PEP-DNA: A Performance Enhancing Proxy for Deploying Network Architectures

Kristjon Ciko, Michael Welzl, Peyman Teymoori
University of Oslo, Norway
{kristjoc, michawe, peymant}@ifi.uio.no

***Abstract*—Deploying a new network architecture in the Internet requires changing some, but not necessarily all elements between communicating applications. One way to achieve gradual deployment is a proxy or gateway which "translates" between the new architecture and TCP/IP. We present such a proxy, called "Performance Enhancing Proxy for Deploying Network Architectures (PEP-DNA)", which allows TCP/IP applications to benefit from advanced features of a new network architecture without having to be redeveloped. Our proxy is a kernel-based Linux implementation which can be installed wherever a translation needs to occur between a new architecture and TCP/IP domains. We discuss the proxy operation in detail and evaluate its efficiency and performance in a local testbed, demonstrating that it achieves high throughput with low additional latency overhead. In our experiments, we use the Recursive InterNetwork Architecture (RINA) and Information-Centric Networking (ICN) as examples, but our proxy is modular and flexible, and hence enables realistic gradual deployment of any new "clean-slate" approaches.**

***Index Terms*—RINA, ICN, Performance Enhancing Proxies**

I. Introduction

With the ever-growing number of connected devices and data traffic, the current Internet architecture is facing major issues related to security, multi-homing, mobility and Quality of Service (QoS). New network architectures have been proposed, each attempting to solve these issues in its own way [11], [19], [26]. There is, however, not much evidence of real-life usage of new network architectures. The Internet, as the backbone of much of today's networking infrastructure, is often found to be rigid and hard to replace, requiring careful downward-compatibility considerations for new ideas [25].

When all infrastructural elements—the sender, receiver, and all routers on a path—support a new architecture, it is in fact possible to directly "switch over" and fully exploit the new communication model as a result. This can be done with minimal performance penalty, using RINA [8], [11]. McCauley et al. tackle deployment in depth in [23], culminating in the design of a system which provides the necessary bootstrapping. They assume that traffic will be tunneled over legacy Internet domains for downwards compatibility. However, tunneling can be a deployment hurdle: it needs operators to upgrade endpoints and ensure that "new" traffic does not "harm" (and is not "pushed aside" by) other Internet traffic.

The authors were part-funded by the Research Council of Norway under its "Toppforsk" programme through the "OCARINA" project (https://www.mn.uio.no/ifi/english/research/projects/ocarina/).

As an alternative to tunneling, in this paper we investigate the possibility of *translating* between the Internet and an alternative network architecture. We present the design, implementation and performance evaluation of PEP-DNA: a TCP proxy which enables TCP/IP applications to send traffic over a network path that either partially or fully follows a different network architecture (and vice versa: instead of tunneling, traffic dynamics can be translated into an Internet-compatible TCP behavior). PEP-DNA is a kernel-based implementation designed for Linux-based servers/routers. It follows the main principles of a "split connection" Performance Enhancing Proxy (PEP), described in RFC3135 [5]. Our proxy achieves high throughput with low latency and can be installed wherever a translation from one technology to another occurs—e.g., in an end host, a network provider's edge router or a border router between two different domains.

PEP-DNA is currently able to interconnect a TCP connection with (i) another TCP connection, (ii) the RINA architecture, and (iii) an ICN domain. Its modular and flexible design facilitates the support of other technologies with small changes. We chose RINA and ICN as case studies architectures, considering the potential benefits their deployment could bring in networking. RINA is secured by design [2], and it can natively provide mobility and multi-homing. Moreover, being a recursive and scope-aware architecture enables RINA to naturally support in-network congestion control [29]. Future research could then investigate the efficacy of such a locally scoped congestion control mechanism when it is connected to TCP using our proxy. ICN is another promising architecture for future networks. It is built on the concept of naming information/data rather than hosts. Unlike IP-based networks, ICN endpoints share addressable and routable named content, leading to more flexible, scalable and secure networks [16].

The rest of the paper is structured as follows: in Section II, we provide some background on PEPs and describe related work. Section III reveals details about the main concepts and the operation of our TCP proxy. We provide a comprehensive performance evaluation in Section IV. Finally, Section V concludes the paper. The PEP-DNA implementation code and all the data needed to replicate the experimental results are available at https://github.com/kr1stj0n/pep-dna.git.

II. Background and Related Work

PEPs [5] are on-path devices that perform operations meant to improve the performance. Here, we focus on PEPs at the transport layer, as our own PEP translates between TCP and a new network architecture. PEP-DNA falls in the category of PEPs that split TCP connections: the connection from the sender is terminated and a new connection is established to the receiver or to the next PEP. IP addresses are spoofed to give the sender the impression that it is talking to the receiver, and to give the receiver the impression that it is talking to the sender. These PEPs shorten the Round Trip Time (RTT) of the involved connections, leading to shorter feedback loops and faster increase of the rate as the sender gets ACKs faster.

PEPs can be good: Connection-splitting PEPs can utilize a different congestion control mechanism tailored for a specific path segment, e.g. when a link is wireless, or even translate into an altogether different network architecture, as we propose in this paper. This is particularly useful when the link in question is a poor fit for an end-to-end TCP connection, as it is commonly the case with satellites. XCP-PEP [17] is an example of such a PEP; it uses XCP (eXplicit Control Protocol) congestion control [18] between PEPs while the end hosts use normal TCP. Congestion control "translation" is also done by PEPsal, which was proposed in [7] as an open source solution for satellite communications. PEPsal operates transparently by intercepting the sender's and receiver's packets, and employs TCP Hybla over the satellite link. It is compatible with the old 2.6 Linux kernel. Another proxy called HTTP PEP (HTTPEP) [10] accelerates web browsing in a satellite-based network. HTTPEP optimizes sequential HTTP operations, optimizes the transport layer between the client and the server, and compresses application data.

In [12], the performance impact of the split connection mechanism in Long Term Evolution (LTE) networks is investigated. By using a simple TCP proxy (LTE-PEP) in a simulated LTE network, the authors noted a significant performance improvement due to connection splitting. The *Mobile Accelerator* in [24] transparently splits the connection between a client and a server in order to improve the TCP performance in mobile networks. Virtualized Congestion Control (vCC) [9] and AC/DC TCP (Administrator Control over Datacenter TCP) [14] apply PEP-like mechanisms in hypervisors to achieve better control over the congestion control behavior in multitenant datacenters.

PEPs can also be bad: Despite their ability to improve performance, PEPs are not universally appreciated. Connection splitters harm the end-to-end semantics of TCP: the sender is made to believe that the receiver has received data, even when only the PEP has received it. PEPsal, for example, acknowledges packets prematurely, allowing it to have data available in time when the downstream congestion control needs it. While this is in fact not a huge problem in practice (connection-splitting proxies are widely deployed [30]), there are other, probably more significant disadvantages of PEPs: because they work with TCP, they have to make assumptions about the TCP header, and TCP's operation in general, and can therefore get in the way of upgrading the protocol [15]. This has led to the *ossification* of the Internet's transport layer [25].

Some approaches reduce these negative side-effects by minimizing the clandestine interference with TCP. For example, one of the earliest PEPs, the "snoop protocol" [1], locally cached packets and retransmitted them from the cache instead of informing the sender of packet loss in order to avoid an unnecessary congestion control reaction. In [22], specific PEP ACKs are introduced to notify the server of any packet loss, but it needs changes at the server side.

A challenge of many types of PEPs is that they cannot operate on IPsec traffic because the TCP header is encrypted. In an attempt to overcome this, a PEP called SatERN was presented in [27], which is based on Explicit Rate Notification (ERN) protocols. In the ERN approach, ERN routers inform the senders about the optimal send rate. In particular, SatERN uses XCP as the ERN protocol.

Some see the inability to operate on encrypted traffic as a feature, not a bug: a large part of the header of the QUIC protocol is encrypted to avoid the ossification problem [20]. As a result, there is a dilemma, where QUIC is supposed to generally outperform TCP, but TCP with a connection-splitting PEP can work much better in satellite scenarios [4]. Also, modern wireless communication creates new incentives to use PEPs: millimeter-wave (mmWave) links exhibit severe and quick fluctuations of the capacity that is exposed to upper layers, making it hard for an end-to-end congestion control loop to cope. The (mostly beneficial) impact of connection-splitting proxies has therefore been at the focus of several TCP-over-mmWave investigations (e.g., see 5G's Access Traffic Steering Switching and Splitting (ATSSS) [6] and [13] as well as references therein).

PEP-DNA in this context: It is hard to know how the opposing incentives of end-to-end behavior preservation via encryption versus in-network performance improvement with PEPs will play out in the long run. Possibly, a future approach could be *not* to make proxies transparent, but to make them visible to the end systems instead, such that they can "correctly" participate in communication (instead of spoofing IP addresses). This could also allow to authenticate such network nodes. The "0-RTT TCP convert protocol" [3] shows how such proxies could be done: like the older SOCKS proxy, the PEP (mainly) operates at the application layer, yet it has more control over TCP than most common application-layer processes (it takes TCP extensions into account and exchanges control information during the handshake). Different from a transparent PEP, the proxy is visible in this scenario: rather than directly connecting to a server, a client connects to the proxy and informs it about the server's IP address and port.

Irrespective of how future deployment scenarios will play out, splitting the connection seemed to us to be the only reasonable choice when doing something as radical as translating between the Internet and a completely different network architecture. To the best of our knowledge, PEP-DNA is the first PEP to offer this capability. It is a from-scratch implementation because none of the available alternatives seemed suitable to adjust for our purpose. Table I provides an overview of the PEP implementations that we have discussed in this section. Among the ones that split connections, only PEPsal is

TABLE I: Overview of PEP implementations. “T” means: evaluation in a testbed; “S” means: simulation was used for performance evaluation.

PEPs	Open source	In-kernel implementation	Evaluation	Split connection	No TCP change necessary
PEPsal	✓		T	✓	✓
XCP-PEP			T	✓	✓
HTTPEP			T	✓	
SatERN			S		✓
LTE-PEP			S	✓	✓
Mobile Accelerator			T	✓	
Snoop protocol	✓	✓	T		
Lightweight PEP			S		
vCC	✓[a]	✓	S/T		
AC/DC TCP		✓	T		✓
PEP-DNA	**✓**	**✓**	**T**	**✓**	**✓**

[a]The published open source code only contains a proof of concept implementation for vCC that patches the linux kernel’s TCP stack, whereas the true vCC is implemented in proprietary VMware ESXi hypervisor.

available as open source. Being designed only for IP networks and implemented in user-space, it is however not suitable for our purpose to develop a high performance, architecture-agnostic proxy (e.g., the communication flow in Fig. 2 cannot be achieved with a purely user-space implementation).

III. Proxy Implementation Details

Our proxy, which is implemented as a Linux Kernel Loadable Module, is compatible with recent kernel releases. It operates entirely in the kernel space, directly on top of the TCP/IP stack, and needs no modification of the network stack nor hardware. Although developing network tools in the kernel level comes with a cost of complexity, such designs are more efficient and convenient for a high performance proxy [21]. In-kernel implementations reduce the context switching overhead produced by copying data back and forth between the kernel/user space, thus increasing the overall system performance.

The proxy has three main components integrated in one module, (i) an IP-layer hook based on Netfilter, (ii) the connection handler (a.k.a the connector), and (iii) the data relay engine. Fig. 1 illustrates the three main components of the proxy, along with the necessary steps for establishing a communication channel and sending data from the client to the server, in a TCP-RINA-TCP scenario. In addition, we present in Fig. 2 a detailed sequence diagram of data and signaling packets interaction during the connection establishment and data transfer phases of an inter-domain communication scenario between two TCP applications over a RINA network.

In Step 1 of Fig. 1, the client initiates a connection to the server by sending a SYN packet with destination IP address 6.7.8.9 and destination port 80, which are the address and port

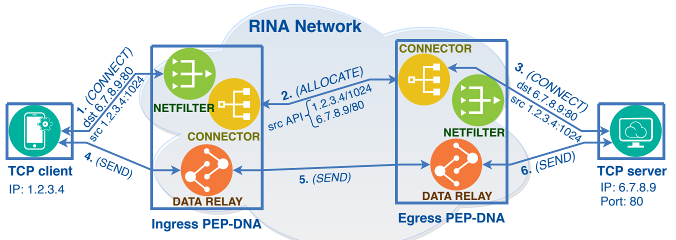


Fig. 1: Inter-domain communication between two TCP applications through proxies deployed at the RINA-IP borders.

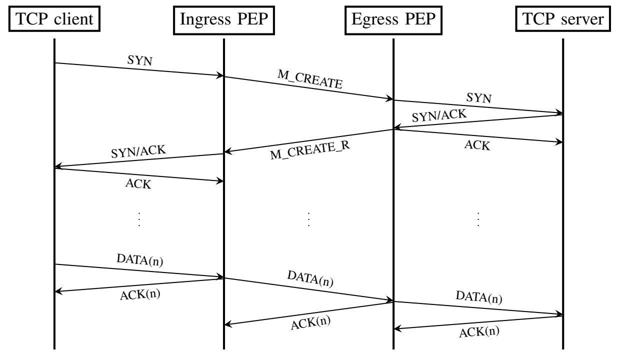


Fig. 2: Connection Establishment when two PEP-DNA proxies interconnect TCP connections over a RINA domain. This assumes that RINA is aware of the IP namespace (it knows how to route to the destination).

the server is listening on. Netfilter allows the proxy to register a callback function within the network stack, which is then called back for every incoming SYN packet. After the Netfilter hook of the Ingress PEP intercepts the SYN packet and stores its 4-tuple (source IP address, source port, destination IP address, destination port) in a hash table, the SYN packet is redirected unchanged by iptables to a local socket of the proxy. The connector immediately initiates connection establishment within the new architecture, illustrated in Step 2. Instead, one flow is allocated for each TCP connection. PEP-DNA leverages the Naming and Addressing principles of RINA, which identify an application by its Application Process Name (APN) and Application Process Instance (API). For every incoming TCP connection request, the proxy spawns a new API to send an “M_CREATE” message for allocating a flow towards the other PEP-DNA (Egress PEP in Fig. 1). The end-point IP addresses and TCP ports are concatenated as a string which represents the name of the API.

In Step 3 the Egress PEP’s connector extracts the end-point information from the requesting API given in the flow request header, initiates a TCP connection to the destination IP address and port of the server, and sends back an M_CREATE_R message as a positive flow response to the Ingress PEP only after the Egress PEP - TCP server connection is established. After receiving the flow response, the Egress PEP sends a SYN/ACK packet to the client. In this way, the proxy connects with both the client and the server as early as possible and also achieves basic *fate-sharing* at the same time (the client is not made to believe that it has a connection before the connection to the server really is established). Once the connection between the client and server is established, the data relay engines of the proxies take care of forwarding data directly in the kernel space, avoiding context switching overhead and allowing zero-copy transmission.

TCP clients connect to the IP address and port of the server and are not aware of the connection splitting performed by the proxies. In the same way, the server receives packets with the source address and port of the client. Fig. 2 shows why implementing our proxy in the achieves the best performance: a user-space proxy would have to complete the handshake of the incoming connection request from the client before it can initiate the connection to the server. This adds delay for other implementations like PEPsal, for example.

Fig. 3 shows a different scenario: here, our proxy is used to integrate TCP applications with ICN networks. We use the CCN-lite[1] implementation of CCN (CCN; Content-Centric Networking is a prominent ICN architecture. We use CCN and ICN interchangeably in this paper.) to simulate a CCN network for this scenario. After establishing a TCP connection with the proxy, the HTTP client requests through the HTTP protocol a content which is identified in the CCN network by the hierarchical name *"/ndn/test/content"*. PEP-DNA parses the HTTP GET request, constructs a CCN request for content, with the interest being *"/ndn/test/content"*, and injects it in the CCN network. Then, the interest request is propagated inside the CCN network until it reaches the CCN Web server where the content is stored. The CCN Web server sends the content back to the request originator, PEP-DNA, which forwards it to the TCP HTTP client. Prior to running the scenario, we configured PEP-DNA as an HTTP proxy in the client host (PEP-DNA needs to translate the application-level request).

The proxy can be adapted to translate to other supported technologies while being transparent to the endpoint. In both RINA's and ICN's cases, we deployed PEP-DNA in a Linux host with a pre-installed RINA stack and CCN-lite, and wrote *"rina.c"* and *"ccn.c"* files with approximately 250 lines of code for each file. Due to this extensible design, our proxy can easily support new network architectures and future promising features, such as racing and fallback mechanisms. There are only two additional requirements in case PEP-DNA is used to interconnect two domains as in Fig. 1: i) finding the right peer (the Ingress PEP must know that it should contact the Egress PEP for the TCP connection to the given server IP address, and the alternative architecture must know how to route to the Egress PEP)—as already mentioned, this is out of the scope of our paper, and has already been addressed in related work [23], and ii) the ability to signal the IP addresses and port numbers of the end points from Ingress PEP to Egress PEP, such that they both can correctly map traffic between a TCP connection and an alternative architecture's representation thereof.

[1]https://github.com/cn-uofbasel/ccn-lite

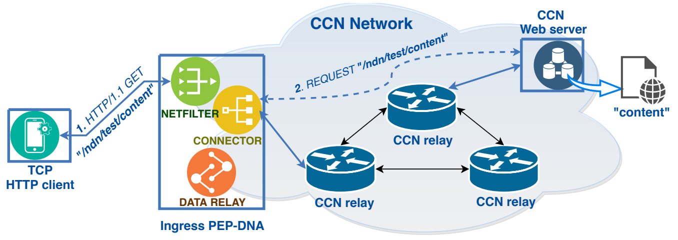


Fig. 3: Interconnection between a TCP HTTP client and a CCN network.

IV. Performance Evaluation

The local testbed that we use to evaluate the performance of PEP-DNA consists of two physical machines, each configured with two Intel E5-2620 processors and 64 GB of physical memory. Both machines run Debian 10 operating system with kernel version 4.19.The two hosts are directly connected to each other through a 10 Gbps Ethernet link. At the server node, we install CCN-lite and the IRATI implementation of RINA [28] as a protocol stack alongside TCP/IP. We choose RINA and ICN as examples of novel network architectures.

We run the experiments for five different scenarios (see Fig. 4): (i) **TCP** *(baseline)*: two TCP applications talk directly via pure TCP, (ii) **TCP-TCP_U**: a simple user-space application takes data from one TCP connection and sends it to another, thereby implementing a very simple user-space PEP, (iii) **TCP-TCP**: two TCP applications communicate via a TCP/IP path which is split by PEP-DNA, running directly on the server host, into two TCP connections, (iv) **TCP-RINA**: the on-host PEP-DNA performs the translation between a TCP-based application and a RINA-enabled application, and (v) **TCP-CCN**: PEP-DNA enables a TCP application to request and receive content from an CCN domain. TCP-TCP_U is meant as a replacement for the state-of-the-art: Table I shows that PEPsal, XCP-PEP, HTTPEP, LTE-PEP and Mobile Accelerator all split connections in user space. Of those, only PEPsal is open source, but its code is older, and it has more overhead than our minimal user-space proxy (PEP-US).

Through the experiments we evaluate the performance of PEP-DNA and test how fast it can process and forward packets between technologies. For this reason, we operate the experiments under high data rates and deploy PEP-DNA at the

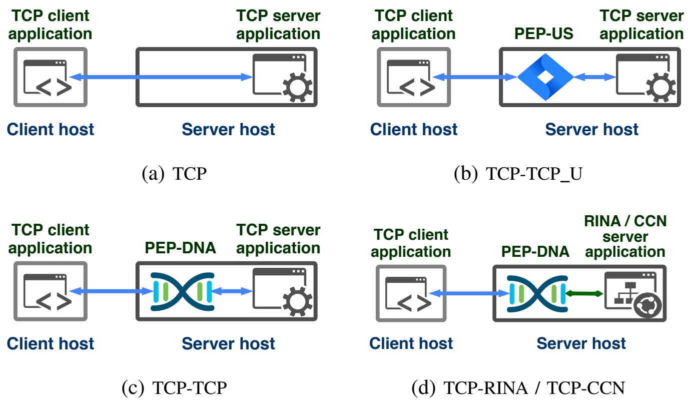


Fig. 4: Different scenarios used to evaluate the performance of PEP-DNA in comparison with TCP baseline and a generic user-space PEP.

same host as the server. This allows the applications to send and receive data as fast as possible. We refrain from long-distance communication with RINA and CCN because we do not want to evaluate the architecture per se.[2] For the CCN test, we used a simple CCN relay (a part of CCN-Lite) as a web server. For all other tests, we used a self-written httping-like[3] application which supports both TCP and RINA and can run as a web client or server.

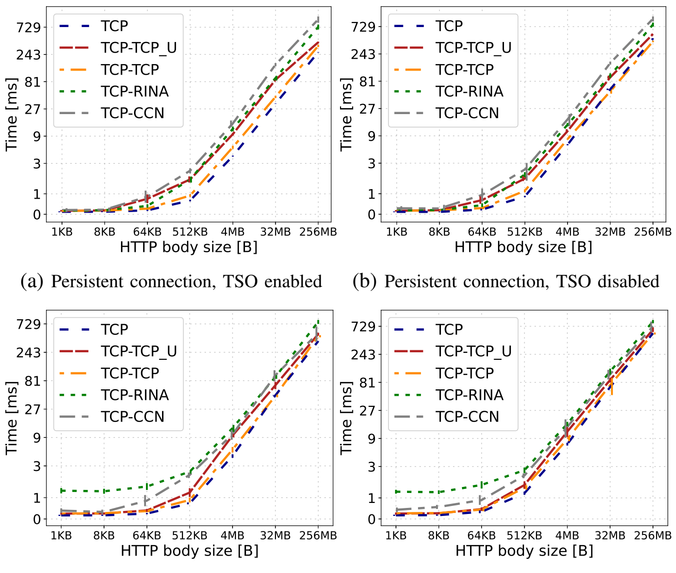
A. *Results and Analysis*

To measure the latency overhead induced by PEP-DNA, we conduct an experiment in which our client application connects to the server instance on port 8080 and sends 100 HTTP HEAD requests with a rate of one request per second. After each request sent, the client waits for the server's response and retrieves only its headers; therefore, during the measurement of the end-to-end application RTT in this experiment the overhead is minimal. Also, we use the IP address of the server rather than the URL to avoid DNS resolution.

Table II shows the mean and standard deviation of application-layer latency measurements for the five scenarios described earlier, using HTTP with both persistent and non-persistent connections. In case of non-persistent connections, each HTTP request and response is sent over a new connection and the measurements include the Establishment, Data Transfer and Termination phases of the connection. The Establishment phase consists of the 3-way TCP handshakes and/or the RINA flow allocation. The Data Transfer phase includes the time when the client sends the request and retrieves the headers of the response. The Termination phase comprises the TCP connection termination and/or the RINA flow deallocation. Since in CCN-lite CCN packets are carried over UDP, there is no connection establishment nor termination phase in CCN. When persistent connections are used, the HTTP messages are sent over a single pre-established connection and the measurements include only the Data Transfer phase. As expected, the results in the case of non-persistent connections are slightly higher because of the initial Establishment phase. When connecting TCP to TCP, PEP-DNA adds a small latency overhead of 0.03 ms compared to the scenario where no proxy is being used. The latency overhead increases slightly up to 0.1 ms when translating to CCN, because of encoding, encryption and caching of contents inside the CCN network. On the contrary, this latency overhead is higher (around 1 ms) when connecting to RINA due to the additional packet processing time during RINA flow allocation and data transfer. In the case of persistent connections, PEP-DNA forwards packets with a low latency, introducing a minimal overhead of 0.01-0.05 ms. In all cases, our proxy slightly outperforms the user-space proxy, even though the latter is minimalistic and does not support any additional features.

[2]We tested the scenario in Fig. 1 as well, and found a significant performance improvement when the RINA segment was the bottleneck. This happened because the only correctly working "congestion control" implementation in the IRATI prototype simply transmits data with a fixed (large) window size. While not telling us much about the performance of PEP-DNA, this test at least shows that it *can* improve throughput by allowing to deploy a more performant network architecture on a path segment.

[3]https://www.vanheusden.com/httping/



(a) Persistent connection, TSO enabled (b) Persistent connection, TSO disabled

(c) Non-persistent, TSO enabled (d) Non-persistent, TSO disabled

Fig. 5: Time between HTTP HEAD requests and corresponding responses. The HTTP body size varies from 1 KB to 256 MB. Experiments were run 100 times, and the graphs show mean and standard deviation values.

In order to observe the throughput and latency behaviour, we conduct two more experiments with our httping-like application. First, we measure the time between a HEAD request being sent from the client to the web server and the corresponding HTTP response. The client sends frequent HEAD requests to the web server with an HTTP body size varying from 1 KB to 256 MB. Fig. 5 shows the results of this experiment when persistent and non-persistent connections are used both with TCP Segmentation Offload (TSO) enabled and disabled. This evaluations confirms that the proxy adds very little overhead. Unsurprisingly, the added delay is slightly larger when translating from TCP to the experimental implementation of CCN, CCN-lite, or the prototypical RINA implementation than when translating to another TCP connection. With non-persistent connections, the delay is larger in RINA because of flow creation, which takes a significant amount of time. The TSO impact is quite minimal. In all cases, the added delay of PEP-DNA is smaller than that of PEP-US.

Second, we measure the Flow Completion Time (FCT) of a 4 GB transfer from the server to the client in five different scenarios: TCP, TCP-TCP_U, TCP-TCP, TCP-RINA, and TCP-CCN. The results for the TSO enabled and disabled cases are shown in Fig. 6. Bars and error bars represent the

TABLE II: Latency results (mean $\mu \pm$ standard deviation σ in milliseconds), obtained from the web client by sending 100 HTTP HEAD requests to the web server with a rate of 1 request/second. NP means non-persistent HTTP connection, in which the results show how long it takes to connect, send the request and retrieve the response. In the persistent connection case (P), we measure the time between sending a request and retrieving the response.

	TCP	TCP-TCP_U	TCP-TCP	TCP-RINA	TCP-CCN
P	0.128±0.023	0.168±0.034	0.143±0.033	0.187±0.030	0.203±0.030
NP	0.186±0.025	0.247±0.051	0.218±0.038	1.273±0.059	0.267±0.035

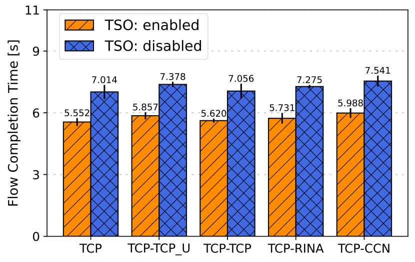


Fig. 6: Flow Completion Time of a 4 GB transfer from the server to the client. Mean and standard deviation over 100 runs per scenario are shown.

mean and standard deviation over 100 runs. With TSO, the FCT is obviously lower due to reduced CPU overhead and increased throughput from segmentation offload. We do not see any throughput degradation when PEP-DNA is used to translate from TCP to TCP; the FCT in this experiment is only affected by the minimal delay overhead of our proxy (in the range of a few tens of milliseconds), which, because of context switching avoidance of the kernel implementation, is smaller (0.2-0.3 s) than the overhead of the user-space PEP.

V. Conclusions

This paper has presented the design and implementation of PEP-DNA, a TCP connection-splitting proxy that is fast and lightweight, and can be used to interconnect TCP not only with another TCP connection, but also with an entirely different network architecture. From the numerical results obtained through the experiments conducted in our real testbed, we conclude that the gateway/proxy is able to efficiently interconnect TCP/IP applications with a RINA or an ICN network and vice versa. PEP-DNA is lightweight and scalable—it has low latency overhead and stores only 172 bytes per connection.

Since RINA and ICN are completely different than the Internet, and CCN-lite as well as the RINA implementation are only research prototypes, PEP-DNA indeed offers a realistic path towards gradual deployment of a novel network architecture. This can be done by switching to the new architecture on a path segment, possibly natively using the new architecture on one end (as we have done in our experiments), or translating back, such that TCP-based applications run on both sides (as we also have successfully tested). We stress that, for realistic deployment, at least a mechanism to know that a host is reachable via the new architecture must be in place. In our experiments, we statically made RINA aware of the peers on both sides, and configured TCP HTTP clients to use PEP-DNA as HTTP proxy when translating from TCP to ICN. To dynamically know how a peer is reachable, a system like Trotsky [23] could be used; PEP-DNA complements such systems with the ability to translate instead of tunneling.

We see various future work possibilities. For example, PEP-DNA currently only works with TCP, but it could also be extended to support other protocols. Extensions to make it visible and authenticated could be considered; this would allow to use it only when this is desired, thereby avoiding the problems with proxies that need to "cheat". PEP-DNA could also be dynamically enabled or disabled, e.g. by trying to reach a host via a new architecture, but disabling proxying upon a failure. This could further facilitate gradual deployment.

References

[1] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz. Improving tcp/ip performance over wireless networks. In *MobiCom*. ACM, 1995.
[2] G. Boddapati, J. Day, I. Matta, and L. Chitkushev. Assessing the security of a clean-slate internet architecture. In *IEEE ICNP*, Oct 2012.
[3] O. Bonaventure (Ed.), M. Boucadair (Ed.), S. Gundavelli, S. Seo, and B. Hesmans. 0-RTT TCP Convert Protocol. RFC 8803, July 2020.
[4] J. Border. Google QUIC over satellite links, Jun 2020.
[5] J. Border et al. Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations. RFC 3135 (Informational), June 2001.
[6] M. Boucadair et al. 3GPP Access Traffic Steering Switching and Splitting (ATSSS) - Overview for IETF Participants. Internet-Draft draft-bonaventure-quic-atsss-overview-00, IETF, 2020. Work in Progress.
[7] C. Caini, R. Firrincieli, and D. Lacamera. PEPsal: a performance enhancing proxy designed for TCP satellite connections. In *2006 IEEE 63rd Vehicular Technology Conference*, volume 6, May 2006.
[8] K. Ciko and M. Welzl. First contact: Can switching to RINA save the internet? In *IEEE ICIN*, pages 37–42, 2019.
[9] B. Cronkite-Ratcliff, A. Bergman, S. Vargaftik, M. Ravi, N. McKeown, I. Abraham, and I. Keslassy. Virtualized congestion control. In *SIGCOMM*, New York, NY, USA, 2016. ACM.
[10] P. Davern, N. Nashid, C. J. Sreenan, and A. H. Zahran. HTTPEP: a HTTP performance enhancing proxy for satellite systems. *Int. J. Next Gener. Comput.*, 2(3), 2011.
[11] J. Day. *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall, 2008.
[12] V. Farkas, B. Héder, and S. Nováczki. A Split Connection TCP Proxy in LTE Networks. In R. Szabó and A. Vidács, editors, *18th European Conference on Information and Communications Technologies (EUNICE)*, volume LNCS-7479, Budapest, Hungary, Aug. 2012. Springer.
[13] D. A. Hayes, D. Ros, and Ö. Alay. On the importance of TCP splitting proxies for future 5g mmwave communications. In *IEEE LCN*, 2019.
[14] K. He et al. AC/DC TCP: Virtual congestion control enforcement for datacenter networks. In *SIGCOMM*, New York, USA, 2016. ACM.
[15] M. Honda et al. Is it still possible to extend TCP? In *SIGCOMM IMC*, pages 181–194, New York, NY, USA, 2011. ACM.
[16] V. Jacobson et al. Networking named content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '09, page 1–12, 2009.
[17] A. Kapoor, A. Falk, T. Faber, and Y. Pryadkin. Achieving faster access to satellite link bandwidth. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE CS and ComSoc.*, volume 4. IEEE, 2005.
[18] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *SIGCOMM*. ACM, 2002.
[19] T. Koponen et al. A data-oriented (and beyond) network architecture. In *SIGCOMM*, New York, NY, USA, 2007. ACM.
[20] A. Langley et al. The QUIC transport protocol: Design and internet-scale deployment. In *SIGCOMM*. ACM, 2017.
[21] Y. Lin, C. Ku, Y. Lai, and C. Hung. In-kernel relay for scalable one-to-many streaming. *IEEE MultiMedia*, 20(1):69–79, 2013.
[22] K. Liu and J. Y. B. Lee. On improving TCP performance over mobile data networks. *IEEE Transactions on Mobile Computing*, 2016.
[23] J. McCauley et al. Enabling a permanent revolution in internet architecture. In *SIGCOMM*. ACM, 2019.
[24] A. Mihály et al. Supporting multi-domain congestion control by a lightweight PEP. In *2017 IINTEC*, 2017.
[25] G. Papastergiou et al. De-ossifying the internet transport layer: A survey and future perspectives. *IEEE Communications Surveys Tutorials*, 2017.
[26] C. Severance. Van Jacobson: Content-centric networking. *Computer*, 46(1):11–13, 2013.
[27] T. T. Thai, D. M. L. Pacheco, E. Lochin, and F. Arnal. SatERN: a PEP-less solution for satellite communications. In *2011 IEEE ICC*.
[28] S. Vrijders et al. Prototyping the recursive internet architecture: the IRATI project approach. *IEEE Network*, 28(2), March 2014.
[29] M. Welzl, P. Teymoori, S. Gjessing, and S. Islam. Follow the model: How recursive networking can solve the internet's congestion control problems. In *IEEE ICNC*, pages 518–524, 2020.
[30] R. Zullo, A. Pescape, K. Edeline, and B. Donnet. Hic sunt proxies: Unveiling proxy phenomena in mobile networks. In *2019 Network Traffic Measurement and Analysis Conference (TMA)*, 2019.