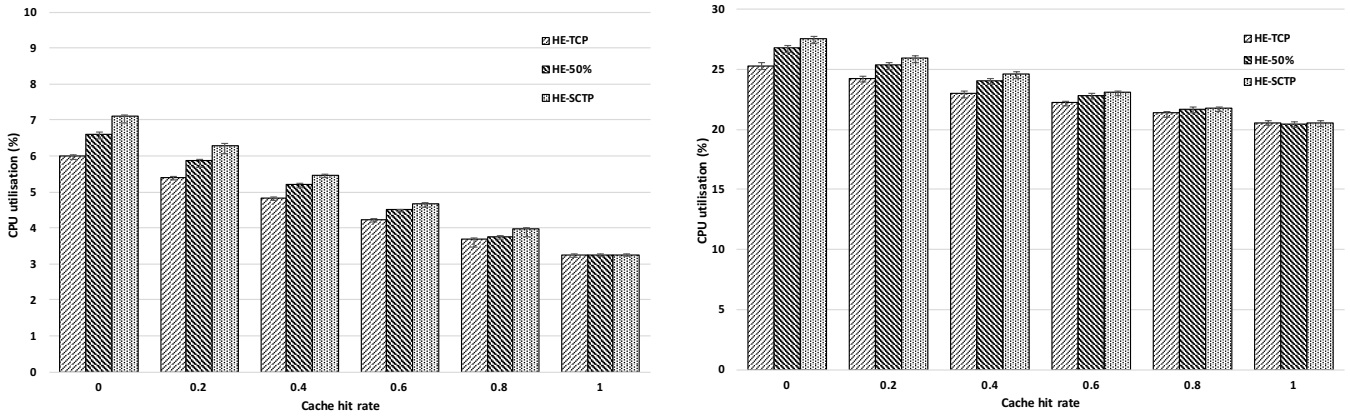
4. CONCLUSIONS

The Happy Eyeballs algorithm was originally proposed, and is currently being deployed, as a way of making a smooth transition from IPv4 to IPv6. However, the algorithm has also been proposed as a transport-selection mechanism. This paper evaluates happy eyeballing between TCP and SCTP, and shows that although HE increases CPU load as compared with a single TCP or SCTP connection establishment, the increase is in the order of 10% for 35 KiB Web objects, i.e., fairly typical Web objects, and is even smaller in those cases the happy eyeballing takes place between TLS-encrypted connections. Moreover, we show that the caching of connection-request results substantially reduces the HE CPU load, especially in comparison with the cost of TLS. As regards memory usage, our results suggest that HE has essentially the same memory footprint as single TCP/SCTP flows. The analysis in this paper shows that integrating a Happy Eyeballs mechanism into a library which provides a generic transport service is indeed a viable option to enable the use of advanced transport-protocol features whenever they are available. An example of such a library is the `neat`



(a) Basic scenario. (b) TLS scenario.

Figure 6: Impact of cache hit ratio on CPU utilisation. Requested object size is 1 KiB with a request rate of 1000 requests/s.



(a) Basic scenario. (b) TLS scenario.

Figure 7: Impact of cache hit ratio on CPU utilisation. Requested object size is 35 KiB with a request rate of 1000 requests/s.

library currently being developed by the NEAT project[2].

Future work includes evaluating the performance of HE when there are more than two transport solutions to be tried, e.g., TLS or DTLS encrypted traffic using IPv4 or IPv6 as the network layer, and TCP, native SCTP, or UDP-encapsulated SCTP as the transport layer, giving a total of six protocol candidates. Already, we note that the use of caching becomes even more important in these cases. Future work would also consider real-world experiments, where middlebox interference can be taken into account, as well as additional metrics to further examine the effects of transport happy eyeballing on both the network and destination end hosts, such as resource consumption on middleboxes, network load, and transaction times.

5. ACKNOWLEDGMENTS

The authors would like to thank Patrick McManus (Mozilla) for providing the Firefox cache-hit statistics.

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644334 (NEAT). The views expressed are solely those of the authors.

[2] `https://github.com/NEAT-project/neat`

6. REFERENCES

[1] E. Aben. Hampering Eyeballs – Observations on Two "Happy Eyeballs" Implementations. RIPE NCC, Nov. 2011. `https://labs.ripe.net/Members/emileaben/hampered-eyeballs`.

[2] V. Bajpai and J. Schoenwaelder. Measuring the effects of happy eyeballs. Internet Draft draft-bajpai-happy, work in progress, July 2013. `https://tools.ietf.org/html/draft-bajpai-happy`.

[3] F. Baker. Testing Eyeball Happiness. RFC 6556 (Informational), Apr. 2012.

[4] O. Bonaventure. Happy eyeballs makes me unhappy..., Dec. 2013. `http://perso.uclouvain.be/olivier.bonaventure/blog/html/2013/12/03/happy.html`.

[5] S. Gueron. Intel® Advanced Encryption Standard (AES) New Instructions Set. *Intel Corporation*, 2012.

[6] httperf. The httperf page on SourceForge. https://sourceforge.net/projects/httperf.

[7] G. Huston. Bemused Eyeballs: Tailoring Dual Stack Applications for a CGN Environment. The ISP Column, May 2012. http://www.potaroo.net/ispcol/2012-05/notquite.html.

[8] IETF. Transport Services (taps) Working Group. https://datatracker.ietf.org/wg/taps/charter/.

[9] Lighttpd. Lighttpd – fly light. https://www.lighttpd.net.

[10] D. Schinazi. Apple and IPv6 — Happy Eyeballs. Email to the IETF v6ops mailing list, July 2015. https://www.ietf.org/mail-archive/web/v6ops/current/msg22455.html.

[11] D. Wing and A. Yourtchenko. Happy Eyeballs: Trending Towards Success (IPv6 and SCTP). Internet Draft draft-wing-tsvwg-happy-eyeballs-sctp-02, work in progress, Oct. 2010. https://tools.ietf.org/html/draft-wing-tsvwg-happy-eyeballs-sctp-02.

[12] D. Wing and A. Yourtchenko. Improving User Experience with IPv6 and SCTP. *The Internet Protocol Journal*, 13(3), Sept. 2010. http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-49/133-he.html.

[13] D. Wing and A. Yourtchenko. Happy Eyeballs: Success with Dual-Stack Hosts. RFC 6555 (Proposed Standard), Apr. 2012.

[14] D. Wing, A. Yourtchenko, and P. Natarajan. Happy eyeballs: Trending towards success (IPv6 and SCTP). Internet Draft draft-wing-http-new-tech, work in progress, Aug. 2010. https://tools.ietf.org/html/draft-wing-http-new-tech-01.