Congestion Control in the Recursive InterNetworking Architecture (RINA)

Peyman Teymoori*, Michael Welzl†, Stein Gjessing‡, Eduard Grasa§,
Roberto Riggio¶, Kewin Rausch‖, Domenico Siracusa**
*†‡Department of Informatics, University of Oslo, Norway. Email:{peymant, michawe, steing}@ifi.uio.no
§i2CAT, Barcelona, Spain. Email: eduard.grasa@i2cat.net
¶‖**CREATE-NET, Trento, Italy. Email: {roberto.riggio, kewin.rausch, domenico.siracusa}@create-net.org

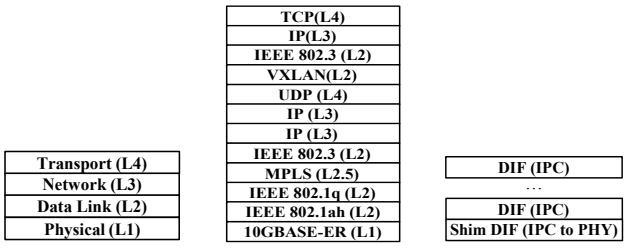
***Abstract*—RINA, the Recursive InterNetwork Architecture, is a novel "back to basics" type approach to networking. The recursive nature of RINA calls for radically different approaches to how networking is performed. It shows great potential in many aspects, e.g. by simplifying management and providing better security. However, RINA has not been explored for congestion control yet. In this paper, we take first steps to investigate how congestion control can be performed in RINA, and demonstrate that it can be very efficient because it is applied close to where the problem happens, and through its recursive architecture, interesting effects can be achieved. We also show how easily congestion control can be combined with routing, enabling a straightforward implementation of in-network resource pooling.**

I. Introduction

In the Internet, congestion control is embedded in a protocol at the transport layer or above – most commonly in TCP (which we will refer to in the following), but also in SCTP, DCCP and in certain RTP-based applications. The TCP protocol operates "end-to-end", where "end" is the host the application is running on. Although this might ensure scalability since interior network elements do not have to worry about congestion control,[1] the control is potentially executed far from where congestion appears in the network.

The Recursive Network Architecture (RINA) [2], [3], is a back to basics approach learning from the experience with TCP/IP [4] and other technologies in the past. In RINA, every layer (called a "Distributed InterProcess Communication (IPC) Facility" (DIF)) has the same set of mechanisms and goal: providing and managing the communication among its entities (called the "IPC Processes" (IPCPs)). All DIFs have the same internal structure. For example, each DIF has one transport protocol (the "Error and Flow Control Protocol" (EFCP)); this is a mechanism that cannot be changed. However, how this mechanism performs flow control, retransmission control or congestion control is defined as "policies" and can be programmed differently in various DIFs.

Many cases can be found in which there are layers in the network stack that do not follow the traditional "transport/network/data link/physical" architectural model (Fig. 1(a)). For example, in Fig. 1(b) we can see L2 over L2 (MAC-in-MAC), L2 over L3 (Ethernet over MPLS), L2 over L4 (VXLAN, L2TP), L3 over L3 (IP in IP, GRE, LISP), and so on. RINA allows theoretically an indefinite number of stacked DIFs to provide IPC services to each other (Fig. 1(c)). This is how it shows the ability to solve long-standing network problems of the Internet architecture (complexity, scalability, security, mobility, QoS or management, see [2], [5]–[7]). However, previous work on RINA has just focused on the above issues and did not investigate it on congestion control research.



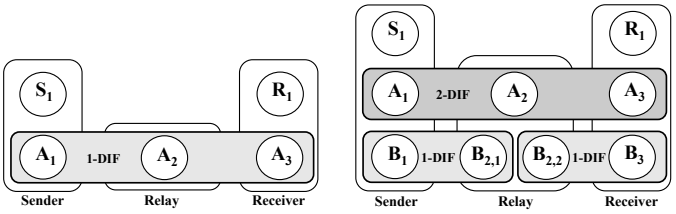
(a) The TCP/IP stack in theory (b) The TCP/IP stack in practice (c) RINA Stack

Fig. 1. Protocol stacks. (In RINA, shim DIF is the bottom DIF designed to operate on the physical layer, link layer, or any other specific technology.)

In this paper, we take a first look at how a congestion control method can work in this new architecture; by implementing a TCP-like congestion control policy, we compare it with similar approaches in the Internet, and discuss its benefits. We also elaborate how a range of different optimizations in congestion control can fit within a single framework. In RINA, *recursion* arises from the ability to arbitrarily arrange structurally-equivalent DIFs. Through simple topologies, we show that improvements that have been done to TCP such as Split-TCP on the internet "naturally appear" with RINA without their side effects; we present some DIF layouts to show how they solve some congestion control problems in the Internet. We also show that in RINA, each DIF can detect and manage the congestion for its resources, pushing back to higher layer DIFs when resources are overloaded. There, the "Relaying and Multiplexing" (RMT) task – another mechanism in every DIF – is in charge of forwarding the EFCP PDUs; it can load-balance the traffic by sending it on other paths, or recursively pushback upwards to achieve *in-network resource pooling*.

We will elaborate on the design of RINA congestion control

[1]It has been said that this design of TCP matches the end-to-end argument. This is true, but the end-to-end argument really only prohibits implementing *application-specific* functions inside the network [1]. Much like routing, congestion control addresses a problem that occurs inside the network (the end-to-end argument does also not forbid complex routing algorithms).



(a) End-to-end (b) Split

Fig. 2. Two possible RINA stack configurations by different organizations of "Distributed InterProcessCommunication Facilities" (DIFs).

in Section II. In Section III, we will take a look at how a TCP-like mechanism would play out in RINA, focusing on some simple "toy" scenarios. Section IV discusses RINA congestion control in the context of prior work, and Section V concludes.

II. Congestion Control in RINA

In RINA, every function is bound to one DIF, and DIFs can be of different sizes, e.g. two nodes connected to each other can form a DIF, or a public DIF can contain all the nodes of a network. Every DIF can have a different type of congestion control (or none at all). DIFs can also be stacked; the lower DIF serves the upper one based on a list of requirements that the upper DIF needs for its flows. This provides the flexibility of mapping several upper flows with the same requirements (e.g. QoS) to just one lower flow. Here, we only discuss congestion control-related modules of RINA; however, for further information on RINA, see [2].

As a simple starting point, consider a path with one intermediate hop, given by a physical setup of 3 nodes: a sender, a relay/router node in the middle, and a receiver, as shown in Fig. 2. S_1 and R_1 are the application instances sending and receiving packets, respectively. A_i (or B_i) is called IPC Process (IPCP); it is an application process that is a member of a DIF and locally implements the functionality to support and manage IPC using multiple sub-tasks. In Fig. 2(a), there is one DIF for the network that performs congestion control with a control loop from A_1 to A_3. The module controlling congestion in A_1 and A_3 is called EFCP. If congestion appears in the relay IPCP (in the RMT module of A_2 which is responsible for relaying PDUs), the RMT can, for example, set the Explicit Congestion Notification (ECN) field of PDUs. The sender's window or rate is a function reacting to congestion in this network. This scenario has its advantages in simplicity and scalability, but it has some obvious disadvantages too: A_1 cannot benefit from the knowledge specific to the two links below A_1–A_2 and A_2–A_3 (just like TCP can usually not benefit from link-layer knowledge), and if congestion appears in one of these links, EFCP needs a full round-trip-time (from A_1 to A_3 and back) to react.

The diagram in Fig. 2(b) shows two separate DIFs at layer 1 and hence two separate congestion controllers. Although the two DIFs have the same set of mechanisms, each one can have different *policies* for flow or congestion control that are tailored to the underlying link layer technology or physical link characteristics. The programmable functions inside DIFs are called *policies*. For example, EFCP can be seen as a family

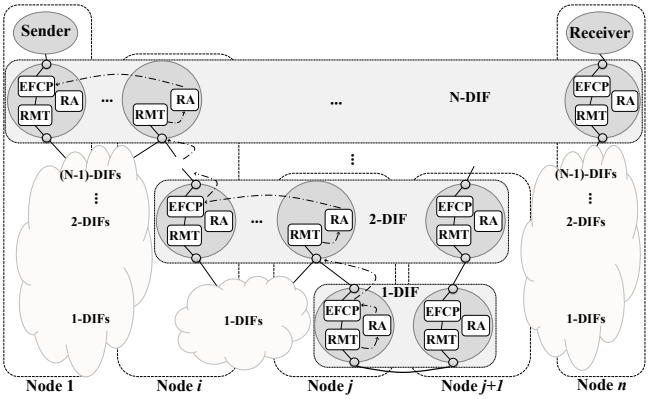
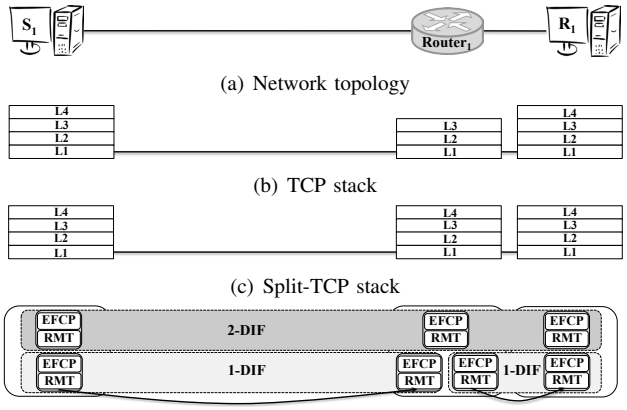


Fig. 3. Recursive congestion control in RINA: the recursive interactition of IPCP modules "Error and Flow Control Protocol" (EFCP), "Relaying and Multiplexing Task" (RMT), and "Resource Allocation" (RA).

of congestion/flow control protocols, and policy is a specific implementation. In the figure, we have another DIF on top. The top DIF's EFCP connection includes no congestion control in our example. EFCP connections inside the layer 1 DIFs include IPC processes $B_{2,1}$ and $B_{2,2}$, which are not directly visible to the layer 2 IPCP A_2. However, A_2 is connected to each one through a logical interface similar to a buffer, called *port*. If A_2 does not receive PDUs from $B_{2,1}$ as quickly as $B_{2,2}$ drains it, A_2 is able to tell $B_{2,1}$ that it can speed up. $B_{2,1}$ can then allow B_1 to send faster via EFCP's flow control. If, on the other hand, $B_{2,2}$ is sending data slower than it arrives at $B_{2,1}$, $B_{2,2}$'s buffer will become full. A_2 is then not able to forward PDUs anymore and, therefore, will have to stop taking them from $B_{2,1}$. Thus, $B_{2,1}$ will again obtain the information it needs to make B_1 slow down via EFCP's flow control.

We see that the way RINA controls congestion is a generalization of how it is done in the Internet: if there is only one DIF doing congestion control in the network, it operates in an end-to-end fashion. If two or more congestion controlled DIFs are concatenated, the end-to-end control loop is broken into shorter loops. As another interesting capability, RINA allows DIFs to be stacked, and upper DIFs can have their own congestion control policies. If, in Fig. 2(b), there are several flows from S_1 to R_1 through several EFCP connections from A_1 to A_3, packets of all of them are mapped to only one EFCP connection in the DIFs below; this means that at the lower DIFs, there is only one *aggregated* flow, and congestion control in these DIFs operates on aggregates. As we consider it an important feature, we generally refer to RINA's congestion control as "Aggregate Congestion Control" (ACC).

A general architecture of N layers is illustrated in Fig. 3. The dashed, curved arrows in the figure represent notification transmission between modules, and IPCPs are shown as circles. In case of congestion in the 1-DIF between nodes j and $j+1$, the Resource Allocation entity (RA) of the transmitting IPC Process sends a notification to the IPCP of the EFCP instance sending the PDU. This EFCP instance is located in the same IPCP. Congestion causes queue growth in this EFCP instance; when the queue reaches its maximum size limit, the EFCP instance shuts down its incoming port, and PDUs are



(a) Network topology

(b) TCP stack

(c) Split-TCP stack

(d) RINA stack. Only EFCP and RMT modules are shown in IPCPs.

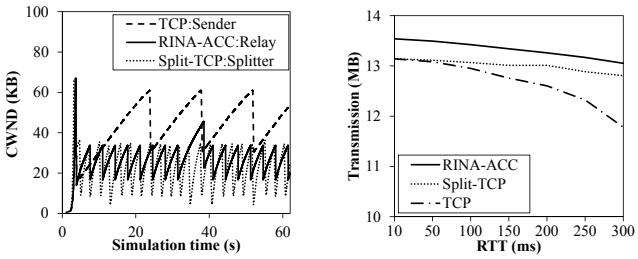
Fig. 4. The topology used for comparison and its correspoding stacks in end-to-end TCP, Split-TCP, and RINA.

backlogged in the upper RMT output port (the port will be unblocked again when the queue length decreases). Next, the output queue of the RMT in 2-DIF will reach its maximum threshold, and this continues; it sends a slow-down signal to the Resource Allocator (RA) module, and the RA sends a notification to the sending EFCP instance. This continues until the signal reaches the source of congestion, generating a "pushback" behavior that *emerges due to recursion.*

A full implementation of the scenario in Fig. 3 requires upstream notifications within a DIF, which can have implications on efficiency (probably positive) and scalability (probably negative) that are beyond the scope of the preliminary investigation presented here. We note, however, that scalability constraints are not as prohibitive here as they are in the Internet: because the Internet architecture essentially limits congestion control to the setup in Fig. 2(a), upstream notifications become impossible to use [8], whereas they merely constrain the maximum size of a DIF in RINA.

III. Simulations With TCP-like Congestion Control

To better understand the implications of placing congestion controlled DIFs next to or above each other, we have carried out simulations of a few "toy" scenarios using the OMNeT++ RINA module [9]. The simulator and scenarios can be found in the provided URL ([9]). We implemented a simple TCP Tahoe-like congestion control policy in EFCP (and, for the simulations in Sections III-A and III-B, nothing else: the effects were entirely achieved by configuring how DIFs are stacked). As we will see, TCP-like congestion control in RINA plays out in ways that are similar to certain functions that are getting deployed as "hacks" to the Internet infrastructure today (e.g. in performance-enhancing proxies (PEPs) [10]) as well as mechanisms that are proposed in the academic literature. In RINA, this behavior naturally appears as a result of layering, whereas the very design of PEPs illustrates the difficulty of overhauling the Internet's base architecture. In the next subsections, we will continue our discussions on three different arrangement types of DIFs which RINA allows us to do: consecutive DIFs, stacked DIFs, and "side-by-side" DIFs.



(a) CWND size of a run with RTT $= 200ms$ (in Kilo Bytes)

(b) Transmitted volume of data (in Mega Bytes)

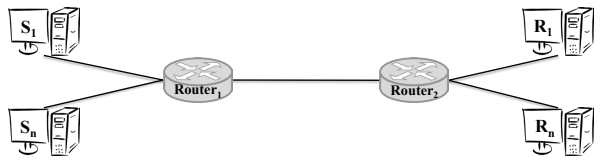
Fig. 5. Comparison results of a single flow in TCP, Split-TCP, and RINA.

A. Horizontal: Consecutive DIFs

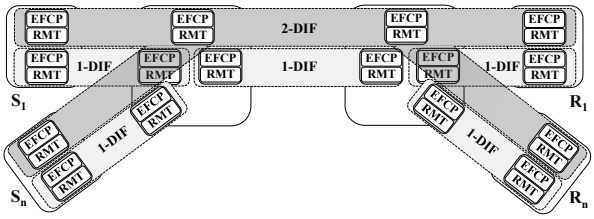
If we consider the description of Fig. 2(b) in the previous section and assume that the congestion control in the Layer 1 DIFs is TCP-like, what we have described is very similar to a PEP function called Connection Splitting [10]. TCP splitters divide TCP connections by "lying" to the end systems, acting as the receiver towards the sender and as the sender towards the receiver. As one significant difference to RINA-ACC, a TCP splitter must take care of end-to-end reliability, thereby breaking the end-to-end reliability semantics of TCP. While retransmitting packets from within the network can also have benefits and would easily be possible with RINA, it is not automatically tied to dividing the control loops as it is with TCP, and we do not consider it further here.

To compare RINA-ACC with Split-TCP and see the performance gain over end-to-end TCP, we simulated data transfers in the topology shown in Fig. 4; Router_1 acts as a splitter while using Split-TCP, which we have added to the OMNeT++ INET framework as implemented in [11]. The protocol stacks of the three approaches are also shown for clarity. The traffic sent by S_1 was a single large file, so there was only one flow from S_1 to R_1 in the network. The simulation was run for one minute, and after that we collected the total volume of data transmitted. The capacity of the link between the sender and the router was 10 Mbps, and for the link between Router_1 and R_1 it was 2 Mbps. Congestion appeared in the interface on the right-hand side of Router_1. We limited the output queue of this interface to the bandwidth-delay product (BDP) in the end-to-end TCP case. In the Split-TCP and RINA-ACC cases, we used the BDP of the Router_1–R_1 link for this buffer as well as the RMT's output buffer at the layer above.

The performance of end-to-end TCP, Split-TCP, and RINA with TCP per DIF is illustrated in Fig. 5. This diagram shows two separate and expected things: 1) Fig. 5(a) shows congestion window (CWND) sizes of the sender in end-to-end TCP ("TCP:Sender") and the second connections of Split-TCP and RINA ("Split-TCP:Splitter" and "RINA-ACC:Relay") between Router_1 and R_1; this implies that our implementation of TCP in RINA indeed performs very similar to Split-TCP (with a small throughput improvement because its pushback mechanism prevents packet drops and timeout), and 2) referring to Fig. 5(b), Split-TCP performs better than end-to-end TCP at large RTTs. The latter fact is known from earlier literature but it shows that our implementation of Split-TCP is correct.



(a) Network topology



(b) RINA stack: there is one 1-DIF between every pair of adjacent nodes, and a single 2-DIF on top which connects all the nodes.

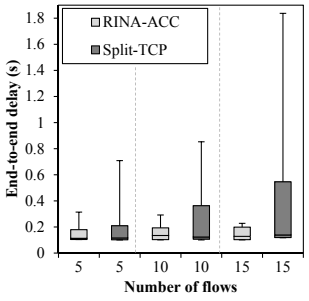
Fig. 6. Network topology for multiple flows and its RINA stack.

B. Vertical: Stacked DIFs

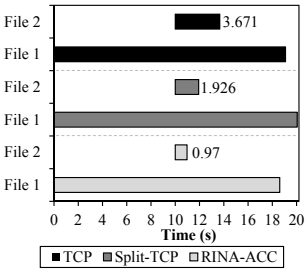
In the simplistic example above, the traffic carried by RINA came from one single end-to-end flow in the network. In a real RINA network where DIFs are stacked above each other, an N-DIF would carry an aggregate of flows from the (N+1)-DIF sitting above it. Edge router pairs would then only keep the congestion state of active flow aggregates between them. Here, RINA-ACC automatically avoids the competition between multiple end-to-end flows that occurs in the Internet today: when many end-to-end or Split-TCP flows individually push up the queue at the bottleneck, they harm each other via increased delay and loss, and it can be better to combine them [12], [13]. We simulated multiple flows competing in the network using the network topology shown in Fig. 6. Senders S_1 through S_n sent a large file to receivers R_1 through R_n, respectively. All the links had the same bandwidth; the $Router_1$–$Router_2$ link was the bottleneck. For RINA, the DIF structure is also indicated in Fig. 6: there were some consecutive lower DIFs and one upper-layer DIF on top. We compared RINA-ACC against the better Internet case from our previous simulations – Split-TCP, but with no aggregation.

In Fig. 7(a), end-to-end delay results of RINA-ACC and Split-TCP are shown in a box-and-whisker diagram; it shows the range and 10th percentile/median/90th percentile boxes of all packets in one simulation. Although the median of delay is almost the same, we observe that due to the competition among the TCP connections in the $Router_1$–$Router_2$ segment, some packets had much longer end-to-end delays, which also causes a higher jitter at receivers. In RINA-ACC, all traffic from the senders was carried through one flow between $Router_1$ and $Router_2$ with no competition. The effect of competition can be mitigated to some extent by employing Active Queue Management (AQM) in Split-TCP. However, AQM does not resolve the high jitter problem arisen by competition.

With RINA-ACC, we see a slight reduction in peak delay as the number of flows increases because at the start, more flows translate into more packets to be sent by the aggregated flow between $Router_1$ and $Router_2$, keeping its send buffer from draining and allowing its congestion window to grow faster.



(a) End-to-end delay



(b) Transmitting two files from one sender to one receiver.

Fig. 7. Benefits of RINA ACC.

This implies another benefit of ACC: flows take advantage of an already open window of the aggregate flow in their path for faster transmission and shorter delay.

The previous simulation scenario showed the advantage of aggregating flows in the core of the network. A similar argument holds when a number of flows originate from the same sender to the same receiver. Following an example from [14], a user might be surfing a web site while downloading a file from it – in this case, it can be more efficient to use only one congestion controller between the two nodes.

We simulated this scenario and show the results in Fig. 7(b). The network topology was the same as in Fig. 6 with $n = 1$. The sender sent two files to the receiver. The transmission of File 1 with the size of 20 MB started at time 0. File 2 was 500 KB, and its transmission started 10 seconds later. The horizontal axis of the figure shows time, and the bars show the start and finish times of each file for the three methods. Due to the aggregation of the second flow into the already started one in RINA-ACC, the second transfer can benefit from the large congestion window of the ongoing transmission which already has a better approximation of the available bandwidth between the two nodes. Moreover, compared to TCP and Split-TCP, there is less competition in this case which results in a shorter transmission time for the first file. In Split-TCP, the two flows were split into six pieces, and every two connections competed for one of the three links. Compared with the TCP scenario in which there was only one bottleneck link, here there were three bottleneck links because sometimes, for example, the total congestion window size in the splitter of $Router_1$ became smaller than the total size in the sender. In our simulation, this caused one timeout in the second connection of File 1; this is why it took around 1 second longer than with TCP.

The effect shown here with ACC is comparable to a Congestion Manager (CM) [15] or multi-streaming as in e.g. SCTP [14] – but with these mechanisms, the benefit shown in Fig. 7(b) only appears when multiple flows are used as described between the same sender and receiver. With RINA-ACC however, the two files could just as well be transferred between different sender-receiver pairs, and the same holds if there are hundreds or thousands of senders: File 2 can be transmitted faster if *any* sender is transmitting data at any given time. In-network aggregate congestion control is, therefore, a much more powerful mechanism than previously proposed methods for congestion control coupling.

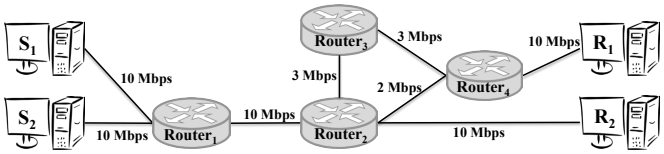


Fig. 8. A toy topology for evaluating INRP (adopted from [19]).

C. Around: In-Network Resource Pooling

Resource pooling [16] has recently gained significant attention from the routing and congestion control point of view. There exist some methods at the transport layer like Multi-Path TCP (MPTCP) [17] or at the routing layer like Equal-Cost Multi-Path Routing (ECMP) [18] with the aim of load-balancing. However, they operate independently. A recent approach to enable a combination of these two types of mechanisms was presented by the name of In-Network Resouce Pooling (INRP) in [19]. This work claims that dynamic routing/detouring compared with the Internet's static routing, as a way of reacting to congestion by finding alternative lower-load routes, combined with a hop-by-hop congestion control mechanism in case there is no other low-load path towards the destination, is beneficial. However, this approach might face Internet-specific problems such as head-of-line blocking in case of detouring TCP traffic [20].

RINA enables us to cope with these problems with its layered architecture; it offers various options for reliable out-of-order delivery between any two EFCP instances, and it is up to each layer how to serve upper layers (DIFs). For example, as proposed by [19], [21], DIFs are automatically able to serve upper DIFs with multiple flows like *flowlets* through multiple paths. These paths are provided through multiple side-by-side lower DIFs operating on different paths. Detouring can also be bound to some lower DIF in the core of a network, and that DIF might even provide in-order-delivery to upper DIFs. At the edge of a DIF, delay from head-of-line blocking is bounded whereas in TCP it is accumulated along the path.

To show how easily RINA can provide resource pooling, we implemented a load-based routing policy in the RINA simulator in OMNeT++. This policy is coupled with another short-term queue size monitoring policy to react to local congestion in an output queue. In case of congestion, first another low-load path to the destination is looked up. If no other path is available, the congestion control module reacts.

Fig. 8 shows the simple topology that was used to explain in-network resource pooling in [19]. In case of proper load-balancing, both S_1–R_1 and S_2–R_2 connections should be allocated a 5 Mbps share. The main advantage of resource pooling here is that it provides global fairness and local stability [19]. The corresponding RINA stack is similar to the one in Fig. 6 meaning that there is one 1-DIF per link, and a single 2-DIF connecting all the nodes on top. We ran this scenario in RINASim for one minute, and measured the volume of data received by R_1 and R_2. Jain's fairness index [22] with three digits precision was 0.999, which shows global fairness while local stability was provided through RINA-ACC as shown before.

IV. Discussion and Implications

Because it operates at the transport layer, it is impossible for TCP to always do "the right thing" for every network segment. By measuring only the round-trip time (RTT) and packet loss, it is very difficult to optimally adapt the transmission rate of the sender when the path to the receiver is a chain of technologies (e.g. WLAN, Ethernet, satellite, 3G). TCP cannot make any form of link technology-specific decision. This problem is compounded by well-known TCP issues like e.g. its inability to distinguish between congestion losses and losses that are due to link impairments. It should not be surprising that there is a huge amount of research work on cross-layering with a focus on TCP-over-X, where X is any link layer technology and X is assumed to be the only problematic link. Similarly, given that TCP must work everywhere, it should not be surprising that such research does not typically influence the TCP standard. TCP's end-to-end congestion control also does not scale well in several dimensions:

1) *The diameter of the network*. The effectiveness of any congestion control scheme will deteriorate with increasing network diameter. TCP maximizes this effect because it operates in "rounds" based on the round-trip time. This is particularly problematic in high bandwidth networks where the capacity does not become a limit, especially with short flows that are common in web traffic. Such flows typically terminate in TCP's slow-start phase, making the completion time strictly a function of the round-trip time. Recently, there has therefore been increased interest in reducing the number of round-trips for web traffic [23].

2) *The number of flows*. When multiple flows traverse a path, they compete for the available bandwidth, pushing up the queues and creating delay and loss. As shown in [12], jointly controlling them as a group can lead to much better behavior and enable precise prioritization between flows. This functionality is currently being proposed for WebRTC in the IETF RMCAT Working Group [24].

3) *The bottleneck link capacity*. TCP often poorly saturates high-capacity links due to its linear increase in standard TCP's congestion avoidance phase. This has been addressed approximately a decade ago; now, the most prominent solution is the CUBIC congestion control mechanism that is used by default in Linux hosts [25]. When CUBIC and other related mechanisms were proposed, explicit feedback-based schemes were found to operate best (e.g. XCP [26], RCP [27], MaxNet [28], and CADPC/PTP [29]).

RINA can solve all of these problems by 1) breaking up the long control loop into shorter ones, 2) controlling flow aggregates inside the network, and 3) enabling the deployment of arbitrary congestion control mechanisms per DIF.

As we have seen in the previous section, with TCP, the behavior emerging with RINA can resemble Performance Enhancing Proxies (PEPs) [30]. PEPs usually break end-to-end connections into several closed-loop connections; in each connection, a congestion control scheme which considers local link characteristics can be exploited [31]. Also, hop-by-

hop congestion control has gained much attention in wireless networks due to serious problems of end-to-end methods [32].

Internet PEPs have several disadvantages. In addition to the already mentioned reliability problem (see Section III-A), complexities in using IPsec and SSL [33] arise because security is an end-to-end function in these cases. This is why works such as [33] are motivated by designing PEP-less solutions. PEPs also do not scale well with the number of flows [32], [34]. A mapping of incoming-outgoing connection pairs to interfaces must be maintained to be able to e.g. slow down the correct incoming connections when congestion is experienced on an outgoing interface [35]. TCP's fast retransmission/recovery is triggered by triple duplicate ACKs which needs a window size of at least 4 packets, meaning that a splitter's buffer size (advertised window) must be at least 4 times the number of flows [34]; otherwise, performance degrades. In addition, in terms of the processing delay, running a separate TCP instance for every flow is expensive [19].

The layered architecture of RINA does not have these problems of Internet PEPs because 1) security is a per-DIF function: PDUs are protected as they cross DIF boundaries [7], and because each DIF has its own congestion control, splitting encrypted connections is not problematic, and 2) flows towards each next hop are aggregated at the lower DIFs, which means that there is much less state to maintain.

In TCP splitters, buffer sizes have also been shown to affect the performance, but, as shown by [36], the required splitter's buffer size is fairly modest, and it is independent of the number of flows. The splitter's buffer size scales linearly with BDP of the bottleneck link [34]. In our evaluations, we used a buffer of the size of its link's BDP in RINA for RMT output queues at upper DIFs. Running the same simulation with different buffer sizes revealed that in our scenarios, buffers of at least this size result in a high performance, close to the upper limit, while imposing a reasonable queuing delay.

A. Stability and Scalability

We have discussed benefits of breaking up control loops; at the extreme end of this approach we have hop-by-hop congestion control. There has been a concern regarding stability of hop-by-hop congestion control methods since many years ago because some unstable behavior was observed in some networks [37], [38]. However, as [38] argues, "the fear of instability" might be a result of the unsuccessful use of non-discriminatory on-off type controls.

Many works have investigated the stability of hop-by-hop congestion control, e.g. [32], [35], [37]–[39]. They showed that using per-hop controllers accelerates the response to bandwidth changes in the path, and that series of control loops are stable under certain conditions. These methods usually use a rate-based mechanism with proven stability properties. Split-TCP consists of several consecutive TCP connections, and each TCP connection has been shown to have stability properties [35], [40] especially for shorter RTTs. In [41], it is also proven that even in case of not using TCP receiver-window based backpressure in Split-TCP, the system is stable under certain conditions.

In RINA, consecutive DIFs operate similarly to Split-TCP, and due to our TCP-like controller implementation, we inherit the same stability properties. If DIFs are layered, as shown in Fig. 3, it can be observed that we have a concatenation of TCP controllers; however, these controllers, from the 1-DIF, cross the layers up to the EFCP instance at the sender node in the N-DIF. In other words, there is one active controller between every two consecutive EFCP senders shown in the figure down to the bottleneck link. This implies that in the recursive form we have the same stability properties. Hop-by-hop rate-based congestion controllers with proven stability which were mentioned above can also be used in RINA. Due to the inherent flow aggregation in RINA, using this kind of controllers imposes limited overhead on intermediate nodes in RINA; this overhead has, however, been a major obstacle in deploying these approaches previously [35].

An important issue regarding using stable hop-by-hop congestion controllers is how well they can scale [32] because a larger sequence of controllers might affect stability. However, works such as [40] show that it is possible to achieve a scalable and stable method for arbitrarily large network topologies and arbitrary delays.

In general, any form of hop-by-hop congestion control or connection splitting can be viewed as a concatenation of several controllers. There is a large number of works in the control theory literature on the stability of interconnected systems, cf. [42]–[44]; each of them considers different assumptions in the system such as parameter dependencies between subsystems or time-varying properties. These works show that even under arbitrary interconnection of systems, stability can be achieved. These results can be helpful in the design of stable congestion controllers for RINA under the same assumptions.

RINA theoretically allows an infinite number of DIFs to be stacked; each DIF has its own buffers, and hence, the total end-to-end buffer size in RINA is another matter of further investigation. We do however not expect this to become a significant issue, as hop-by-hop rate-based congestion control schemes such as [32], [39] have already been shown to be more efficient than end-to-end approaches in terms of buffer usage (e.g. peak buffer size) at intermediate nodes.

V. Concluding Remarks

The Internet's end-to-end'ness of congestion control and its static routing are inevitable by-products of its architectural design. With RINA, the natural way of applying such algorithms is very different, with control executed closer to where the problem is. The main achievement of RINA is that the series of hacks and patches found in the Internet, with its problems that we have discussed in the case of congestion control, are not required. RINA is therefore an ideal vehicle for investigating drastic changes to how congestion control, and in-network resource pooling as another example, could be done, and it provides a suitable framework with many promising dimensions for future research.

In this paper, through some simple evaluation topologies, we have shown that congestion control in RINA "naturally" exhibits properties of various improvements that have been made to (or at least proposed for) the Internet, without inheriting the problems that come from imposing these mechanisms on an architecture that was not made for them (all the problems that PEPs have). However, it is just as "natural" that such a drastic departure from common methods requires answering many new questions, such as: 1) how to effectively manage buffers between DIFs, 2) which hop-by-hop congestion controllers are best to use, given their stability and scalability properties, 3) how to best apply in-network resource pooling, 4) effects of different congestion control policies at lower DIFs on congestion control policies of upper DIFs, and 5) how large a DIF can be without performance degradation, and how its scalability can be improved. We believe that these are exciting issues for future work, and hope that this paper inspires others to also consider RINA for congestion control research.

REFERENCES

[1] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 277–288, Nov 1984.

[2] J. Day, I. Matta, and K. Mattar, "Networking is IPC: a guiding principle to a better internet," in *Proc. ACM CoNEXT*, 2008, p. 67.

[3] J. Day, *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall, 2007.

[4] ——, "How in the heck do you lose a layer!?" in *Network of the Future (NOF), 2011 International Conference on the*, Nov 2011, pp. 135–143.

[5] G. Gursun, I. Matta, and K. Mattar, "On the performance and robustness of managing reliable transport connections," CS Department, Boston University, Tech. Rep., 2009, bUCS-TR-2009-014.

[6] V. Ishakian, J. Akinwumi, F. Esposito, and I. Matta, "On supporting mobility and multihoming in recursive internet architectures," *Computer Communications*, vol. 35, no. 13, pp. 1561–1573, 2012.

[7] J. Small, "Patterns in network security: An analysis of architectural complexity in securing recursive inter-network architecture networks," Master's thesis, Boston University Metropolitan College, 2012.

[8] F. Gont, "Deprecation of ICMP Source Quench Messages," RFC 6633 (Proposed Standard), Internet Engineering Task Force, May 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6633.txt

[9] RINASim, "Rina simulator," https://github.com/kvetak/RINA/tree/UiO-ACC/examples/SmallNetwork2, 2015.

[10] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," RFC 3135 (Informational), Internet Engineering Task Force, Jun 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3135.txt

[11] A. Bakre and B. Badrinath, "I-tcp: indirect tcp for mobile hosts," in *Proc. IEEE ICDCS*, May 1995, pp. 136–143.

[12] S. Islam, M. Welzl, S. Gjessing, and N. Khademi, "Coupled congestion control for RTP media," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, p. 101, Aug 2014.

[13] F.-C. Kuo and X. Fu, "Probe-Aided MulTCP: An aggregate congestion control mechanism," *SIGCOMM CCR*, vol. 38, no. 1, Jan 2008.

[14] M. Welzl, F. Niederbacher, and S. Gjessing, "Beneficial transparent deployment of sctp: the missing pieces," in *Proc. IEEE GLOBECOM*, 2011, pp. 1–5.

[15] H. Balakrishnan, H. S. Rahul, and S. Seshan, "An Integrated Congestion Management Architecture for Internet Hosts," in *SIGCOMM*, 1999, pp. 175–187.

[16] D. Wischik, M. Handley, and M. B. Braun, "The resource pooling principle," *ACM SIGCOMM CCR*, vol. 38, no. 5, pp. 47–52, 2008.

[17] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, Implementation and Evaluation of Congestion Control for Multipath TCP," in *NSDI*, vol. 11, 2011, pp. 8–8.

[18] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," RFC 2992 (Informational), Internet Engineering Task Force, Nov 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2992.txt

[19] I. Psaras, L. Saino, and G. Pavlou, "Revisiting resource pooling: The case for in-network resource sharing," in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XIII. New York, NY, USA: ACM, 2014, pp. 24:1–24:7.

[20] B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, I.-J. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl, "Reducing internet latency: A survey of techniques and their merits," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1–1, 2014.

[21] S. Sinha, S. Kandula, and D. Katabi, "Harnessing TCPs Burstiness using Flowlet Switching," in *3rd ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, San Diego, CA, November 2004.

[22] R. Jain, D. Chiu, and W. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems," DEC, Tech. Rep. TR-301, 1984.

[23] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan, "Reducing web latency: the virtue of gentle aggression," in *Proc. ACM SIGCOMM*, 2013.

[24] M. Welzl, S. Islam, and S. Gjessing, "Coupled congestion control for RTP media," Internet-draft (work in progress) draft-welzl-rmcat-coupled-cc-05.txt, 2015.

[25] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-friendly High-speed TCP Variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul 2008.

[26] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 89–102, 2002.

[27] N. Dukkipati, N. McKeown, and A. G. Fraser, "RCP-AC: Congestion control to make flows complete quickly in any environment," in *Proceedings of IEEE INFOCOM*, 2006, pp. 1–5.

[28] B. P. Wydrowski, L. L. Andrew, and I. M. Mareels, "Maxnet: Faster flow control convergence," in *Proc. Networking*. Springer, 2004.

[29] M. Welzl, *Scalable performance signalling and congestion avoidance*. Springer Science & Business Media, 2012.

[30] C. Caini, R. Firrincieli, and D. Lacamera, "PEPsal: a Performance Enhancing Proxy for TCP satellite connections," *IEEE Aerospace and Electronic Systems Magazine*, vol. 22, no. 8, pp. 7–16, 2007.

[31] A. Kapoor, A. Falk, T. Faber, and Y. Pryadkin, "Achieving faster access to satellite link bandwidth," in *Proc. IEEE INFOCOM*, 2006.

[32] Y. Yi and S. Shakkottai, "Hop-by-hop congestion control over a wireless multi-hop network," *IEEE/ACM ToN*, vol. 15, no. 1, pp. 133–144, 2007.

[33] T. T. Thai, D. M. L. Pacheco, E. Lochin, and F. Arnal, "SatERN: a PEP-less solution for satellite communications," in *Proc. IEEE ICC*, 2011, pp. 1–5.

[34] J. Zhu, S. Roy, and J. H. Kim, "Performance modelling of TCP enhancements in terrestrial-satellite hybrid networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. 4, pp. 753–766, 2006.

[35] S. Bohacek, "Stability of hop-by-hop congestion control," in *Proc. IEEE Decision and Control*, vol. 1, 2000, pp. 67–72.

[36] M. Luglio, M. Y. Sanadidi, M. Gerla, and J. Stepanek, "On-board satellite "split tcp" proxy," *Selected Areas in Communications, IEEE Journal on*, vol. 22, no. 2, pp. 362–370, 2004.

[37] V. Kulkarni, S. Bohacek, and M. Safonov, "Stability issues in hop-by-hop rate based congestion control," in *Proc. Annual Allerton Conference on Communication Control and Computing*, vol. 36. University of Illinois, 1998, pp. 79–88.

[38] P. P. Mishra and H. Kanakia, "A hop by hop rate-based congestion control scheme," in *ACM SIGCOMM CCR 22(4)*, 1992.

[39] P. P. Mishra, H. Kanakia, and S. K. Tripathi, "On hop-by-hop rate-based congestion control," *IEEE/ACM TON*, vol. 4, no. 2, pp. 224–239, 1996.

[40] F. Paganini, J. Doyle, and S. Low, "Scalable laws for stable network congestion control," in *Proc. IEEE Decision and Control*, 2001.

[41] F. Baccelli, G. Carofiglio, and S. Foss, "Proxy caching in split TCP: Dynamics, stability and tail asymptotics," in *INFOCOM*, 2008.

[42] C. Langbort, R. S. Chandra, and R. D'Andrea, "Distributed control design for systems interconnected over an arbitrary graph," *IEEE Trans. Automatic Control*, vol. 49, no. 9, pp. 1502–1519, 2004.

[43] C. Maffezzoni, N. Schiavoni, and G. Ferretti, "Robust design of cascade control," *IEEE Control Systems*, vol. 10, no. 1, pp. 21–25, 1990.

[44] N. Motee and A. Jadbabaie, "Optimal control of spatially distributed systems," *IEEE Trans. Automatic Control*, vol. 53, no. 7, pp. 1616–1629, 2008.