UiO : Department of Informatics
University of Oslo

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

Research report no. 500

Kristjon Ciko, Michael Welzl, Peyman Teymoori

September 2021

ISBN: 978-82-7368-600-8

ISSN: 0806-3036

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 and low CPU and memory 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*—Future Internet, Network Architectures, 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 challenges. Many extensions of the TCP/IP architecture have been defined to address issues related to security, multi-homing, mobility and Quality of Service (QoS). The once relatively simple suite of Internet protocols is now a muddle of fixes and improvements, and the overall system has become complex and difficult to manage.

New network architectures have been proposed, each attempting to solve these issues in its own way [1]–[5]. A particularly recent effort is "New IP" [6]. One of the main goals behind New IP is to avoid ManyNets and "islands" of communications; these problems can happen in new networks such as the Internet of Things (IoT) and industrial networks [7]. As a result, New IP has offered adding variable-length addresses, semantic, and a user-defined header to the IP layer. While ideas for new architectures are plentiful, their realistic deployment is an entirely different matter—there is not much evidence of real-life usage of new network architectures. The Internet, which forms the backbone of much of today's networking infrastructure, is often found to be rigid, with infrastructure that is hard to replace, requiring careful downward-compatibility considerations for new ideas [8]–[10].

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/).

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. The authors of [11] have shown that this can be done with minimal performance penalty, using RINA [4]. McCauley et al. tackle the deployment problem in depth in [12], culminating in the design of a system called "Trotsky" which provides the necessary bootstrapping functionality. They assume that traffic will be tunneled over legacy Internet domains for downwards compatibility. However, such tunneling can become a deployment hurdle: it needs operators to ensure that "new" traffic does not cause harm to (and is not "pushed aside" by) other Internet traffic, and it needs the endpoints to be upgraded.

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 to be deployed on Linux-based servers/routers. It follows the main principles of a "split connection" Performance Enhancing Proxy (PEP), which is described in detail in RFC3135 [13] (we will also briefly introduce this design in the next section). Our proxy aims to achieve high throughput with low latency and can be installed wherever a translation from one technology to another needs to occur—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 extending it to support other new 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 [14], 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 [15]. 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, and it is very different from RINA and TCP/IP. ICN is built on the concept of naming information/content/data rather than hosts. Unlike IP-based networks, ICN endpoints share named content, which is addressable and routable, 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 in a public repository. We provide the necessary details in Appendix A.

II. Background and Related Work

PEPs [13] are on-path devices that perform operations meant to improve the performance of TCP. Although PEPs can operate at different layers such as the application, transport, or link layer, 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 real receiver, and to give the receiver the impression that it is talking to the real sender. Splitting the connection makes the Round Trip Time (RTT) of the involved connections shorter. This shorter feedback loop yields a faster increase of the transmission rate: since the congestion control algorithm is driven by acknowledgments (ACKs), the sender is able to send more packets as it gets ACKs faster.

PEPs can be good: Connection-splitting PEPs are powerful devices, as they 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 [19] 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. As stated by its authors, PEPsal is the first open-source PEP implementation, and it is compatible with the old 2.6 Linux kernel. Another proxy called HTTP PEP (HTTPEP) [20] 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 [21], 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 [22] transparently splits the connection between a client and a server in order to improve the TCP performance in mobile networks. Virtualized Congestion Control (vCC) [23] and AC/DC TCP (Administrator Control over Datacenter TCP) [24] apply PEP-like mechanisms in hypervisors to achieve better control over the congestion control behavior in multitenant datacenters. While these two solutions do not split connections in order not to violate TCP end-to-end semantics, they have a similar degree of freedom as a connection splitter because of the extremely small RTT between the Virtual Machines (VM) and the hypervisor.

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 [25]), 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 [26]. This has led to the *ossification* of the Internet's transport layer [9].

Some approaches reduce these negative side-effects by minimizing the clandestine interference with TCP. For example, one of the earliest PEPs, the "snoop protocol" [27], 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. Much more recently, in [28], Liu and Lee propose a Lightweight PEP that enables Multi-Domain Congestion Control. This solution uses specific PEP ACKs to notify the server of any packet loss, which can then trigger the usage of a different congestion control algorithm. For this reason, changes at the server side are necessary in order to process special messages sent by the PEP.

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 [29], which is based on Explicit Rate Notification (ERN) protocols. In the ERN approach, ERN routers inform the senders about the optimal send rate. SatERN uses XCP as the ERN protocol. In particular, the SatERN gateway employs XCP to compute the optimal window size, which it then sends back to the sender. While SatERN solves some shortcomings of PEPs by avoiding to split connections, its performance cannot be compared to other PEPs.

TABLE I: Overview of PEP implementations. "T" means that the evaluation was performed in a testbed, whereas "S" indicates that a simulator environment 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.

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 [30]. 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 [31]. 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 (see [32] and references therein). Additionally, 5G's Access Traffic Steering Switching and Splitting (ATSSS) service offers new opportunities for multipath communication involving proxies [33].

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 cheating, e.g. by spoofing IP addresses). This could also allow to authenticate such network nodes. The "0-RTT TCP convert protocol" [34] 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 can explicitly take TCP extensions into account and is able to exchange control information during the handshake). Different from a transparent PEP, the proxy is explicitly visible in this scenario: rather than directly connecting to a server, a client initiates a connection towards the proxy and informs it about the server's IP address and port number.

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 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 (LKLM), is compatible with recent Linux kernel releases. It operates entirely in the kernel space, directly on top of the existing TCP/IP stack, and requires 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 [35]. 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[1], (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

[1]https://netfilter.org

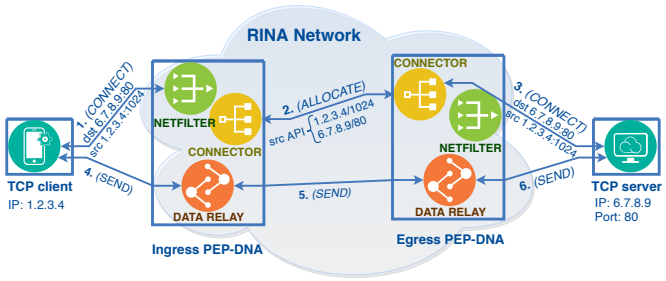


Fig. 1: Inter-domain communication between two TCP applications through proxies deployed at the RINA-IP borders.

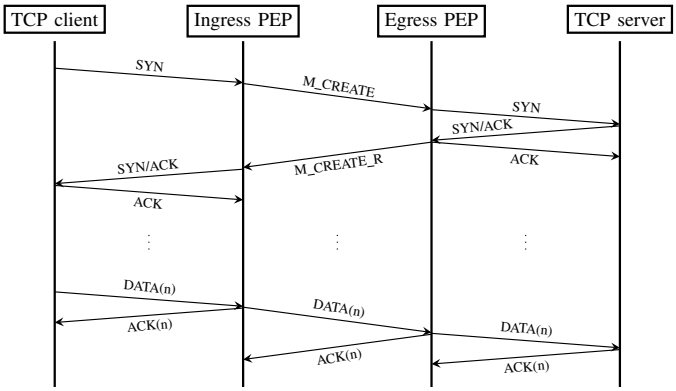


Fig. 2: Sequence diagrams showing the Connection Establishment and Data Transfer phases when two PEP-DNA proxies interconnect TCP applications over a RINA domain. This assumes that the RINA network is configured to be aware of the IP namespace (i.e., it knows how to route to the destination).

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 fast Jenkins hash table [36], the SYN packet is redirected unchanged by TPROXY iptables[1] to a local socket of the proxy. The connector immediately initiates a connection establishment procedure within the new architecture (RINA, in this case), illustrated in Step 2. For the sake of simplicity, we do not implement multiplexing of several TCP connections over a RINA flow. 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 number 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 our proxy needs to be implemented in the kernel to achieve the best performance: a pure 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 (the connection to the server can only be created after one client-PEP RTT) 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[2] 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. In reply to the interest request, 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

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

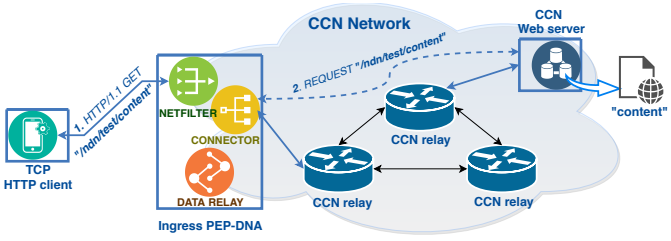


Fig. 3: Interconnection between a TCP HTTP client and a CCN network.

technologies while being transparent to the endpoint. 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 [12], 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.

In order to facilitate the connection establishment and termination phase within other architectures, the proxy leverages Netlink, which is a mechanism used to transfer information between the kernel and user-space processes. Netlink enables the proxy to communicate with processes that are responsible for setting up/terminating new connections, without needing to reimplement such tasks for every architecture. This yields a reduction of complexity and simplification of the procedures to add support for a new architecture. Also, connection establishment becomes fully dynamic and there is no need for additional configurations nor static mappings between IP and other architecture's services.

For all events such as *accept_from*, *connect_to*, *read_from*, *write_to* etc., PEP-DNA executes a callback function registered during the initalization phase. In order to add support for a new network architecture, new callback functions have to be provided and the proxy needs to load architecture-specific libraries. 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.

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 4.19.0-16-amd64. 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 [37] as a protocol stack alongside TCP/IP. We choose RINA and ICN as examples of novel network architectures to connect to.

We run the experiments for five different scenarios illustrated in Fig. 4: (i) **TCP** *(the baseline)*: two TCP applications talk to each other via a full TCP/IP path, (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

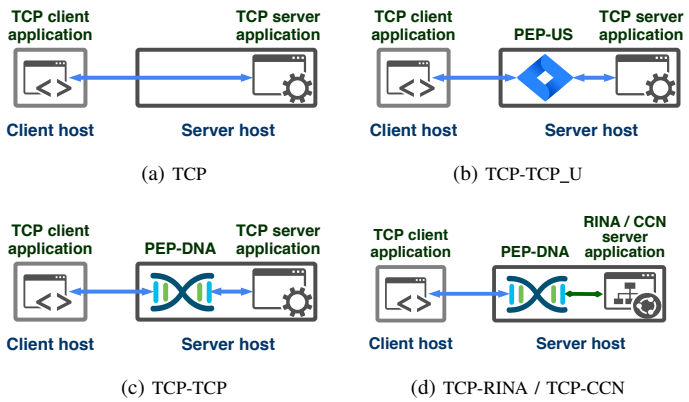


Fig. 4: Different scenarios used to evaluate the performance of PEP-DNA in comparison with TCP baseline and a generic user-space PEP.

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 operate in user space and split connections. Of those, only PEPsal is openly available, but its code is older, and it incurs more overhead than our minimalistic user-space proxy (PEP-US).

Through the conducted experiments we want to evaluate the performance of PEP-DNA and test how fast it can process packets and forward data from one technology to another. For this reason, we operate the experiments under high data rates and deploy PEP-DNA at the same host as the server to allow the applications to send and receive data as fast as possible. In particular, we refrain from long-distance communication with RINA and CCN because we do not want to evaluate the architecture per se.[3] 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 simple httping-like[4] application which supports both TCP sockets and the RINA API and can run as a Web client or server.

Due to the inaccuracy of queuing disciplines for high bandwidth-delay product network conditions, we do not apply any traffic shaping to limit the rate or inject any delay in the network. Instead, we use *ethtool* to set the ethernet cards' speed to 10 Gbps and enable/disable TCP Segmentation Offloading (TSO). In each experiment, the testing applications are pinned to a dedicated CPU core and CPU frequency scaling is disabled, thus changing the CPU from *powersave* mode to *performance* mode.

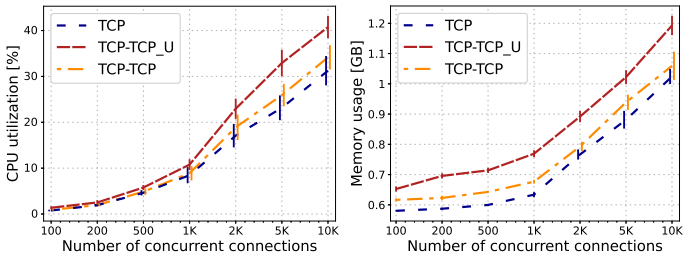
[3]We tested the scenario in Fig. 1 as well, and found a significant performance improvement when the RINA segment was the bottleneck. This happened because the only correctly working "congestion control" implementation in the IRATI prototype simply transmits data with a fixed (large) window size. While 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.

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

A. Results and Analysis

We mimic the experiment of [24] to measure the impact of PEP-DNA on the overall memory usage and CPU utilization of the host where it is deployed. The client initiates multiple simultaneous TCP connections to the sever. After all TCP connections are established, the client sends a burst of 128 KB every 100 milliseconds. We measure the overall CPU utilization and memory usage at the server side by sampling the kernel statistics from */proc/meminfo* and */proc/stat* every second. The number of concurrent connections varies from 100 to 10000. Since the IRATI implementation of RINA is just a prototype which needs to be further optimized, and CCN-lite is developed as a code base only for experimental purposes, we do not include RINA and CCN in this intensive experiment, but run it only for the TCP scenario (without PEP-DNA) and the TCP-TCP scenario (with PEP-DNA), illustrated in Figs. 4a and 4c.

Fig. 5 shows the CPU utilization and memory usage comparison between two different scenarios, with and without proxy being used (TCP and TCP-TCP respectively). Graphs show the medians, which mark the 50th percentile, and error bars, which represent 10th to 90th percentiles. From Fig. 5a, we see that the CPU utilization values of both scenarios begin to differ when the number of concurrent connections is above 1000. For a higher number of concurrent connections, there is a low CPU overhead of less than 3% in the TCP-TCP case, with the largest difference being around 2.9% for 10 K concurrent flows. Unlike CPU utilization, the memory usage overhead stays fairly constant (between 25 MB - 55 MB) for all the measurements, as illustrated in Fig. 5b.



(a) CPU utilization (b) Memory usage

Fig. 5: Sampled CPU utilization and memory usage measured at the server for TCP (without proxy), TCP-TCP_U (with user-space PEP), and TCP-TCP (with PEP-DNA).

For the next experiments, 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.

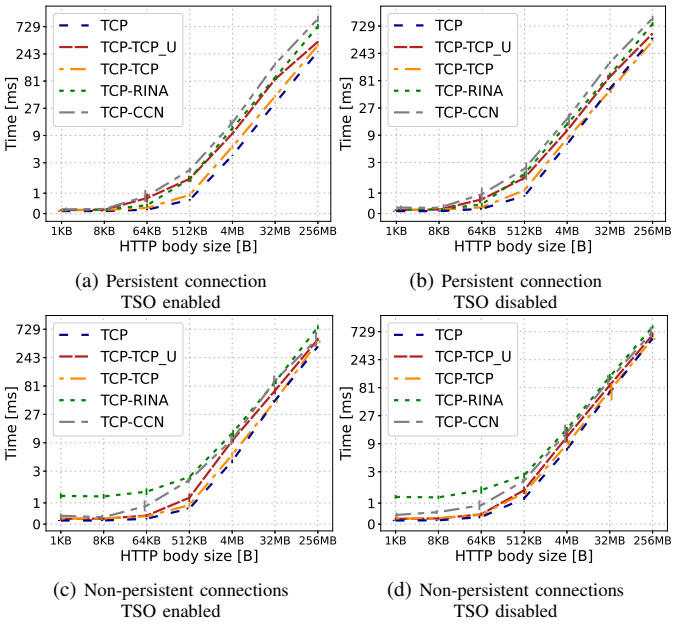
Table II shows the mean and standard deviation of application-layer latency measurements for the five scenarios described earlier, using HTTP with both persistent and non-persistent connections. In case of non-persistent connections, each HTTP request and response is sent over a new connection and the 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.

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

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. 6 shows the results of this experiment when persistent and non-persistent connections are used both with 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. When connections are not persistent, the delay is larger in RINA because of the flow creation, which takes a significant amount of time. The impact of TSO 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)



(a) Persistent connection TSO enabled

(b) Persistent connection TSO disabled

(c) Non-persistent connections TSO enabled

(d) Non-persistent connections TSO disabled

Fig. 6: 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.

of a 4 GB file 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 both the TSO enabled and disabled cases are shown in Fig. 7. Bars and error bars represent respectively the mean and standard deviation over 100 runs. When TSO is enabled, the FCT is obviously lower, as a result of reduced CPU overhead and increased throughput due to the segmentation offload performed by the Network Interface Controller (NIC). 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.

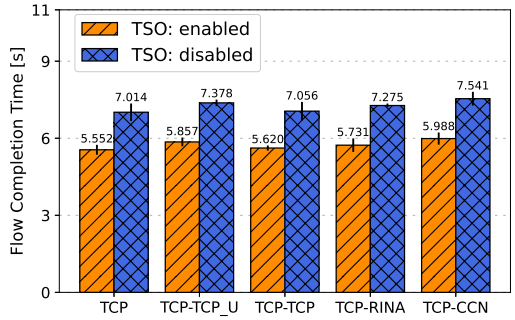


Fig. 7: Flow Completion Time of a 4 GB file transfer from the server to the client. The bars show the mean and standard deviation values over 100 runs for each scenario.

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 a lightweight proxy—it stores only 172 bytes per connections and generally exhibits low CPU and memory usage. Hence, PEP-DNA is scalable and can handle many concurrent connections.

Considering that RINA and ICN are completely different than the Internet, and CCN-lite as well as the RINA implementation are only research prototypes, this gives us reason to believe that PEP-DNA indeed offers a very realistic path towards gradual deployment of a novel network architecture. This can be done by switching to the new architecture on a segment of the path, possibly using a native application of that new architecture on one end (as we have done in the experiments documented in section IV), or translating back, such that native TCP applications run on both sides (as we also have successfully tested). In this way, new architectures could be deployed quite easily and quickly, without the need to develop new applications or change the current ones. We stress that, for realistic deployment, at least one more element needs to be in place: a mechanism to know that a host is reachable via the new architecture. 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 [12] could be used; PEP-DNA does not replace, but complement such systems with the ability to translate instead of tunneling.

We see various possibilities for future work. For example, PEP-DNA currently only works with TCP, but it could be extended to support other Internet protocols just as well. Also, extensions to make it visible and authenticated (rather than transparent) could be considered; this would allow to use it only when this is desired, thereby avoiding the ossification problem due to proxies that need to "cheat". Mechanisms to dynamically enable or disable the proxying mechanism could also be embedded in PEP-DNA itself, e.g. by trying to reach a host via a new architecture, but disabling proxying in case the new architecture is not available. This could further facilitate the new architecture's gradual deployment.

References

[1] A. Afanasyev, J. Burke, T. Refaei, L. Wang, B. Zhang, and L. Zhang, "A brief introduction to named data networking," in *IEEE MILCOM*, 2018, pp. 1–6.

[2] C. Severance, "Van Jacobson: Content-centric networking," *Computer*, vol. 46, no. 1, pp. 11–13, 2013.

[3] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *SIGCOMM*. New York, NY, USA: ACM, 2007.

[4] J. Day, *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall, 2008.

[5] J. Pan, S. Paul, and R. Jain, "A survey of the research on future internet architectures," *IEEE Communications Magazine*, vol. 49, no. 7, 2011.

[6] Z. Chen, C. Wang, G. Li, Z. Lou, S. Jiang, and A. Galis, "New IP framework and protocol for future applications," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, 2020.

[7] H. Sharp and O. Kolkman, "Discussion paper: An analysis of the "new ip" proposal to the ITU-T," 2020, accessed: 2021-06-30. [Online]. Available: https://www.internetsociety.org/resources/doc/2020/discussion-paper-an-analysis-of-the-new-ip-proposal-to-the-itu-t/

[8] M. Handley, "Why the internet only just works," *BT Technology Journal*, vol. 24, no. 3, pp. 119–129, Jul. 2006. [Online]. Available: https://doi.org/10.1007/s10550-006-0084-z

[9] G. Papastergiou, G. Fairhurst, D. Ros, A. Brunstrom, K. J. Grinnemo, P. Hurtig, N. Khademi, M. Tuexen, M. Welzl, D. Damjanovic, and S. Mangiante, "De-ossifying the internet transport layer: A survey and future perspectives," *IEEE Communications Surveys Tutorials*, vol. 19, no. 1, pp. 619–639, Firstquarter 2017.

[10] H. Balakrishnan, S. Banerjee, I. Cidon, D. Culler, D. Estrin, E. Katz-Bassett, A. Krishnamurthy, M. McCauley, N. McKeown, A. Panda, S. Ratnasamy, J. Rexford, M. Schapira, S. Shenker, I. Stoica, D. Tennenhouse, A. Vahdat, and E. Zegura, "Revitalizing the public internet by making it extensible," *SIGCOMM Comput. Commun. Rev.*, vol. 51, no. 2, p. 18–24, May 2021. [Online]. Available: https://doi.org/10.1145/3464994.3464998

[11] K. Ciko and M. Welzl, "First contact: Can switching to RINA save the internet?" in *IEEE ICIN*, 2019, pp. 37–42.

[12] J. McCauley, Y. Harchol, A. Panda, B. Raghavan, and S. Shenker, "Enabling a permanent revolution in internet architecture," in *SIGCOMM*. New York, NY, USA: ACM, 2019, pp. 1–14.

[13] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," RFC 3135 (Informational), RFC Editor, Fremont, CA, USA, Jun. 2001. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3135.txt

[14] G. Boddapati, J. Day, I. Matta, and L. Chitkushev, "Assessing the security of a clean-slate internet architecture," in *IEEE ICNP*, Oct 2012.

[15] 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*, 2020, pp. 518–524.

[16] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 1–12. [Online]. Available: https://doi.org/10.1145/1658939.1658941

[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.*, vol. 4. IEEE, 2005.

[18] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *SIGCOMM*. ACM, 2002.

[19] C. Caini, R. Firrincieli, and D. Lacamera, "PEPsal: a performance enhancing proxy designed for TCP satellite connections," in *2006 IEEE 63rd Vehicular Technology Conference*, vol. 6, May 2006.

[20] 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.*, vol. 2, no. 3, 2011.

[21] V. Farkas, B. Héder, and S. Nováczki, "A Split Connection TCP Proxy in LTE Networks," in *18th European Conference on Information and Communications Technologies (EUNICE)*, R. Szabó and A. Vidács, Eds., vol. LNCS-7479. Budapest, Hungary: Springer, Aug. 2012.

[22] A. Mihály, S. Nádas, S. Molnár, Z. Krämer, R. Skog, and M. Ihlar, "Supporting multi-domain congestion control by a lightweight PEP," in *2017 International Conference on Internet of Things, Embedded Systems and Communications (IINTEC)*, 2017.

[23] 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: ACM, 2016.

[24] K. He, E. Rozner, K. Agarwal, Y. J. Gu, W. Felter, J. Carter, and A. Akella, "AC/DC TCP: Virtual congestion control enforcement for datacenter networks," in *SIGCOMM*. NY, USA: ACM, 2016.

[25] 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.

[26] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?" in *SIGCOMM IMC*. New York, NY, USA: ACM, 2011, pp. 181–194.

[27] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving tcp/ip performance over wireless networks," in *MobiCom*. New York, NY, USA: ACM, 1995, pp. 2–11. [Online]. Available: https://doi.org/10.1145/215530.215544

[28] K. Liu and J. Y. B. Lee, "On improving TCP performance over mobile data networks," *IEEE TMC*, vol. 15, no. 10, pp. 2522–2536, 2016.

[29] T. T. Thai, D. M. L. Pacheco, E. Lochin, and F. Arnal, "SatERN: a PEP-less solution for satellite communications," in *2011 IEEE International Conference on Communications (ICC)*. IEEE, 2011, pp. 1–5.

[30] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The QUIC transport protocol: Design and internet-scale deployment," in *SIGCOMM*. ACM, 2017.

[31] J. Border, "Google QUIC over satellite links," Jun 2020. [Online]. Available: https://datatracker.ietf.org/meeting/interim-2020-panrg-01/materials/slides-interim-2020-panrg-01-sessa-google-quic-over-satellite-testing-update

[32] D. A. Hayes, D. Ros, and Ö. Alay, "On the importance of TCP splitting proxies for future 5g mmwave communications," in *IEEE LCN*, 2019.

[33] M. Boucadair, O. Bonaventure, M. Piraux, Q. D. Coninck, S. Dawkins, M. Kühlewind, M. Amend, A. Kassler, Q. An, N. Keukeleire, and S. Seo, "3GPP Access Traffic Steering Switching and Splitting (ATSSS) - Overview for IETF Participants," IETF, Internet-Draft draft-bonaventure-quic-atsss-overview-00, May 2020, work in Progress.

[34] O. Bonaventure (Ed.), M. Boucadair (Ed.), S. Gundavelli, S. Seo, and B. Hesmans, "0-RTT TCP Convert Protocol," RFC 8803 (Experimental), RFC Editor, Fremont, CA, USA, Jul. 2020.

[35] Y. Lin, C. Ku, Y. Lai, and C. Hung, "In-kernel relay for scalable one-to-many streaming," *IEEE MultiMedia*, vol. 20, no. 1, pp. 69–79, 2013.

[36] B. Jenkins, "A hash function for hash table lookup (2006)," *URL www. burtleburtle. net/bob/hash/doobs. html*, 2015.

[37] S. Vrijders, D. Staessens, D. Colle, F. Salvestrini, E. Grasa, M. Tarzan, and L. Bergesio, "Prototyping the recursive internet architecture: the IRATI project approach," *IEEE Network*, vol. 28, no. 2, March 2014.

APPENDIX

Since few PEP implementations are open source, it is not easily possible for the research community to evaluate the performance of PEPs and compare their behavior with other solutions. We aim to make our work entirely reproducible and encourage interested researchers to test the code and replicate the reported experimental results. The PEP-DNA implementation and documentation needed to reproduce all the experiments described in this paper are available in a public repository.[5] This includes the tools that we developed to the run the experiments, as well as the scripts for analyzing and plotting of the generated data. In addition, we provide information on how we set up third party tools that we used for our experiments.

Below, we provide additional information on how to achieve *repeatability*, *replicability*, and *reproducibility*.

Repeatability: we repeated each experiment 100 times, and the standard deviations are small (see Section IV for details).

Replicability: a different research team will be able to obtain the same results when repeating the experiments with the same artifacts under the same conditions.

Reproducibility: when trying to reproduce the results using a different experimental setup, we expect that similar behavioral trends will show, as we also evaluated PEP-DNA in a smaller testbed, with 1 Gbps links, and also with VMware virtual machines, and saw similar trends. The absolute values of CPU and memory overhead may be system dependent. This could have an effect on the results in Figure 5, but it should only have a minimal impact on all other results.

[5]https://github.com/kr1stj0n/pep-dna.git