First Contact: Can Switching to RINA save the Internet?

Kristjon Ciko, Michael Welzl
University of Oslo, Norway
kristjoc@ifi.uio.no, michawe@ifi.uio.no

***Abstract*—RINA is a new network architecture that has been found to have many benefits over the Internet in prior works. In this paper, we investigate the possibility of directly switching from the Internet to RINA "under the hood", such that the user or application does not notice what happens. This does not involve tunneling, gateways, etc., but is meant to switch from *native* Internet to *native* RINA. We explain the theoretical number of messages and round-trips involved in such direct switch-over and evaluate its efficiency in a local testbed.**

I. Introduction

The Recursive InterNetworking Architecture (RINA) is a novel "back to basics" type approach to networking. RINA, described in detail in [6], is designed to be better than the current Internet architecture in many ways: it should be easier to manage, be more secure, more flexible and perform better.

Because it is drastically different from the Internet, RINA is not downwards-compatible: it cannot directly communicate with standard Internet hosts. This is certainly a major problem when we imagine transitioning a long and diverse Internet path to a new technology—e.g., how could TCP/IP communication from Australia to Europe be made to use RINA all of a sudden? However, in today's networked world, much communication uses short paths as a way to minimize latency and improve scalability. Content Distribution Networks (CDN) have become ubiquitous—popular services such as Google search, YouTube, Facebook, Twitter and Netflix are often only a handful of hops away, directly provided via a customer's Internet Service Provider (ISP) or one of its neighboring ISPs.

In this paper, we ask: assuming that the right software would be installed in the client, server and intermediate nodes, could it be possible to efficiently carry out a direct switch-over from TCP/IP to RINA? We note that similar things are already happening: for IPv6 or QUIC [10], end hosts make attempts to use the new technology and gracefully fall back to the legacy system (IPv4 or TCP, respectively) in case of a failure. While switching to RINA is arguably more drastic than switching from IPv4 to IPv6 or from TCP to QUIC, conceptually there is no big difference—and we need to ask the same key question as in all of these cases: can it be done *fast*?

This paper provides a first full explanation of how switching over to RINA could work in a simple but realistic one-ISP scenario. As we set this in contrast with the Internet, we are able to elaborate on the differences in terms of overhead at a server (which could translate into scalability limitations), and, using the IRATI RINA prototype [15], evaluate the latency of initial communication via either the Internet or RINA.

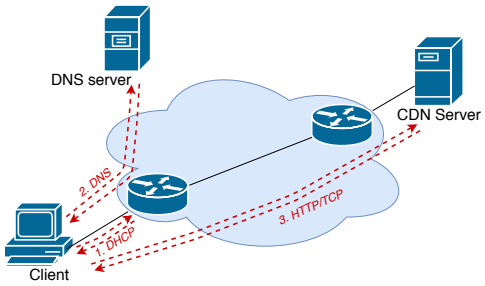

Fig. 1: One-ISP scenario: Internet case

In the next two sections, we will introduce the necessary background. In these sections, and in the whole paper, We divide our considerations into two parts: "first contact"—the very beginning, when no preparations were made by the end hosts to enable communication—and "'second contact"', when a host has already found the peer and managed to talk to it, and local host caches were filled with information. Note that we assume a significant amount of time (say, an hour) between first and second contact—e.g., typically, in the Internet, HTTP would not re-use the same TCP connection after such a long time. Section IV compares the theoretical number of messages transmitted when establishing communication in the two cases. Then, we look at real-life measurements and quantify the potential efficiency of RINA vs. the Internet. We discuss related work in Section VI, and Section VII concludes.

II. "First contact" (and "second contact") ...in the Internet

Let us consider how the initial access to a webpage works in a simple scenario of only one ISP, hosting a DNS server and a CDN server that provides data to the ISP clients. The topology for this scenario is shown in Fig. 1. For simplicity, we only consider an HTTP request over TCP/IPv4 without TLS and no use of other protocols such as QUIC. While some of the following text may appear to be at undergraduate textbook-level simplicity, we feel it is necessary in order to clarify how RINA differs from the Internet when communication begins.

A. First contact

In order to send data on the Internet, a host must have an IP address and other information such as gateway and DNS server address, which can be configured manually or obtained through DHCP. We consider the latter.

The client and the lower router in Fig. 1, which acts as a DHCP Server, exchange 4 packets. Then, the client has obtained an IP address and it can access the Internet. To reach a specific website, it needs to resolve the requested URL to an IP address using DNS. Since the client knows from the static well-known structure of IP addresses that it is not in the same subnet as the DNS server[1], it first asks for the MAC address of the default gateway by broadcasting an ARP packet. When the MAC address is learnt from the ARP reply, the DNS query can be sent to the router in order to be delivered to the appropriate host. The DNS server replies back specifying the IP address associated with the hostname that the client is trying to reach (this assumes that the DNS server is able to locally resolve the IP address; otherwise, more messages are needed).

Next, the client attempts to establish a TCP connection to the well-known port number 80 at the resolved IP address using a three-way handshake: the client sends a SYN segment with an initial sequence number. The server responds with a SYN-ACK to acknowledge the client's SYN and send its own initial sequence number. Finally, the client sends an ACK back to the server. The client can now issue an HTTP GET request, which is answered with data by the server.

TCP's connection establishment synchronizes initial sequence numbers, protecting both sides against mistakingly accepting old messages, e.g. after a host has crashed. It also provides some form of protection against overload from Denial-of-Service attacks, as SYN requests carrying spoofed source IP addresses only provoke SYN-ACKs which cannot be answered, but do not yet involve the server application. The method has the obvious disadvantage of delaying communication by one Round-Trip Time (RTT). Another disadvantage is that, just like any other packets, the initial SYN and SYN-ACK packets may be dropped. Considering the large default retransmission timeout of typically 1-3 seconds, losing these packets can have a significant impact on user-experience, especially for web traffic, which often consists of multiple short TCP transfers [5].

B. Second contact

TCP connections are somewhat costly, as they need buffer space and timers. Given that we assume a delay in the order of an hour in between first and second contact, it is unlikely that a TCP connection would be kept open for so long. Usually, web servers minimize their resource usage by directly closing the connection after answering a GET request (i.e., beginning a FIN-FIN/ACK 4-way handshake).

TCP Fast Open (TFO) [14] is a mechanism that strives to allow fast web communication by removing the extra round-trip. TFO enables data to be carried in the SYN and SYN-ACK segments, saving up to one full RTT for any but the very first connection. For security purposes, with TFO, the server has to send a cookie as a TCP Option to the client during the first three-way handshake. Using this cookie in later connections allows the client and server to send data immediately without waiting for a handshake to be completed. We carried out a test with our nearest Google server and found that this server kept the cookie for several days.

Despite its obvious benefits, TFO is experiencing slow deployment. According to results in [12], a considerable number of packets with the experimental TFO TCP option did not reach the server. Some middleboxes, such as firewalls or NAT boxes may cause issues with the new experimental option:

- Old devices drop packets with unknown options, causing a retransmission without TFO, thus increasing the delay.
- The TFO option can simply be removed.
- Some middleboxes drop SYN segments with data.
- The SYN/ACK packet can be dropped because it acknowledges SYN+data, more than a middlebox was expecting, hence preventing connection establishment. Similarly, Intrusion Detection Systems can block attempts of a server to send data before the handshake is completed. Even later non-TFO connections may then be blocked.

Moreover, TFO does not deal with duplicate data that might be received due to the possible retransmission of SYN packets—only applications that are tolerant of duplicate arrival of the first message should use TFO. This is a deviation from the otherwise byte-stream-based communication that TCP offers: to use TFO, a "sendto(...)" call (normally only used with datagram sockets) must be used with TCP's SOCK_STREAM.

III. "First contact" (and "second contact") ...in RINA

A. RINA and its terminology in a nutshell

RINA is based on the fundamental principle that Networking is **Inter-Process Communication (IPC)** [7]. A **Distributed IPC Facility** (DIF) is the main component of this recursive architecture; it is a set of **IPC processes (IPCPs)** and serves as a single type of layer that repeats itself as many times as necessary, over different scopes. Each IPCP is a container that provides many functionalities like routing, transport, security and management. IPCPs can join a DIF through the process of **Enrollment** by contacting pre-existing IPCP members and being authenticated by them.

RINA layers are recursive and stacked upon each other, such that an N-DIF uses the service provided by the (N-1)-DIF, offering a service to the (N+1)-DIF or an application. All layers have the same set of mechanisms but use their own configured policies. The lowest level DIF is called **Shim-DIF** and allows RINA to operate on top of a physical medium, e.g. Ethernet or Wireless. Each DIF provides communication flows as a service to upper layers and/or applications. A **Flow Allocator (FA)** creates a flow along with a per-flow **Flow Allocator Instance (FAI)** that is responsible for handling the flow during its lifetime. The FA is helped by a **Resource Allocator (RA)**, which monitors resources within the DIF and shares information between IPCPs. All objects that define IPCP states are stored in a **Resource Information Base (RIB)**, accessed through a **RIB Daemon** and exchanged by the **Common Distributed Application Protocol (CDAP)**.

[1]We stress the "static well-known structure" because this kind of hard-coding does not exist in RINA. This renders RINA much more flexible, albeit at the cost of signaling overhead.

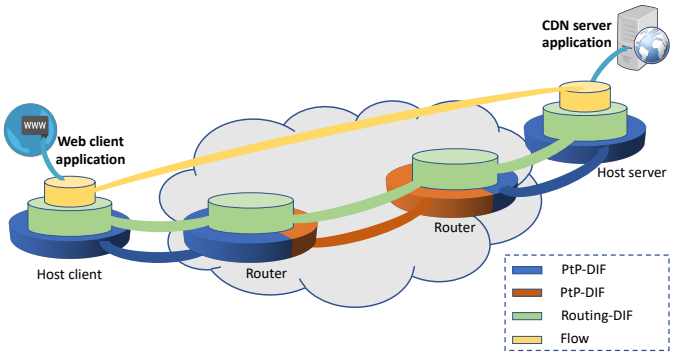

Fig. 2: One-ISP scenario: RINA case

All modules of an IPCP can be classified in three main categories: IPC API, Data Transfer and IPC Management. The **IPC Resource Manager (IRM)** acts as an orchestrator that manages DIFs and their IPCPs inside a processing system. It is responsible for handling flow requests and forwarding to the underlying IPCPs. A flow request contains a string variable that holds the **Application Process Name (APN)** of the target application. The information about APNs is shared across distributed directories which are maintained by a distributed application named **DIF Allocator (DA)**. DA has more than a DNS-like function. Besides resolving application names, it is also responsible for creating supporting DIFs between applications when they are needed.

B. "First contact" in RINA

When a web client wants to connect to a web server, it prepares an *AllocateRequest* specifying the destination APN and QoS requirements in a method called "createFlow(src-name, dst-name, QoS, ...)". The IRM of the client host receives this *AllocateRequest*, which it passes to all locally available DIFs in the system. Because the requested application name is not yet locally known, the IRM will get a negative result and it can directly or via any DIF ask the DA to determine which DIF to use. Different from the Internet, DNS is not required in RINA as the application discovery mechanism is based on the DA. This communication with the DA is the first non-local operation.

In the RINA scenario described in this section, the client host is already enrolled in the PtP-DIF (blue) and Routing-DIF (green) in Fig. 2. IPCPs exchange several CDAP Messages in handshakes (lasting several RTTs) as part of the enrollment process; this includes authentication, updating of RIB objects, and connecting / starting signals. Using the Routing-DIF, it contacts the DA of the nearest router. Upon registration of the web server application, an entry mapping the APN to the DIF name is added to the DA directory and shared with other DAs. The DA queries its RIB database to find in which DIF the web server application resides. If it does not find an entry for the requested application, it forwards the query to next peer DA.

Once the IRM knows about an available N-DIF (the green Routing-DIF in Fig. 2) and a supporting (N-1)-DIF, it checks if any IPCP can support this communication. The client's system is already a member of the (green) N-DIF, hence the IRM will delegate the request to the appropriate IPC's FA. After some local procedures, the FA Instance needs to forward a request to the IPCP at the server host via the (green) Routing-DIF. It instantiates an instance of RINA's data transfer protocol, the **Error and Flow Control Protocol (EFCP)** for this new flow and sends an A-Data M_CREATE CDAP Message. This message, and any other management messages, are transmitted over a Management Flow that is allocated for this purpose in the Routing-DIF.

The purpose of this CDAP Message is to create a RIB Entry at the other side with the key: IPCP Address, PortID. This establishes a new Flow in the Routing-DIF between the IPCPs by which the applications will communicate. At this time, the server-side IPCP that is already a member of the Routing-DIF is asked if it can allow the flow. If so, EFCP Instances and necessary bindings are created. The responsible server-side FAI prepares and sends back a M_CREATE_R CDAP Message with necessary information to enable further communication in the N-DIF. The client host then updates its RIB Database regarding Flows. At this moment, the flow in the Routing DIF is allocated and the applications can transfer data through a reliable or unreliable connection, which is created by EFCP. EFCP is built on the time-based "Delta-t" proposed by Watson in [16]. According to the operation of this protocol, a pair of EFCP instances are created for every connection, one instance for each endpoint. Watson stated that in order to ensure reliable connections, three timers need to be bounded: a) **Maximum Packet Lifetime (MPL)**, the maximum time a packet can exist within the network; b) $\mathbf{R_{timer}}$, the maximum time a sender will attempt the retransmission; c) $\mathbf{A_{timer}}$, the maximum time a receiver will hold the acknowledgment before sending it. In other words, timers are sufficient to maintain a synchronising state, and explicit messages like SYN, SYN-ACK or FIN are not needed. The state is installed at the sender and receiver by initiating timers when they send or receive a packet. Sending or receiving a packet also refreshes the timers, and state is discarded when there is no traffic for 2-3 MPL.

C. "Second contact" in RINA

Even hours after "first contact", it is probably safe to assume that the client's system is already a member of the resolved DIF (just like DNS information can be cached and stored for hours, information about DIF enrollment can be stored by the client and server's OSes). The FAI will maintain the state of the flow during its lifetime and the applications will continue to use the current flow as long as none of them makes a *deallocateRequest()* call. RINA's flows are a concept that is hard to map to anything in the Internet: they represent state that is kept as long as applications want to communicate, but they do not include TCP's overhead that is related to the operation of the protocol itself (e.g., RTT-timescale timers). Considering the "second contact", it may or may not be necessary for applications to reallocate a flow in RINA: depending on resource availability, an application could decide to keep a flow open just like it could decide to keep a TCP connection open, with the former being less

costly than the latter. When a flow is available, an application can simply send data via an EFCP connection as explained above: new state will automatically be created and timers will be refreshed.

IV. Comparison Internet vs RINA

Table I provides an overview of Internet vs. RINA "first" and "second" contact communication, respectively. For "first contact", with RINA, it is not yet necessary to know the address that the application will ultimately want to talk to. A web browser could therefore immediately contact any closely located "web IPCP" in order to join the right DIF upon startup, and stay enrolled in this DIF until the application is closed (or even longer, if the IPCP in charge of the DIF is implemented in the OS). Later, when the user enters a URL, talking to a new server application that is already registered in the DIF just requires the M_CREATE – M_CREATE_R handshake to create a flow.[2] In the Internet, all "first contact" handshakes after DHCP must be carried out every time a new (and not cached) URL is entered by the user.

The efficiency of "second contact" depends, upon other things, on whether a RINA flow is kept or removed since "first contact". Similar to the arrival of a TCP SYN at the server side, various resources are reserved when a flow creation request (M_CREATE) arrives: memory needs to be allocated, an EFCP instance is created, etc. Different from a simple TCP SYN (as opposed to, say, a TLS handshake), RINA already checks access control with the application at this early point. Also different from a SYN arrival, the state associated with a flow does not include the necessary elements for reliable or unreliable communication (timers at RTT timescales, buffers to correct re-ordering, etc.). Thus, it requires less effort to maintain a RINA flow than to keep a TCP connection open. If immediate communication is found to be necessary with busy servers that may not even be able to keep RINA flow state for a long time, the RINA specification could be changed to allow the M_CREATE message to also carry data; in this case however, reception of M_CREATE would also mean that EFCP state would immediately have to be instantiated. Also, because M_CREATE messages are transmitted over a reliable Management Flow in the (N-1)-DIF, this data transmission would necessarily be reliable as well.

While a RINA flow is concerned with the reservation of resources, a RINA connection is the shared state between endpoints inside a DIF that contains all the necessary elements to make EFCP work. From the way a connection is established and then removed, EFCP is considered a pure soft-state protocol, unlike TCP which binds timers but uses explicit signals as well. As mentioned in Section III, RINA's transport protocol is based on Delta-t which, instead of requiring an explicit connection setup handshake, binds three timers in order to satisfy the following conditions [8]:

- If no connection exists, no old packets from a previously closed connection should be accepted. This condition protects against falsely opening a connection.
- If a connection exists, no duplicate packets from a current or previously closed connection should be accepted.

Gursun et al. show in [9] that Delta-t is more robust than TCP and in [2] it is proved that RINA is more resistant than TCP against security attacks. Using the mechanisms described above and without a cookie nor any extensions needed, RINA has all the potential to combine the benefits of TCP Fast Open and Transactional TCP(T/TCP) [3] that also caches the connection states to handle duplicates. In RINA, none of the disadvantages of TFO exist:

- The first message from the client is not limited to one packet (although the number of initial packets will have to be limited in some way by congestion control, which is still work in progress).
- The first message can be guaranteed to not arrive as a duplicate, as it is just a normal message containing data in the framework provided by Delta-t.
- As with any newly defined "from-scratch" technology, since there is no need of downwards compatibility to fit a large installed code base, there is also no risk of bad interaction with middle-boxes. In other words, RINA's initial communication in "second contact" always works.

V. Evaluation

This section presents the measurement results obtained through the experiments conducted in our testbed. The testbed consists of several physical machines, each configured with two Intel E5-2620 processors and 64 GB of physical memory. These nodes are connected to each other through 10 Gigabit Ethernet links as in Figs. 1 and 2. Each machine runs a Debian 9 OS where the IRATI implementation of RINA is deployed as a protocol stack alongside TCP/IP. Experiments were conducted for three different scenarios: TCP without Fast Open, TCP Fast Open (TFO) and RINA. We developed simple HTTP clients and servers in order to measure the exact time it takes to complete the operations described in Table I.

All network operations are classified into three main phases: Preparation, Establishment and Transfer, covering the three line blocks in Table I. For "first contact", in TCP with and without TFO, the Preparation phase includes the time needed for the client to complete the DHCP and DNS procedures. In RINA, the Preparation phase is composed of the startup enrollment in the routing-DIF, which enables the client to connect to ISP routers and application discovery via the DIF Allocator. In case of TCP, Establishment consists of the 3-way TCP handshake. In RINA, it comprises the end-to-end flow allocation between the client and the server. Both with TCP and with RINA, the Transfer phase is measured from the moment when the client sends the first GET request until it receives the ACK + Data.

[2]This assumes a policy where all IPCPs dynamically learn about all registered addresses in the DIF, which has scalability limits but may be appropriate for small DIFs. Other policies could involve a DNS-like system, with more handshakes. In the Internet, DNS is always the same, no matter how far away a server is. This kind of flexibility is a benefit of RINA.

INTERNET				RINA	
TCP		TCP Fast Open			
1st	2nd	1st	2nd	1st	2nd
DHCP ⇄ DNS ⇄		DHCP ⇄ DNS ⇄		Startup DIF Configuration Enrollment ⇄ DIF Allocator ⇄	
SYN → ← SYN/ACK	SYN → ← SYN/ACK	SYN + CookieRequest → ← SYN/ACK + Cookie		M_CREATE → ← M_CREATE_R	
ACK + GET → ← ACK + Data	ACK + GET → ← ACK + Data	ACK + GET → ← ACK + Data	SYN + GET + Cookie → ← SYN/ACK + Data	GET → ← ACK + Data	GET → ← ACK + Data

TABLE I: Internet vs RINA, from "first contact" until the first data is received by a web client. Double arrows indicate communication that can involve other systems and may require several handshakes. In all cases, "second contact" is assumed to start much later than the first.

The web applications include methods to measure the time of each phase. The results obtained are also confirmed by Wireshark captures. We use netem[3] to run the experiments with different RTTs by setting a delay in each interface of the nodes. Using delay values of 2 ms and 20 ms, we obtain minimum RTTs of 12 ms and 120 ms, respectively. By transmitting 100 ping packets with a normal frequency of 1 packet/sec, we find that the real average RTT is 12.52 ms in the first case and 120.52 ms in the latter, with standard deviations of 0.11 ms and 0.405 ms, respectively. For RINA, we used the tool "rina-echo-time", obtaining the values 12.57 ms and 120.57 ms with standard deviations of 0.007 ms and 0.012 ms.

Figs. 3 and 4 show the results for "first contact". We see that the Preparation phase takes longer than the other two phases. The DHCP process takes around 90% of the Preparation time in TCP (with or without Fast Open) and the rest consists of DNS resolution. RINA Preparation is particularly fast because enrollment in the Routing-DIF (green in Fig. 2)—the only non-local operation in this phase—only requires handshakes between the client and the first-hop router. Also note that, as mentioned before, RINA Preparation can already be performed when the browser is started, whereas the browser has to wait until a user enters a URL in the Internet case.

As expected from Table I, there is no significant difference between TCP and RINA in the Establishment and Transfer phases. Unsurprisingly, RINA is slightly slower because the IRATI implementation of RINA is just a prototype, whereas the OS TCP code has been highly optimized over many years. This overhead is constant: its relevance diminishes as the RTT grows. From Fig. 3, RINA's Establishment and Transfer phases for RTT=12 ms take around 14.2 ms and 13.32 ms, respectively. These values are around 0.6-1.6 ms lower for TCP in both cases.

The most prominent result in Figure 5 also confirms our intuition: because only TCP without TFO needs an extra round-trip, it is slower than both TFO and RINA, and this difference is more significant when the RTT is larger. More surprisingly, even the RINA prototype code slightly outperforms TFO (0.08-0.09 ms). This is most likely a result of the cookie validation procedure in TFO which does not exist in RINA. This would also explain why TCP with TFO was slightly slower than without in the "first contact" case.

[3]https://wiki.linuxfoundation.org/networking/netem

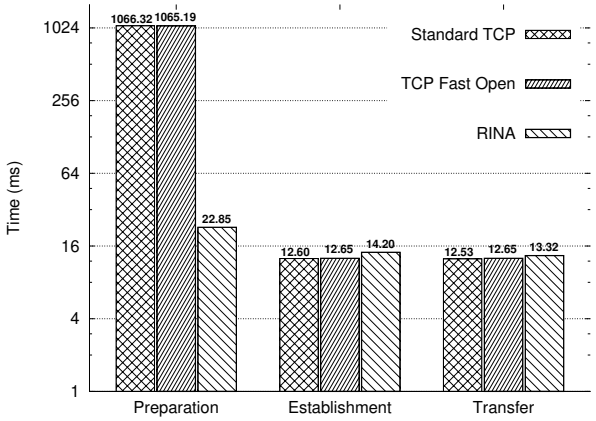

Fig. 3: "First contact". Min. RTT=12ms, avg. 12.52 / 12.57

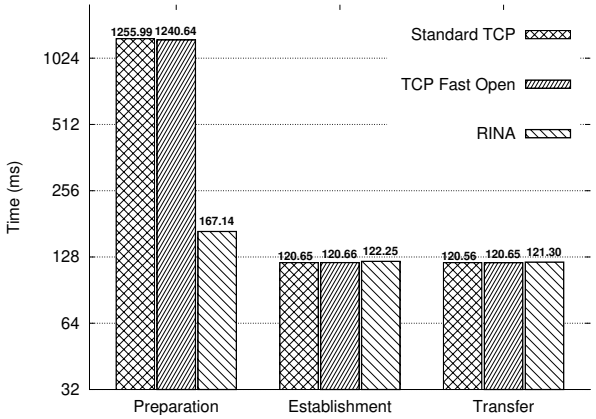

Fig. 4: "First contact". Min. RTT=120ms, avg. 120.52 / 120.57

VI. Related Work

"Switch-over", similar to what is described here, has been proposed for the Internet's transition to IPv6 under the name of "Happy Eyeballs" (HE), where a client issues two instead of one requests, one via IPv4 and one via IPv6 [17]. Slightly different from the RINA case where RINA can be assumed to work end-to-end if both hosts are RINA-enabled, HE needs to be performed end-to-end in order to see whether a protocol or protocol mechanism works along an Internet path, as middle-boxes in the Internet often drop or alter packets that do not match their expectations (for this reason, also

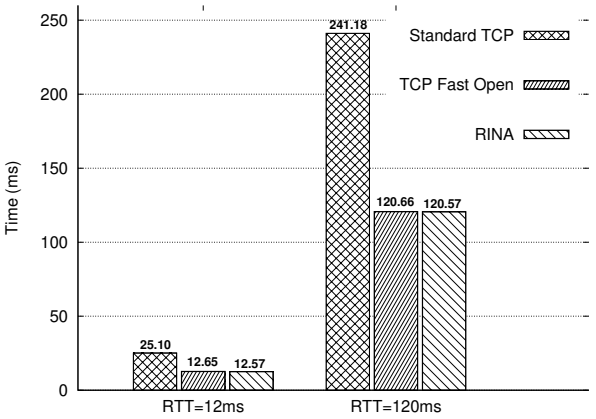

Fig. 5: Results of the "second contact" for both RTTs

TFO falls back to a regular non-TFO TCP handshake after an initial SYN timeout [4]). Because HE causes multiple potentially unnecessary requests to arrive at a server, it has raised scalability concerns. The authors of [13] investigated the impact of HE in terms of server load, finding that it is significant, yet dwarfed by the overhead of the security handshake for HTTPS. HE needs to use timers to check which protocol or protocol mechanism worked; the choice of these timers embeds a trade-off that has been investigated in [1].

At the transport layer, some browsers now apply a HE strategy between QUIC (over UDP) and TCP. More generally, the IETF Transport Services (TAPS) Working Group[4] is developing a set of standard documents that make applications independent of the choice of the transport protocol, allowing to benefit from protocols such as QUIC without involving the application programmer. Apple's Network.Framework (part of the beta for iOS 12 and macOS Mojave) implements TAPS principles. Similarly, the open source NEAT library [11] offers easy and protocol-independent access to all the functions of the SCTP and MPTCP protocols in addition to TCP and UDP. NEAT's policy system can control protocol and interface choice without application involvement.

TAPS provides an opportunity to implement and benefit from any kind of new technology underneath a common new interface that applications are talking to; therefore, it seems reasonable that RINA could operate underneath the TAPS interface just like IPv6, MPTCP, SCTP or QUIC.

VII. Conclusions

Following our comparison between RINA and the Internet, it would seem obvious that the Internet's more rigid, hard-coded nature lets it greatly benefit in terms of latency for initial communication. While our evaluation has indeed confirmed initial extra delay of RINA due to signaling overhead, this overhead entirely disappears in the "second contact" case.

We have found that the IRATI prototype implementation of RINA, which naturally cannot be expected to be on par with the highly optimized TCP/IP kernel code, has relatively minor, constant overhead compared to TCP/IP. RINA even performed slightly better than TFO in "second contact", probably because it does not need to validate cookies. Additional benefits—TFO-style communication in RINA *always* works and the first message is not limited by the allowed payload in a TCP SYN segment—make the case that a full "switch-over" to RINA can indeed be efficient. This motivates us to further investigate this line of deployment, e.g. by using a gateway to support clients without installing software in the end user equipment.

[4]https://datatracker.ietf.org/wg/taps/about

VIII. Acknowledgments

The authors were part-funded by the Research Council of Norway under its "Toppforsk" programme through the "OCARINA" project (http://www.mn.uio.no/ifi/english/ research/projects/ocarina/). The views expressed are solely those of the authors. We thank Marcel Marek and Eduard Grasa for helping us understand RINA's communication establishment.

References

[1] V. Bajpai and J. Schönwälder, "Measuring the effects of happy eyeballs," in *Proc. 2016 Applied Networking Research Workshop*, ser. ANRW '16. New York, NY, USA: ACM, 2016.

[2] G. Boddapati, J. Day, I. Matta, and L. Chitkushev, "Assessing the security of a clean-slate internet architecture," in *IEEE ICNP*, Oct 2012, pp. 1–6.

[3] R. Braden, "T/TCP – TCP Extensions for Transactions Functional Specification," RFC 1644 (Historic), pp. 1–38, Jul. 1994.

[4] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain, "TCP Fast Open," RFC 7413 (Experimental), Dec. 2014.

[5] D. Damjanovic, P. Gschwandtner, and M. Welzl, "Why is this web page coming up so slow? investigating the loss of SYN packets," in *NETWORKING 2009*, 2009.

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

[7] J. Day, I. Matta, and K. Mattar, "Networking is IPC: A guiding principle to a better internet," in *Proc. ACM CoNEXT*, 2008.

[8] J. G. Fletcher and R. W. Watson, "Mechanisms for a reliable timer-based protocol," *Computer Networks*, vol. 2, 1978.

[9] G. Gursun, I. Matta, and K. Mattar, "On the performance and robustness of managing reliable transport connections," 2009. [Online]. Available: https://open.bu.edu/handle/2144/1738

[10] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," IETF, Internet-Draft draft-ietf-quic-transport-16, Oct. 2018, work in Progress.

[11] N. Khademi, D. Ros, M. Welzl, Z. Bozakov, A. Brunstrom, G. Fairhurst, K. Grinnemo, D. Hayes, P. Hurtig, T. Jones, S. Mangiante, M. Tuxen, and F. Weinrank, "NEAT: a platform- and protocol-independent internet transport API," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 46–54, June 2017.

[12] A. M. Mandalari, M. Bagnulo, and A. Lutu, "TCP Fast Open: Initial measurements," in *Poster at CoNEXT 2015*, 2015.

[13] G. Papastergiou, K.-J. Grinnemo, A. Brunstrom, D. Ros, M. Tüxen, N. Khademi, and P. Hurtig, "On the cost of using happy eyeballs for transport protocol selection," in *Proc. 2016 Applied Networking Research Workshop*. ACM, 2016.

[14] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan, "TCP Fast Open," in *Proc. CoNEXT*. ACM, 2011.

[15] 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, pp. 20–25, March 2014.

[16] R. W. Watson, "Timer-based mechanisms in reliable transport protocol connection management," vol. 5, no. 1, 1981.

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