A Congestion Control Framework Based on In-Network Resource Pooling

Sergi Rene, Onur Ascigil, Ioannis Psaras, *Member, IEEE*, and George Pavlou, *Fellow, IEEE*

***Abstract*— Congestion control has traditionally relied on monitoring packet-level performance (*e.g.*, latency, loss) through feedback signals propagating end-to-end together with various queue management practices (*e.g.*, carefully setting various parameters, such as router buffer thresholds) in order to regulate traffic flow. Due to its end-to-end nature, this approach is known to transfer data according to the path's slowest link, requiring several RTTs to transmit even a few tens of KB during slow start. In this paper, we take a radically different approach to control congestion, which obviates end-to-end performance monitoring and careful setting of network parameters. The resulting *In-Network Resource Pooling Protocol (INRPP)* extends the resource pooling principle to exploit in-network resources such as router storage and unused bandwidth along alternative sub-paths. In INRPP, content caches or large (*possibly bloated*) router buffers are used as a place of temporary custody for incoming data packets in a store and forward manner. Data senders push data in the network and when it hits the bottleneck link, in-network caches at every hop store data in excess of the link capacity; nodes progressively move/send data (from one cache to the next) towards the destination. At the same time *alternative sub-paths* are exploited to move data faster towards the destination. We demonstrate through extensive simulations that INRPP is TCP friendly, and improves flow completion time and fairness by as much as 50% compared to RCP, MPTCP and TCP, under realistic network conditions.**

***Index Terms*— Transport protocol, resource pooling, multipath.**

I. Introduction

TRANSPORT-LAYER protocols have been traditionally establishing end-to-end sessions, that is, from the sender to the receiver of data. This model fitted well the traditional client-server model of computing where clients are requesting data from a server that (potentially) applies computation to permanently stored data and sends the requested data back to the client. The assumption has always been that in-network elements (*i.e.*, network routers) are "dump" devices that do not possess storage or computation capabilities and should therefore, exclusively focus on packet forwarding.

Middleboxes, distributed cloud computing and more recent developments in the areas of edge and fog computing necessitate reconsideration of this traditional model as more and more functionality is taking place inside the network, mid-path from the server to the client. In-network storage [1] and computation [2] is gradually transforming "forward-only" routers to server-like elements, which can increasingly store and/or process incoming data. In other words, as storage and compute become cheaper, in-network elements can increasingly serve as independent servers.

In view of these advances, we consider it necessary to rethink the design of transport-layer protocols. In this paper, we start from the assumption that *network routers possess increased amounts of storage* and can therefore, act as temporary storage nodes. Ultimately, the goal is to overcome some of the weaknesses of the current design, namely, *eventual packet loss* (which in turn has triggered false bufferbloat designs[1]), *low link utilisation* (due to single-path transmission, or end-to-end multipath only [3], [4]) and *slow responsiveness to utilise available bandwdith [5]*, which in turn results to increased flow completion times, even in cases of very short flows and bandwidth availability [6]. These design weaknesses force ISPs to be conservative and overprovision their networks [4] resulting in increased maintenance costs. To address those issues, we design a framework which pools mid-path storage and link resources together to transfer data across the end-to-end path. According to the *In-Network Resource Pooling Protocol* (INRPP) [7], data can move from one hop to the next along the path from the original server towards the client. INRPP includes a group of mechanisms that guarantee: i) zero packet loss in intermediate network routers, ii) network stability, and iii) increased link utilisation.

For the purposes of this work, we ignore in-network computation at intermediate network nodes, but it is trivial to extend our framework to include such capability.

A. High-Level Overview

The In-Network Resource Pooling Protocol is composed of three main operational states: *i) push:* content is pushed as far in the path as possible in an open-loop, processor sharing manner [8]; the sending rate is based on the path's hop-by-hop bandwidth resources to take advantage of underutilised links;

Manuscript received February 6, 2020; revised November 11, 2020 and September 2, 2021; accepted October 14, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor R. Lo Cigno. This work was supported by the EPSRC through the COMIT Project under Grant EP/K019589/1. *(Corresponding author: Onur Ascigil.)*

Sergi Rene and George Pavlou are with the Department of Electronic and Electrical Engineering, University College London (UCL), London WC1E 6BT, U.K.

Onur Ascigil is with the School of Computing and Communications, Lancaster University, Lancaster LA1 4YW, U.K., and also with the Department of Electronic and Electrical Engineering, University College London (UCL), London WC1E 6BT, U.K. (e-mail: o.ascigil@ucl.ac.uk).

Ioannis Psaras is with Protocol Labs, San Francisco, CA 94104 USA, and also with the Department of Electronic and Electrical Engineering, University College London (UCL), London WC1E 6BT, U.K.

Digital Object Identifier 10.1109/TNET.2021.3128384

[1] https://www.bufferbloat.net

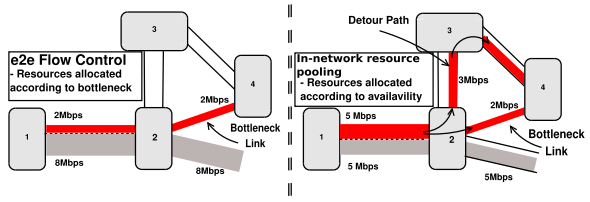


Fig. 1. Left: *e2e*: Bandwidth is split according to the slowest link. Right: *INRPP:* Bandwidth is split equally up to the bottleneck link. Detour applies to guarantee stability.

ii) store and detour (S&D): when (and if) data reaches a bottleneck link, the excess data is stored in local memory. At the same time, detour paths towards the destination are sought in order to utilise extra available resources; *iii) backpressure:* if detour paths do not exist and the node's local memory is filled up, the system enters a backpressure mode of operation [9], [10] to avoid overflowing of the local memory.[2] During the backpressure state, the nodes enter a closed-loop mode.

All three main operational states of INRPP are necessary to guarantee full network utilisation, network stability and flow fairness. INRPP states are interdependent similarly to the different phases and mechanisms of traditional transport protocols (*e.g.*, TCP) - disabling one can rip the protocol apart. In particular, the *push* state is similar to TCP's slow-start and intends to speed-up data transfer when bandwidth is available. The *Store & Detour* state is similar to the multipath feature of recent proposals to exploit multihoming and multiplicity of available *e2e* paths [11], [12]. INRPP, however, can also take advantage of *mid-path multipath*, as opposed to traditional end-host-based multihoming only, which as we show later provides significant performance gains. Finally, the *backpressure* state is similar to TCP's AIMD-based congestion avoidance. We argue and show through extensive evaluations that the combination of these three main operational states strike the right balance between aggressiveness and responsiveness to utilise all available bandwidth resources, eliminate packet loss, optimise performance and guarantee stability.

As a representative example of INRPP's main operation, consider two flows in the topology of Fig. 1. According to traditional *e2e* transport design (left part), the flow that traverses the bottleneck link (2-4) would achieve 2Mbps throughput, while the second flow would dominate the shared link (1-2) and achieve 8Mbps.

In contrast, according to the In-Network Resource Pooling Protocol introduced here (right part of Fig. 1), the shared link (1-2) is split equally among the two flows. Node 2 is acting as the temporary server for incoming data. Node 2 is therefore temporarily storing incoming data and has two options: *i)* find alternative routes to reach node 4, or *ii)* enter backpressure mode and notify node 1 to reduce its sending rate (closed-loop system to avoid extensive caching at node 2).

B. Platform Essential Elements

INRPP inherently prerequisites availability of alternative sub-paths (to allow for detours) and in-network content caches.

[2]We refer to storage, memory and cache interchangeably to denote local, in-network storage resources.

TABLE I
AVAILABILITY OF DETOUR PATHS IN REAL TOPOLOGIES

Network	1-hop Detours	Number of detour paths (% of col. 2)					Max 1-hop
		1	2	3	4	5+	
Telstra	68.75 %	27.45	30.09	11.5	7.9	23.06	38
Sprintlink	57.3 %	44.9	20.7	14.9	6.8	57.6	27
Ebone	51.8 %	55.5	28.78	10.6	3.53	1.5	10
Verio	71.75 %	23.56	18.01	13.45	12.56	32.52	25
Tiscali	24.44 %	60.60	19.19	12.12	5.05	3.03	8
Level3	92.30 %	13.40	14.10	12.42	13.02	47.06	95
Exodus	50.33 %	50.67	17.93	16.59	8.96	5.82	6
VSNL	25 %	100	0	0	0	0	1
AT&T	34.84 %	56.07	17.68	11.88	4.7	9.67	24

We look in detail into the extra investment needed to add cache capacity in routers in Section III-D, and we conclude that it is feasible to build the cache system required at a reasonable cost. In order to prove the feasibility of having one or more detour paths to divert excessive traffic at the router level, we analysed a set of real topologies (from Rocketfuel [13]) for nine ISPs (see Table I). Indeed, we find that six out of the nine real network topologies analysed can provide at least one 1-hop detour path on more than 50% of links, reaching up to 92.3% for Level-3 topology (second column). Columns 3-7 show how the percentage of 1-hop detour paths (shown in column 2) is split between 1, 2, 3, 4 and 5+ detour paths. We see that, in most cases, when a detour path exists, there is more than one detour sub-path for the link. The extreme case of the Level3 network shows that 47.06% of its edge-to-edge paths with at least one detour sub-path have five or more 1-hop detour sub-paths. The final column in Table I is the maximum number of 1-hop detour sub-paths that the topology has for at least one of its links. Overall, we observe that networks are rather well-connected and would realistically allow for mid-path detouring of excessive traffic.

The rest of the paper is organized as follows. In Section II, we present an overview of our framework, describing the main operation of an INRPP network. In Section III, we describe further details on the operation of INRPP including detouring mechanism, end-point functionality, and a cost-effective cache system for INRPP routers. Then, we provide results from a rigorous set of experiments in Section IV to evaluate our framework. In Section V, we discuss real-world deployment of INRPP. Finally, we present the related work followed by conclusions in Section VI and Section VII, respectively.

II. INRPP FRAMEWORK OVERVIEW

In this section, we first discuss in detail the operational states of INRPP router interfaces in Section II-A. Then, in Section II-B, we describe how transitions in interface states affect flow states along the *e2e* path.

A. INRPP Router States

An INRPP router in a simple topology is shown in Fig. 2, where router R_1 has a detour path through R_3 to reach R_2. Each interface of a router has a dedicated cache (apart from the buffer), which operates in one of INRPP's *operational states*. State transitions are triggered by cache occupancy oscillations which we monitor through a low and a high threshold, as we explain below.

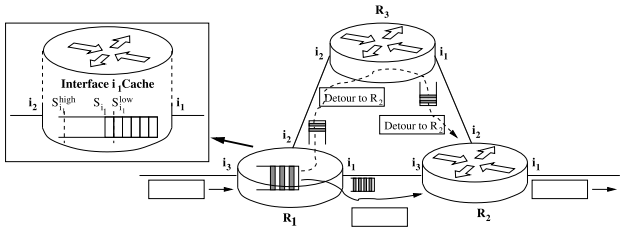


Fig. 2. INRPP router (zoomed in the box on the left) and detour example: Router R_1 has a detour path through R_3 to reach R_2 (dashed lines).

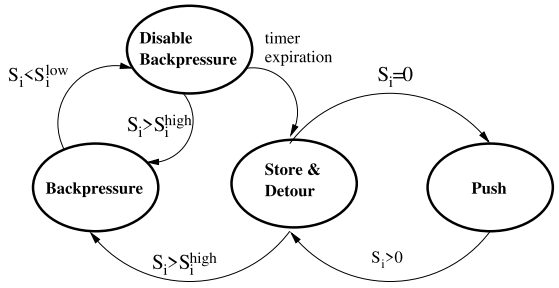


Fig. 3. State transition diagram of interface i.

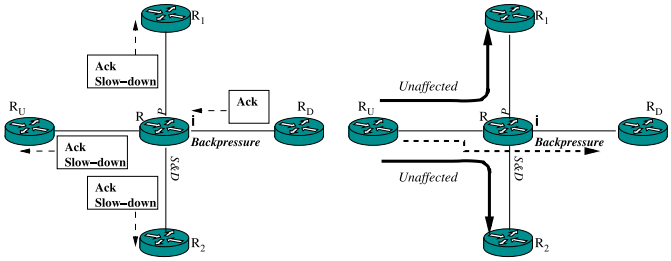


Fig. 4. Backpressure Example: On the left-side is the placing of slow-down notifications in ACK packets (originating from the end-points) and on the right-side is the slowing-down of traffic shown with dashed line.

At any given time, an INRPP router interface is in one of the three main states: *Push* (P), *Store and Detour* (S&D), *Backpressure* (B), or the transitional state *Disable Backpressure* (DB). The state transition diagram is given in Fig. 3 and our notation in Table II.

1) Push (P) State: When the outgoing interface link is not fully utilised and the cache of the interface is empty (each interface has its own dedicated cache as mentioned earlier), the router interface is in *push* state. In the push state, the router forwards incoming data straight to its outgoing interface without using its cache storage or detour interfaces. The push state is similar to the TCP slow-start, and its main purpose is to rapidly push data out as long as there is available bandwidth at the outgoing interface link. When the outgoing interface buffer (*i.e.*, queue) starts building up and eventually overflows, the excess data is sent to the interface cache, at which point the state of the interface switches to *Store-and-Detour*.

2) Store and Detour (S&D) State: The purpose of the S&D state is to send data out of the cache rapidly by exploiting the pool of available bandwidth on the alternative detour paths. While its interface i_1 is in S&D state, router R_1 in Fig. 2 starts storing incoming data at interface i_1's cache. During this state, interface i_1 pulls packets directly from the cache and sends them out at the maximum rate of its outgoing link, denoted $C_{R_1}^{i_1}$. At the same time, router R_1 continuously monitors the share of residual capacity on interface i_1's detour path (through node R_3 in Fig. 2) through *probe packets* (explained in Section III-A). If residual capacity is available, R_1 forwards data through this path too.

In order to keep track of cache oscillations and act accordingly, we introduce a lower- and an upper-bound threshold for each interface i, S_i^{low} and S_i^{high}, respectively. As long as the total cache occupancy for an interface i (denoted S_i) is below S_i^{high}, the interface stays in S&D state. If the outgoing interface rate is higher than the incoming rate, the cache eventually drains and incoming data is now handled by the interface buffer (see $Push$ state above). Instead, if the occupancy of the interface cache increases beyond the upper-bound S_i^{high}, interface i switches to *Backpressure (B)* state.

3) Backpressure (B) State: The purpose of the Backpressure state is to reduce the occupancy of the cache and avoid packet drops. During this state, an interface i sends *"slow-down"* notifications (piggybacked to ACKs) to upstream nodes. Upon receiving a slow-down request, an upstream router R_U enters *closed-loop* operation on the flows that are heading to interface i *only* (path $R_U \rightarrow R \rightarrow R_D$ in Fig. 4). Flows to any other interface of R (*e.g.*, nodes R_1, R_2 in Fig. 4) are not affected. This is achieved by tagging the ACK packets of the affected flows—those traversing interface i—with a slow-down notification along with a nonce, which is effectively a unique alias name for interface i (possibly derived from i's IP address) that propagates upwards. The upstream nodes (node R_U in this case) store the nonce and forward the ACKs further upstream. The sender node echoes this nonce back to data packets of the respective flows, while downstream nodes cache packets with the nonce. Those upstream nodes (*e.g.*, R_U in our example) send one packet for each ACK packet with nonce received. As we explain later, the closed-loop operation moves progressively upstream; that is, the upstream nodes slow down only when their caches reach their upper threshold, S_i^{high}. Once the occupancy of i's cache at R drops below the lower-bound threshold S_n^{low}, interface i leaves the *Backpressure* state and switches to *Disable Backpressure* state.

4) Disable Backpressure (DB) State: Before an interface i in *Backpressure* state can switch back to *S&D*, it first enters a transition state named *Disable Backpressure*. During this state, R's interface i (*e.g.*, in Fig. 4) sends a "*cancel*" notification (again piggybacked with ACKs) along with the nonce corresponding to i to upstream nodes of R (*e.g.*, R_U in Fig. 4), who have been caching traffic heading downstream to i. The upstream nodes of R (*e.g.*, R_U) receiving the cancellation notice, erase the nonce from their list of locally stored nonces and stop caching nonce-carrying packets heading downstream to R. If during the DB state, the cache occupancy exceeds the upper-bound, then the interface switches back to Backpressure state.

NOTE: Our nonce-based backpressure mechanism assumes symmetric paths between data and ACKs. The path symmetry

requirement can easily be relaxed if data packets (instead of ACKs) are used for piggybacking of nonce and *"DB state cancel"* notifications. In this case, receiver end-points would echo the nonce to ACKs which would then be acted upon once they reached the sender. This process would require an extra $\sim 1/2 \cdot$ round-trip time (RTT) before sender can take any action.

Algorithm 1 Processing of ACK Packets at an Interface

```
1: function ACK-PROCESSING(Interface i, Ack Packet p)
2:     if i.state = B & S_i < S_i^low then
3:         i.state ← DB
4:         p.notification.append(cancel)
5:         set_expiration_timer
6:     else if S_i > S_n^high then
7:         i.state ← B
8:         p.notification.append(slow-down)
9:     else if S_i > 0 then
10:        i.state ← S&D
11:    else
12:        i.state ← P
13:    end if
14:    if p.notification = slow-down then
15:        ▷ Add the nonce to the stored-nonces set
16:        i.stored-nonces.add(p.nonce)
17:        Data Packet d ← Cache.get(p.flow)
18:        forward(d)        ▷ Send one data packet for the ack
19:        if i.state ≠ B then
20:            ▷ Don't propagate Closed Loop
21:            p.notification.clear(slow-down)
22:        end if
23:    else if p.notification = cancel then
24:        ▷ Remove the nonce from the stored-nonces set
25:        i.stored-nonces.remove(p.nonce)
26:    end if
27:    forward(p)
28: end function
```

B. Hop-by-Hop Flow(Let) States

In the context of INRPP, it is important to distinguish the router's *interface states* (discussed above) from the *flow's states*. Generally speaking, INRPP flows operate in either *open loop* or *closed-loop* mode. However, the fact that different parts of the flow are cached at different nodes along the path results in the situation where different parts of the flow (vaguely defined as flowlets) are in different states.

Similarly to TCP's slow-start, new flows enter the system in open-loop mode at the sender. Triggered by a slow-down notification originating from some backpressuring interface i along the path, flows that traverse i (denoted as F_i) enter a closed-loop mode *between the current and the immediate upstream node*, that is, not along the entire *e2e* path. The closed-loop operation extends further upstream when the cache of the immediate upstream node of the backpressuring router reaches its upper-bound cache threshold, S_i^{high}.

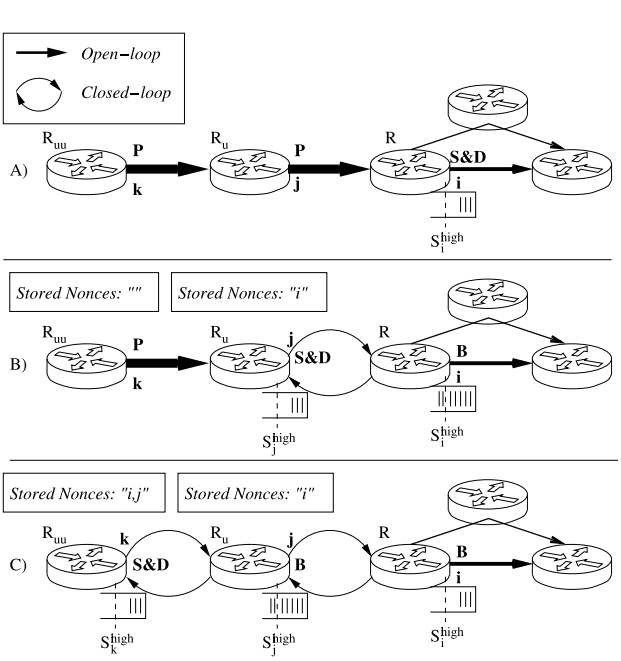


Fig. 5. Hop-by-hop propagation of flow state.

Consider the example in Fig. 5, where we demonstrate a sequence of state changes in three consecutive stages. Initially (see part A of Fig. 5), the upstream nodes of router R (R_u and R_{uu}) send traffic to its interface i in an open-loop (*i.e.*, Push) manner (entire flow is in open-loop mode). Eventually, the occupancy of interface i's cache (at R) exceeds S_i^{high} and interface j of upstream node R_u—upon receiving slow-down message— enters *S&D* state (see part B of Fig. 5), *i.e.*, R_u starts caching the data packets (carrying nonce) belonging to F_i at its interface j, and sends one data packet for one ACK received for the flows in F_i. Note that at this point, interface j is in *S&D* mode, while flow(s) F_i are in closed-loop mode downstream from R_u's interface j and in open-loop mode upstream from R_u's interface j. In other words, the upstream node of R_u (R_{uu}) continues to send packets of F_i in an open-loop manner because R_u's j interface cache has not reached its upper-bound threshold, S_i^{high}, yet. To achieve this, R_u does not propagate the slow-down notification upwards and keeps only the nonce (corresponding to i at R) in the ACK packets. The ACK packets received by R_u and R_{uu} are shown above these routers in Fig. 5.

Once the cache occupancy of interface j of R_u also exceeds the upper-bound S_j^{high}, interface j enters Backpressure mode too, and sends slow-down messages upstream to R_{uu} to slow-down the traffic flowing through j. At this point, the *flow's* closed-loop mode extends to R_{uu} as shown in part C of Fig. 5.

The hop-by-hop flow states and their propagation is essential to avoid packet drops from INRPP caches. A tightly coupled requirement to the propagation of flow states is appropriately setting the upper-bound threshold on the interface cache occupancy, S_i^{high}. In particular, the upper-bound threshold should be set in a way that allows enough time to notify upstream nodes to slow down without overflowing the cache.

TABLE II
INRPP DESIGN NOTATION

Symbol	Definition
S_i^{total}	Cache size at interface i
S_i	Cache occupancy at interface i
$S_i^{high/low}$	Upper-/Lower-bound occupancy threshold of the cache at interface i
C_n^i	Link i capacity at node n
T_n^i	Link i traffic at node n
F_i	Flows traversing interface i
T_p	Interval of probing packets
T_d	Interval of decision on detour path sending rates

The extreme case where the slow-down notification has to propagate all the way to the sender leads us to set this threshold to $S_i^{high} = Size(Cache_i) - RTT \times C_n^i$. On the other hand, the value of lower-bound threshold on the cache occupancy (S_i^{low}) is less critical, as it only affects the duration of the backpressure state. In the evaluation section, we set this threshold to half of S_i^{high}. A sensitivity analysis is provided in Section IV-E to evaluate the impact of the upper- and lower-bound thresholds on the performance.

The ACK packet processing behavior of INRPP routers is shown in Algorithm 1. In lines 2-13, the interface i state of the router is set depending on the occupancy of the interface cache following the discussion in Section II-A. In case the ACK packet contains a slow-down notification, then the flow corresponding to the ACK packet enters the closed-loop operation, and the nonce in the packet is stored locally (line 15). As part of the closed-loop operation, the router forwards exactly one data packet from the cache for the corresponding closed-loop flow (lines 17-18). The closed-loop is not propagated upstream (*i.e.*, slow-down notification is cleared from the ACK packet) in case the interface is not in Backpressure state; otherwise, the notification stays in the ACK packet and propagated upstream (lines 19-22). On the other hand, if the ACK packet contains a Cancel notification, then the nonce is removed from the local storage (lines 23-25).

III. INRPP IMPLEMENTATION DETAILS

In this section, we provide implementation details of the several mechanisms comprising the In-Network Resource Pooling Protocol. In Section III-A, we discuss computation of detour paths and the feedback mechanism to determine the *fair share* of spare capacity on detour paths. Then, in Section III-B, we explain how the feedbacks are used to achieve a max-min fair allocation in a stable way among routers sharing the same detour paths. Afterwards, in Section III-C, we discuss the changes required at the TCP agents at the end-points. Finally, in Section III-D, we present the details of the INRPP cache system.

A. Detour Path Information Signalling

In INRPP, a node detours its traffic through alternative paths to eliminate the excess traffic stored in its cache. In order to avoid severe detour delays and packet reordering, nodes refrain from using already congested detour paths as sending traffic to a congested interface would result in caching of the detour traffic along the detour path. For the purposes of this paper we assume that flows follow 1-hop detours only, *i.e.*, no 2- or more-hop detours are considered (see Table I).

INRPP makes use of a simple link-state protocol in order for each router to identify its 1-hop detour neighbours. In Fig. 2, node R_1 determines that its next-hop neighbours R_3 and R_2 are directly connected by examining the local routing information (*i.e.*, link state advertisements). Router R_1's detour lookup table for the topology of Fig. 2 is shown in Table III (interfaces i_1 and i_2 detour traffic for each other and i_3 has no detour interfaces). In order to make sure the detoured traffic eventually reaches the intended next-hop, routers tag their detoured packets with their original next-hop IP address. The link-state protocol would be slightly more complicated for multi-hop detour paths.

Going back to Fig. 2, when R_1's interface facing R_2 (labeled i_1) is congested (*i.e.*, i_1's buffer overflows), the interface switches to S&D state. At this point, R_1 starts caching excess data and demultiplexes packets between the primary and the detour interfaces. A (1-hop) detour path has spare capacity, only if both of the two (egress) interfaces along the path are operating in the *Push* mode. Therefore, R_1 uses the detour path (dashed line in Fig. 2), only if its own interface i_2 *and* interface i_1 of R_3 are both in *Push* mode.

In order to determine the amount of traffic to send on detour paths, routers periodically (every T_p msec) send *probe packets*[3] to each of their immediate neighbors along the detour paths. Upon receipt of a probe packet, a neighboring router returns the probe packet back to the sender with a detour path-specific "rate feedback". A rate feedback for a detour path is used by a detour interface to revise its aggregate sending rate (for out-going detour traffic) along the corresponding detour path. As an example, consider Fig. 2 where the router R_1 sends a probe packet to obtain a rate feedback for its detour path ($R_1 \rightarrow R_3 \rightarrow R_2$) to R_2. The "detour router" R_3 continuously monitors the utilisation on its interface i_1 and returns the probe packet back to R_1 with a rate feedback. Then, R_1 combines the feedback from R_3 with the knowledge of local interface i_2's spare capacity and eventually makes a decision on the sending rate for its aggregate detour traffic over the detour path to R_2 (see Section III-B for details).

In general, the detour interfaces periodically (every T_d msec) make a decision on their (revised) sending rates of detour traffic based on the feedback they obtain from their neighbors. For stability reasons, the decision time-scales are significantly smaller than the probing time scales: existing traffic engineering solutions [14] have shown that both conditions: $T_d \geq 5T_p$ and $T_p >$ "RTT of the probed path" are necessary to obtain stability. We use $T_d = 50msec$ and $T_p = 10msec$ (10msec is greater than the round-trip time of the one-hop detour paths in the experimented topologies) in our evaluations.

[3]Similar to ICMP packets, probes are control packets that require slow data path processing at the routers.

TABLE III
DETOUR INTERFACE LOOKUP, ROUTER R_1, FIG. 2

Primary Outgoing Interface	Detour Interface(s)
1	2
2	1
3	Null

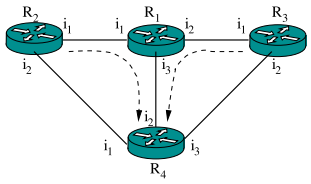


Fig. 6. Detour Example: The routers R_2 and R_3 use detour paths through router R_1 to reach R_4 (dashed lines).

B. Stable Allocation of Spare Capacity

When the spare capacity of a link is shared by multiple detour paths—*e.g.*, the link R_1-R_4 in Fig. 6 is used by two detour paths: $R_2 \rightarrow R_1 \rightarrow R_4$ and $R_3 \rightarrow R_1 \rightarrow R_4$ (see dashed lines)—, the routers sending detour traffic must coordinate their sending rates along the detour path. This is because the uncoordinated decisions of the senders (*e.g.*, R_2 and R_3 in Fig. 6) can lead to not only congestion but also persistent oscillations. As an example to persistent oscillations, consider the case when the link R_1-R_4 in Fig. 6 is underutilized and both R_2 and R_3 decide to shift their traffic to the link. Because both routers move their traffic to the common detour path without any coordination, the total traffic on link R_1-R_4 becomes larger than the link can handle. This causes both routers to eventually move traffic away from their detour paths, resulting with under-utilization of the shared link R_1-R_4; this cycle repeats causing persistent oscillation in the rate of traffic on the shared link.

Using the periodic feedback from their detour router, the detour interfaces iteratively revise their sending rates along the detour path. The objective is to achieve a fair allocation of spare capacity along a detour path by the detour interfaces. The iterative rate adaptation performs *an additive increase, multiplicative decrease* of sending rates depending on the amount of spare capacity on the shared detour path: in case of available spare capacity, the detour interfaces perform additive increase to further allocate an equal share of the spare capacity for their detour traffic; otherwise, the detour interfaces perform multiplicative reduction of their rates proportional to their current sending rates.

The spare capacity of an interface i is computed as follows:

$$SC^i = \alpha \cdot T_p \cdot RB^i - \beta \cdot Q^i, \tag{1}$$

where β and α are constants, RB^i is the average residual bandwidth on the interface (*i.e.*, capacity minus load) calculated using Exponential-Weighted Moving Average (EWMA) over the last few (we use the last five intervals in the evaluation) T_p msec measurement intervals and Q^i is the length of interface i's buffer queue at the end of the current interval (*i.e.*, at the time of feedback computation).

Following the work of Kandula *et al.* [14], a router R computes and returns a pair of Δ values in response to a probe packet for its interface i as follows:

$$\begin{aligned} &\text{if } SC_R^i > 0, \quad \Delta^+ = \frac{SC_R^i}{N},\ \Delta^- = 0 \\ &\text{if } SC_R^i \leq 0, \quad \Delta^- = \frac{SC_R^i}{L^i},\ \Delta^+ = 0 \end{aligned} \tag{2}$$

where SC_R^i is the calculated spare capacity of interface i (using Equation 1), N is the number of routers contending for the detour path (*e.g.*, N = 2 for R_1 in Fig. 6), and L^i is the EWMA of the aggregate load on the interface calculated using the aggregate load values measured during each of the last few T_p msec intervals (we use the last five intervals in the evaluation, consistent with the SC_R^i calculation). In case of spare capacity at the interface i, the Equation 2 returns a non-zero Δ^+ which is the equal share of spare capacity in terms of number of bytes that can be delivered over a T_p msec interval, *i.e.*, the additive rate increment in bytes per second. On the other hand, in case of no spare capacity, the Equation 2 returns a unitless ratio (Δ^-) that is applied for multiplicative decrease of the current rate as we explain below.

Upon receipt of a Δ pair, a detour interface revises its current rate of out-going detour traffic r_d^{curr} to compute a new rate r_d^{new} as follows:

$$r_d^{new} = r_d^{curr} + \Delta^+ - r_d^{curr} \cdot \Delta^-, \tag{3}$$

where the Δ^+ is the additive increase component, while $\Delta^- \cdot r_d^{curr}$ is the multiplicative decrease component. It has been shown in [14] that stable allocation of rates requires α and β constants in spare capacity calculation (SC in Equation 1) to be assigned as follows: $0 < \alpha < \frac{\pi}{4\sqrt{2}}$ and $\beta = \alpha^2\sqrt{2}$.

C. INRPP End-Points

Although TCP itself can be adjusted to exploit multiple paths simultaneously [11], the addition of in-network storage necessitates a rate-based transmission pattern. Recent studies on the impact of buffer-bloat on TCP's performance [15] show that an AIMD-based transmission pattern cannot cope with large in-network storage. Such environments cause TCP to open its window beyond what the network can handle and eventually end up in consecutive timeout events and severe unfairness. We have therefore decided to use a modified, rate-based version of TCP.

Our resulting INRPP end-point client complies with the feedback signals received from notifications (*e.g.*, *slow-down*) in the ACK packets. The fast-recovery mechanism at the sender is disabled; however, the timeout-based retransmission mechanism is kept: even though INRPP routers no longer experience congestion-related packet drops, packets will inevitably need to be retransmitted in case of packet corruption.

As detailed in Section II-B, INRPP flows operate in either open-loop or closed-loop state. The INRPP sender initially transmits in an open-loop manner and forwards packets according to processor sharing. When the INRPP sender receives a nonce within an ACK packet, it stores the nonce

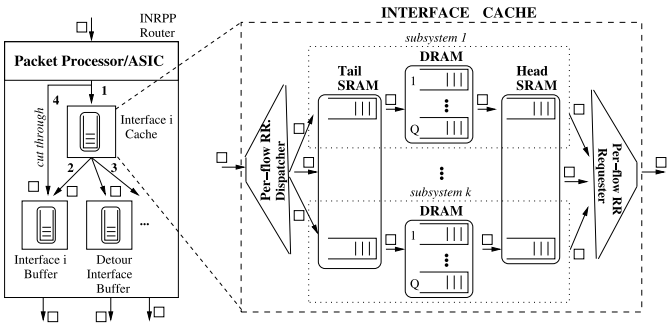


Fig. 7. Router overview (left), cache system (right).

and copies it to the future data packets of this flow. The nonces as well as the *slow-down* and *cancel* notifications are stored within INRPP's packet option fields. When an INRPP sender receives a *slow-down* request, it adapts the sending rate of the specific flow to the rate of acknowledgement packets (*i.e.*, closed loop operation). On the other hand, when an INRPP sender receives a *"cancel"* notification through an ACK packet, it switches back to open-loop mode and sends at the rate available through its outgoing interface. In case of no reception of ACKs during specific time intervals at INRPP end-points—for instance, no ACK arrivals for an entire TCP retransmission timeout (RTO) period (calculated based on recently observed RTTs)—,flows will automatically switch back to open-loop mode and send packets back at full rate.

The INRPP receiver does not require any modifications compared to a standard TCP receiver. It is worth noting, however, that INRPP receivers might see out-of-order packets, due to re-ordering. Although re-ordering might happen due to detouring of some parts of the flow, detour paths add very small extra delays (in the order of a few milliseconds as we show in our extensive evaluation), given they are only 1-hop detours. Also, note that INRPP avoids packet loss and lengthy retransmissions, which are normally the main reason of reordering and cause Head-of-Line (HoL) blocking. That said, reordering in our case differs from the more complicated cases where re-ordered packets experience substantially longer delays.

D. INRPP Cache System Design

An INRPP router contains *per-interface caches* as shown on the left hand-side of Fig. 7. Each interface cache is a bloated buffer with a storage capacity of few (*e.g.*, tens) seconds of traffic that store packets overflowing from the corresponding interface's queue (buffer). When the cache is empty, the incoming packets (arrow 1 in Fig. 7) follows the *cut-through* path (arrow 4) directly to the default (*i.e.*, determined by forwarding lookup) outgoing interface buffer. Cached packets on the other hand, are forwarded either through the default outgoing interface, or through the *detour interface(s)* (arrows 2,3)—that is, the interfaces facing the first link of the detour paths—depending on the availability of residual bandwidth on the detour paths (see Section II-A).

Because existing memory technologies present a speed (*i.e.*, access time) vs. storage cost trade-off, (see Table IV),

TABLE IV
ACCESS TIME AND COST DETAILS OF *RAM

Technology	Access (ns)	Capacity	Cost ($/MB)
SRAM	0.45	260 MB	27
DRAM	11.25	8 GB	0.054

designing a cost-effective interface cache is an important issue. While only SRAM's access time of $0.45ns$ is sufficient to accommodate read/write speeds of 100 Gbps for 64-byte data (which requires 2.56 ns latency), a cache that is solely made from SRAMs would be very expensive. However, recent research has considered cost-effective hybrid memory architectures combining slow DRAMs and fast SRAMs together in a hierarchical manner [16]–[18]. Hybrid memory architectures can achieve the overall access speed of the fast SRAM, while keeping the per-bit storage cost close to the DRAM's cost. This is achieved by "parallelizing" the high throughput of SRAMs through multiple slow DRAMs as shown in the right hand side of Fig. 7.

In Fig. 7, multiple slow DRAMs are "sandwiched" between a single tail and head SRAMs. The purpose of the tail SRAM is to store incoming data at line rate, and similarly the purpose of head SRAM is to maintain line-rate data flow between the cache and the outgoing interface buffer. Based on the access speed ratios between state-of-the-art DRAM and SRAM, around 25 DRAMs are sufficient to achieve an overall linespeed storage. This leads to a storage size of 200GB with 8GB DRAMS, which corresponds to 16 seconds of traffic. The DRAMs cost around $\$1350$ for each cache, which is reasonable for high-end routers.

Overall, a hybrid cache design is cost-effective for high-speed interfaces and leads to $\approx 99\%$ cost savings in comparison to similar-sized cache systems built only of SRAM. Also, the trend of decrease in per-bit storage prices in DRAM memories would make packet caches with hybrid architecture more cost effective in the future. For further details on hybrid memory architectures, we refer the reader to [18].

IV. PERFORMANCE EVALUATION

In this section, we provide a detailed investigation of the performance of the proposed INRPP scheme, and we compare its performance with TCP, RCP and MPTCP for various topologies and network conditions. We implemented INRPP in ns-3 [19], ported the existing ns-2 implementation of MPTCP to ns-3 and used RCP's existing implementation for ns-3 [20]. We use the New Reno version of TCP.

For our evaluation, we use the following three scenarios: *i) Dumbbell Topology with a detour path*, where the purpose is to demonstrate the operation of the INRPP mechanisms in a simple setup, *ii) Multihomed Topology*, where we compare the performance of INRPP with other transport protocols (mainly the MPTCP) when the senders are multihomed, *iii) Rocketfuel Topology*, where the purpose is to evaluate INRPP under a realistic workload using a realistic (*i.e.*, ISP) topology.[4]

[4]The code and scripts to reproduce the results will be made publicly available.

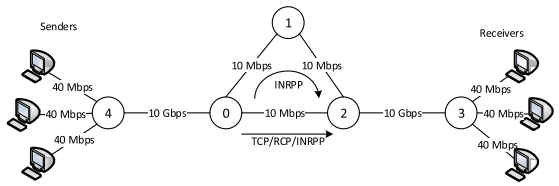


Fig. 8. Detour simple scenario.

A. Dumbbell Topology With a Detour Path

1) Scenario Description: We first evaluate a simple scenario using a dumbbell topology with a fully-connected core component (nodes 0, 1, and 2), depicted in Figure 8. The purpose is to show in detail the operation of the different INRPP mechanisms in a simple setup. This topology has a bottleneck link (link 0-2) of 10Mbps, but it also has another 10Mbps capacity one-hop detour path (link 0-1-2) that can be exploited to complement the bandwidth available at the bottleneck. Hosts are connected with 40Mbps links, and links 4-0 and 2-3 have more capacity than the rest of the links. We pair the senders (three hosts on the left) and the receivers (three hosts on the right), and each sender initiates a single flow to its receiver pair. The three flows enter the system one second apart from each other. Each flow has the same size of 10 MB, and the size of each router cache is set to only 1.25 MB (1s worth of traffic at 1Mbps) in order to demonstrate the operation of the backpressure mechanism, which eventually propagates to the sender. The cache occupancy upper and lower bounds S_n^{high} and S_n^{low} are set to 1 MB and 500 KB, respectively. Packet size is set to 1500 bytes. Router interfaces buffer size is set to 50 msec [21] worth of traffic using Drop Tail and link latencies are 5 msec. For RCP simulations we used the following parameters: $\alpha = 0.1$ and $\beta = 1.0$. We simulated the scenario using INRPP, RCP and TCP. In this scenario, we do not evaluate MPTCP since hosts are not multihomed, and therefore there is no possibility of establishing multiple subflows between peers.

2) Results: AFCT: In Table V, we present the average flow completion time (AFCT) [22]. INRPP completes flows $\approx 50\%$ faster than TCP and $\approx 60\%$ faster than RCP. This is because INRPP is able to exploit all the available bandwidth in the bottleneck and the detour paths. On the other hand, TCP and RCP are only utilising the bottleneck link capacity.

Goodput: In Figure 9, we demonstrate the goodput at the receiver in bps. With INRPP (top figure), we observe that the bandwidth is shared equally between the existing flows and there is no fluctuation when a new flow arrives. Particularly, the first flow starts at second 1 and is transmitted at 20Mbps using both the bottleneck link and the detour path shown with an arrow in Figure 8. When the second flow starts at second 2, the capacity is immediately shared between the two active flows (10Mbps each), while when the third flow starts at 3 seconds the available bandwidth is immediately split between the three active flows (≈ 6.66 Mbps per flow). When flows start completing, the existing flows adapt their rate and share the bandwidth no longer used by the completed flow. With TCP (bottom figure), on the other hand, we can observe that the goodput at the receiver is erratic and fluctuates excessively. TCP shares the bandwidth equally; however, it needs time to adapt to the new flow arrivals. Even after all the flows begin, the goodput oscillates continuously throughout the simulation due to the saw-tooth behaviour of the congestion window of TCP. In contrast, RCP does not have such oscillation in goodput, but we see that RCP also requires time to adapt and share the bandwidth between flows efficiently. RCP's slow adaptation to arriving flows is more pronounced due to the small number of flows: RCP shares the bandwidth among flows by estimating the number of active flows, and when the number of active flows is small, this estimation is less accurate. The slow adaptation of RCP to arriving flows leads to worse AFCT than TCP in this scenario.

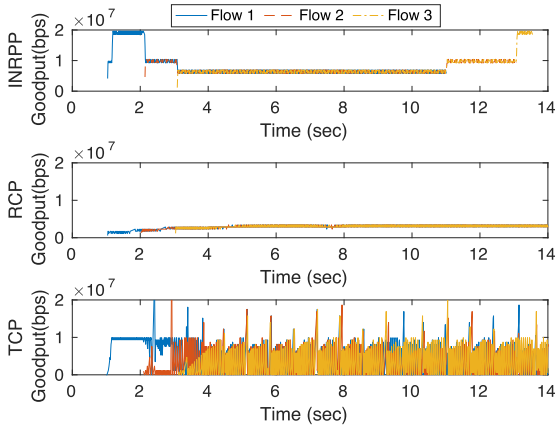


Fig. 9. Receiver goodput.

TABLE V
DUMBBELL TOPOLOGY SIMULATION RESULTS

Protocol	AFCT	Fairness	Fairness different RTTs
INRPP	10.50	0.9945	0.9972
RCP	25.96	0.9934	0.9994
TCP	23.31	0.8902	0.8321

Fairness: In Table V, we also present fairness results, using the Jain's index [23]. In order to demonstrate that INRPP is not affected by diverse RTT paths [24], we also evaluated fairness for heterogeneous access link latencies: 50, 100 and 200 msec. With both homogeneous and heterogeneous RTTs, we observe that INRPP and RCP fairness remains close to the optimal, due to their rate-based transmission pattern. TCP, on the other hand, presents the worst fairness performance, especially under heterogeneous RTTs, due to its erratic throughput oscillations.

Cache Occupancy: In Figure 10, we present the cache occupancy, data input and output rates of nodes 0 and 4. The purpose is to show INRPP's cache system behaviour in detail. The output data rate at node 0 is constant and equal to the capacity of the bottleneck link (0-2) *plus* the detour path capacity (2×10^7bps). That is, the interface of node 0 facing node 2 switches to *Store & Detour (S&D)* state from the initial *Push (P)* state shortly after the first flow starts. The input rate of node 4 initially decreases gradually when the senders of the first and the second flow complete their transmission, marked

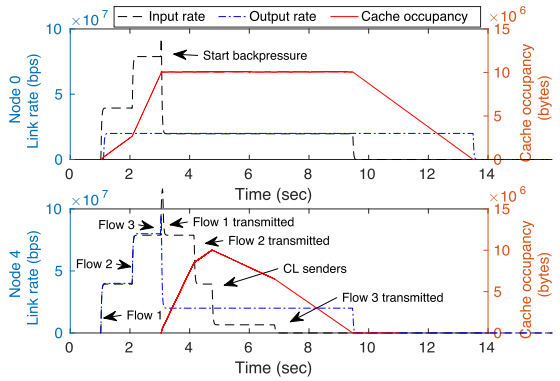


Fig. 10. Cache occupancy and data input/output rates (node 0 - top, node 4 - bottom).

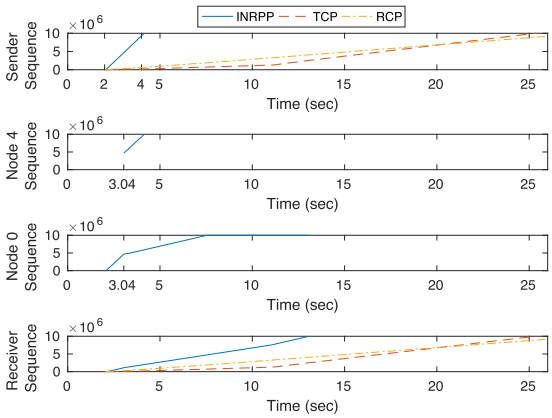


Fig. 11. Flows sequence number transmission.

in the bottom plot with "Flow 1 transmitted" and "Flow 2 transmitted", respectively. After this point, only the sender of flow 3 is still transmitting, and at around time 5s, the cache occupancy exceeds the upper-bound, which causes node 4 to send a *slow-down* notification to the sender of flow 3. This causes the sender of flow 3 to enter closed-loop (CL) mode, and therefore, slow down around time 5s as shown in the bottom plot of the figure with the marker "CL senders". During CL, the incoming rate of node 4 reduces to the rate at which the packets of flow 3 leave the cache of the bottleneck node 0, *i.e.*, 20Mbps/3$\approx$ 6.66 Mbps, since the cache in node 0 contains data from all sources and shares its bandwidth equally among the flows.

Flow of Packets: In Figure 11 we depict the sequence number of the segments flowing through different locations of the network for a single flow. The first plot of the figure is the sequence number of the segments leaving the sender, where we observe how differently a flow is transmitted from an INRPP sender compared to TCP or RCP senders. In particular, in only $2secs$ the INRPP sender transmits the entire flow data of 10 MB at rate of 40 Mbps. TCP and RCP senders are inserting data into the network in a closed-loop manner and are therefore transmitting for the whole duration of the flow, irrespectively of the available resources in the network. The second plot in the figure is the sequence number of the segments that are arriving at the cache of node 4. Here we can see that only the second half of the flow is cached in node 4; node 4 starts caching packets when it receives a *slow-down* request from node 0, which happens later (at $3.04s$) after transmitting approximately half of the flow's data without caching. The third plot is the sequence number of the segments that are cached in node 0. In the beginning, the flow is cached at a higher rate, *i.e.*, slope is steeper in the third plot until time $3.04s$. This is because node 0 is caching packets, and it is not in *Backpressure* state; therefore, the previous node (node 4) is transmitting at full rate. The bottom plot shows the sequence of segments arriving at the receiver host. At time $3.04s$, node 0 gets into *Backpressure* state and sends a *slow down message* to node 4. Node 0 implicitly performs ACK pacing as it sends ACKs at the rate of the bottleneck link ($\approx$ 10Mbps). At this point, node 4 sets all the flows to closed-loop mode; consequently, it starts caching incoming packets and sends one data packet for every ACK received. Node 4's output rate, *i.e.*, the input rate of node 0 shown in the third plot, immediately becomes equal to the rate at which the receiver host receives the flow. This can be observed by the similarity in the steepness of the slopes in plot 3 after time $3.04s$ with the slope of plot 4 after that time.

Cache Occupancy: Figure 12 shows the Total Cache occupancy, as well as the contribution of each flow to the cache at node 0 (top) and node 4 (bottom). In this figure, we can observe that the contribution (in terms of data) of each flow to the cache (sections of the flow plots with positive/increasing slope) and the flushing rate of the data belonging to each flow (sections of the flow plots with negative/decreasing slope) is the same. The latter demonstrates INRPP's fairness property as all flows are transmitted at the same rate. The top plot shows the state of node 0's interface (facing node 2 in Fig. 8) along with the cache occupancy. When the interface is backpressuring the previous node (*i.e.*, node 4 in Fig. 8), the cache occupancy of node 0 is maintained in the upper-bound since the caching and the flushing rates are equal. The gradient in the slope of the cache occupancy of a flow increases as other flows complete. For example, the cache occupancy by flow 3 increases when flow 2 completes at time $\approx 8s$ in the top plot. The bottom plot in Fig. 12 shows the cache occupancy at node 4 after various events. Caching at node 4 starts at $3.04s$ when the node receives the *slow down* message from node 0. At time $\approx 4s$, the arrival rate of flow 2 becomes 0 and the same happens later around time $\approx 5s$ for flow 3.

B. Multihomed Topology

1) Scenario Description: In this second scenario, we aim to evaluate INRPP in a multihomed scenario where we can compare it with multipath transports, such as MPTCP. MPTCP can exploit more than one *e2e* path and establish multiple subflows to take advantage of multihomed senders. It is, however, constrained to end-host multihoming and can therefore, take advantage of *e2e* paths only. We evaluate the scenario depicted in Figure 13 where two paths can be used in parallel when using MPTCP (link 0-1 and link 2-1). MPTCP senders are multihomed and are connected to node 0 and 2 at the

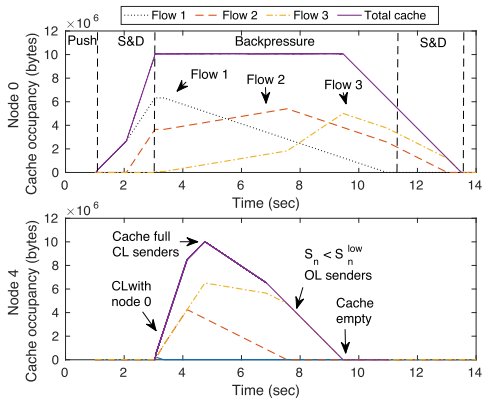


Fig. 12. INRPP cache occupancy.

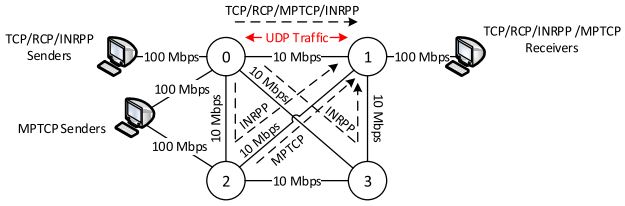
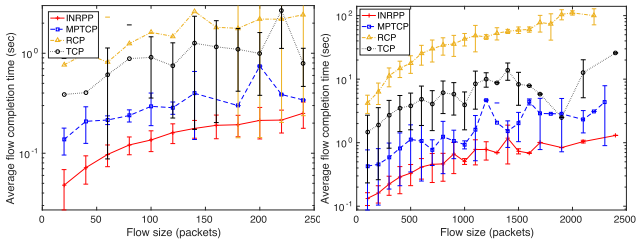


Fig. 13. Multihomed topology scenario.

same time, while the MPTCP receivers are singlehomed and connected to node 1. MPTCP users establish two subflows, one for each pair of *source-destination* IP addresses. In addition, we also evaluate INRPP/TCP/RCP connecting the senders to node 0 and the receivers to node 1 only. This way, TCP and RCP will use only the path across link 0-1 (the shortest path) and INRPP will use this path as the main option, but will also be able to use the detour paths 0-3-1 and 0-2-1. The network parameters are the same as in Section IV-A, however, here, we increase the cache size to 12.5MB (size proportional to 10 seconds worth of traffic); the end-points have 100 Mbps links.

We applied a Poisson Pareto Burst Process (PPBP) [25] to model Internet traffic. We used 1000 poisson-arriving flows with a λ rate determined by the offered load of the network ρ, where $\rho = \lambda \times E[L]/C$ ($E[L]$ is the average flow size in MB and C is the capacity of the link in Mbps), that we set to 0.9. Flow sizes are pareto distributed with shape equal to 1.2. Note that this scenario is beneficial for MPTCP due to symmetric bandwidth and/or latency links. MPTCP behaviour declines in case of highly asymmetric paths (in terms of bandwidth or latency, *e.g.*, 3G and WiFi) due to packet reordering [26], [27].

2) Results: In Fig. 14, we present results using $E[L]$ 30 and 500 packets respectively. We observe that the AFCT of both MPTCP and INRPP is much shorter than TCP or RCP for both cases. RCP still does not outperform TCP because the number of active flows is still not large enough for RCP to reduce its error rate in estimating the number of active flows. The performance of RCP and TCP is substantially inferior compared to MPTCP and INRPP, because neither of them can use more than one path to send data. When comparing MPTCP to INRPP we can see that INRPP clearly outperforms MPTCP,



(a) AFCT for $E[L] = 30$ (b) AFCT for $E[L] = 500$

Fig. 14. Average flow completion time in the multihomed topology.

TABLE VI
PROTOCOL FAIRNESS - TOPOLOGY FIG. 13

	$E[L] = 30$	$E[L] = 500$
TCP	0.8205	0.8575
RCP	0.9301	0.9589
MPTCP	0.6103	0.8947
INRPP	0.9298	0.9895

providing shorter flow completion times - up to around 50% in some cases (note log-scale y-axis in Fig. 14(a)). The reason for this is twofold. First of all, MPTCP can use more resources than TCP, but at the same time it also inherits its limitations, *i.e.*, being end-to-end. In particular, because MPTCP is an end-to-end resource pooling mechanism, it cannot exploit mid-path resources as INRPP does with the residual bandwidth available in detour paths (paths 0-3-1 and 0-2-1 in Fig. 13). Secondly, AIMD-based MPTCP faces drops and time-outs, that although do not necessarily impact significantly AFCTs, they cause poorer fairness performance - see Table VI. In fact, the chances of packet drops and timeouts in MPTCP increases linearly with the number of subflows. This is proved by the substantially worse fairness performance in case of short flows - up to 30% in case of $E[L] = 30$ (see first column in Table VI).

C. Hierarchical Topology

1) Scenario Description: In this scenario, we evaluate INRPP in a network with transit-stub hierarchy, where different sets of users are clustered in the edges, and edges connect each other through a highly connected core. Different edge nodes (E) in Fig. 15, are connected to a transit network consisting of core (C) routers. Some of the edge nodes in the topology are interconnected to other edge nodes to form a small stub network. Senders (servers) are concentrated in a branch of the topology and receivers are placed in the rest of the network, similarly to an ISP network connected to a data-center. All the links have the same capacity of 10 Mbps except the links connecting the edge nodes of the servers to the core nodes, which have 100 Mbps capacity. In this scenario, we can have multiple bottlenecks and multiple detour paths in the network. More importantly, this scenario presents a more challenging environment for INRPP because the traffic flowing on shortest paths occupy all the links in the topology as opposed to previous two scenarios where the detour paths were exclusively used by detour traffic.

2) Results: Each group of senders (top of the figure) connected to a single edge node consists of 3000 hosts, and

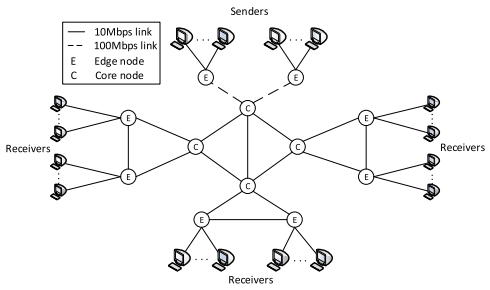


Fig. 15. Multi-domain topology scenario.

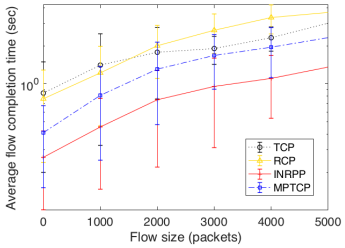


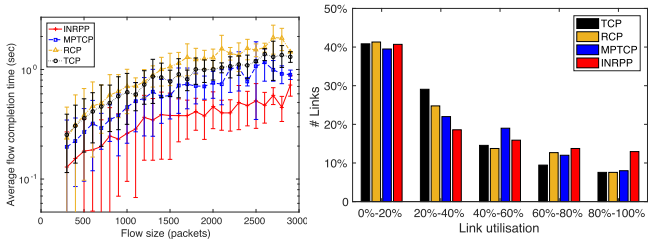
Fig. 16. AFCT for $E[L] = 125$.

each group of receivers (left, right and bottom of the figure) connected to a single edge node consists of 1000 hosts. We randomly pair each sender with a receiver and start a flow from each sender to its pair, which makes a total of 6000 flows. MPTCP end nodes are multihomed with two of the edge routers from the same domain. In this scenario, we start these 6000 flows with an offered load of the access link $\rho = 0.8$ and a $E[L] = 125$.

In Figure 16, we show the AFCT for varying flow sizes, using the same PPBP process, described in Section IV-B. We observe that even in this challenging scenario, INRPP clearly outperforms RCP and TCP by around 100% in terms of flow completion time and also MPTCP up to around 50%, similarly to Section IV-B. It does so because INRPP can take advantage of available residual bandwidth even when it is available for very short time-intervals, e.g., milliseconds, to detour excess traffic. We can conclude that dealing with congestion locally using INRPP is better than the e2e congestion control used by RCP, TCP or MPTCP given that the topology has detour paths (which is the common case - see Table I) and nodes possess caches.

D. Rocketfuel Topology

1) Scenario Description: In this section, we evaluate the performance of INRPP using a real-world topology. In particular, we use the 3257.pop.cch topology from the Rocketfuel dataset [13], which corresponds to the Tiscali network in Table I. Spanning over the European continent, the 3257.pop.ch topology has V = 440 routers and 681 bidirectional links. Out of the 440 routers, 267 are edge routers (with degree less than 3), 126 are gateway routers (*i.e.*, connected to an edge router, degree larger than 2), and 47 backbone routers. A total of 13350 senders and receivers are connected to the edge routers and flow arrivals are poisson-distributed.



(a) AFCT for $E[L] = 125$ (b) Link utilisation

Fig. 17. Performance evaluation for the rocketfuel topology.

We randomly pair each sender with a receiver and start a flow from each sender to its pair, which makes a total of 6675 flows. In this scenario, we start these 6675 flows with an offered load of the access link $\rho = 0.8$ and a $E[L] = 125$. All links have the same capacity of 1 Gbps except for the links connecting edge to gateway routers, which have 100 Mbps capacity, creating this way multiple bottlenecks and detour paths. MPTCP users are multihomed with two of the edge routers uniformly randomly distributed.

2) Results: In Figure 17(a), we show the AFCT for different flow sizes, using the same PPBP process, described in Section IV-B. INRPP clearly outperforms RCP and TCP by at least 50% (note log-scale y-axis) and MPTCP by at least 20% in terms of AFCT. INRPP takes advantage of available residual detour bandwidth to perform *mid-path multipath* even when it becomes available for very short time-intervals. Note that INRPP is not limited by equal-cost sub-paths as is the case for MPTCP. In fact, in our evaluation ECMP is enabled, and hence, all protocols take advantage of it by default. Furthermore, in-network storage enables data to progressively move to downstream caches and therefore avoid *e2e* rate-backoffs. MPTCP is not able to take advantage of bandwidth resources available in all sub-paths, as it is limited by its *e2e* design. We conclude that dealing with congestion *locally* (*i.e.*, without retransmissions) through temporary caching and detouring provides significant performance benefits: *i)* given the availability of detour paths in real network topologies (see Table I) and *ii)* assuming caching is available in network routers (see Section III-D).

In Figure 17(b), we show the percentage of links that fall into various ranges of utilisation levels. As shown earlier in Table I, 24.4% of links have 1-hop detours in the Tiscali topology; the lowest 1-hop detour path ratio among the ISPs in Rocketfuel dataset. We observe that around 40% of the links achieve significantly low utilisation (*i.e.*, in the range 0%-20%) even with INRPP, largely due to the limited number of flows (for scalability of the simulator) failing to utilise all the links in this large topology. On the other hand, the number of links with utilisation in the range 80%-100% are almost doubled with INRPP compared to TCP, RCP and MPTCP. Furthermore, the number of links with utilisation between 60% and 80% is increased. We observed larger gaps in utilisation levels between INRPP and other protocols for other topologies with larger number of detours, but we omit those for space considerations. In Table VII, we show the overall average

TABLE VII
AVERAGE LINK UTILISATION - ROCKETFUEL TOPOLOGY

	Average Link Utilisation (%)
TCP	25.02
RCP	24.41
MPTCP	33.18
INRPP	40.15

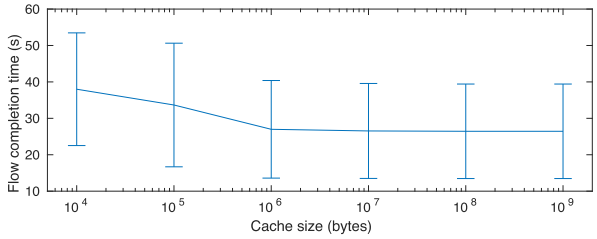


Fig. 18. Flow completion time using different cache sizes.

link utilisation in the simulation, and we observe that INRPP improves the overall utilisation by 50% compared with TCP and RCP, and by 20% compared with MPTCP. This is because INRPP achieves a large increase in the utilisation of backbone router links that have one or more 1-hop detours.

E. Cache Sensitivity Analysis

In this section, we perform a set of experiments to evaluate the sensitivity of INRPP to the size of the caches. In this set of experiments we evaluated the same scenario used in Section IV-A, depicted in Figure 8, but with 100 INRPP flows. The rest of the parameters are the same.

In Figure 18 we represent the flow completion time analysis as a function of the cache size configured in the routers. This cache size is the total cache space available per interface and we set the S_i^{high} threshold to half of the overall cache size and S_i^{low} threshold to the forth (i.e, for a 10MB cache size, the S_i^{high} is set to 5MB and the S_i^{low} to 2.5MB. From the figure we can observe that INRPP starts performing correctly from a cache size of at least 1MB for the evaluated scenario. Below this cache size INRPP caches drop packets, and therefore the performance is suboptimal, requiring retransmissions.

In the experiments, 1MB is the minimum cache size that avoids any packet drop in the cache. Packet drops may occur when the traffic received at incoming interfaces after activating backpressure is higher than the traffic sent through the congested interface, plus the cache space remaining between the S_i^{high} and the total cache size. Therefore, the margin between S_i^{high} and S_i^{total} should be at least higher than the difference between incoming and outgoing link capacities plus the higher link latency (time required to activate the backpressure to the previous router).

V. INRPP DEPLOYMENT

As we have demonstrated so far, INRPP provides benefits to users in terms of fast data transfer completion times and provides unbiased per-flow fairness. INRPP also provides benefits for network operators in terms of moving traffic from over-utilized to under-utilized links. Therefore, we believe that there are incentives for both users and network providers to deploy INRPP. What remains to be shown is how INRPP and TCP can coexist without compromising the performance, and how INRPP can be used in case of partial deployment, *i.e.*, only a subset of nodes implement INRPP. We also touch on the security of INRPP at the end of this section.

A. TCP Friendliness

In order to illustrate that the INRPP protocol can co-exist smoothly with TCP, we partition the per-interface cache of the INRPP routers into two; one for the INRPP flows and one for the TCP flows. As before, the INRPP cache system has drop-tail fair queuing. However, because TCP does not work well with large buffers, we limit the size of the TCP partition to 50 ms worth of traffic in the experiments. The INRPP cache multiplexes incoming packets using the protocol field and places each packet in the appropriate partition.

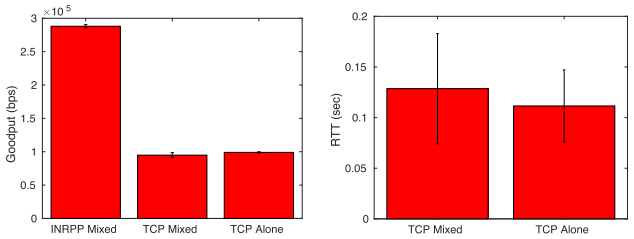
We evaluate our solution using the topology in Figure 8 using the following two scenarios: *i) Mixed Scenario*, where we have 50 INRPP flows and 50 TCP flows using INRPP routers, *ii) TCP Alone Scenario*, where we have only the 100 TCP flows using legacy IP routers (*i.e.*, INRPP cache system disabled) in the same topology. Obviously in both scenarios, the detour path (0-1-2) is not utilised by any of the TCP flows.

The results depicted in Figure 19 represent the average goodput for the INRPP and TCP flows in the scenario mixing TCP and INRPP, and the average flow goodput for the TCP alone scenario. We observe that INRPP and TCP flows share the bandwidth in a fair manner on the bottleneck link (0-2) of 10 Mbps. As shown in Fig. 19(a), each of the 50 TCP flows use half of the bandwidth along the bottleneck, which is around $5\ Mbps/50 = 100\ Kbps$ (depicted as "TCP mixed"). On the other hand, the INRPP flows use both their equal share along the bottleneck link and the entire bandwidth along the detour path. As a result, each of the 50 INRPP flows use around $15\ Mbps/50 = 300\ Kbps$ (depicted as "INRPP mixed"). The performance of TCP flows using legacy IP routers is depicted as "TCP Alone" in Fig. 19(a). We observe that the goodput is almost identical in the case of mixed and alone scenarios for TCP, which means that the TCP goodput is not affected adversely by coexistence of both protocols.

We also measure the RTT of the TCP flows for the "TCP mixed" and "TCP alone" scenarios. We observe that the RTT of the TCP flows in the "TCP mixed" scenario is only marginally higher compared to the "TCP alone" scenario (see Fig. 19(b)). This additional delay is due to the queue management overhead of the cache system, which is utilised by the TCP flows in the "TCP mixed" scenario. These results demonstrate that *TCP flows can co-exist smoothly with INRPP flows*.

B. Incremental Deployment

We envision deployment of INRPP within a single Autonomous System (AS) as ingress-to-egress transport for intra-domain traffic. Within a domain, INRPP can utilise bandwidth resources and alternate flows' operation between



(a) Average flow goodput (b) Average flow RTT

Fig. 19. TCP friendliness.

open and closed loop mode to shape traffic. However, when an ingress router's INRPP interface (facing another router in the same AS) cache fills up, it may be necessary to propagate closed-loop mode of flows beyond the edge of the AS and slow-down the flow's sender. A possible way to slow-down a sender's rate is through manipulating advertised TCP receive window size as done by middleboxes in the Internet today [28]. Specifically, an ingress router reduces the receive window size of ack packets in order to shape the throughput of senders in the opposite direction. The details of such an approach is to be further investigated and we leave it for future work.

Deploying INRPP in a multi-AS setting may be problematic at the AS borders, because AS routing policy violations may occur in case egress traffic are to be detoured at a border router of an AS. Nevertheless, this problem can be mitigated by disabling the detour mechanism at border nodes. We envision that initially INRPP can work as an overlay network under partial deployment. In this case, an overlay routing mechanism is required to compute forwarding tables for the INRPP nodes in order to forward INRPP packets along default or detour paths. To realise partial deployment, one possibility is to exploit the Information-centric networking (ICN) [1] architectures, which conveniently include large packet caches in their routers by design. For this, we consider the incrementally-deployable Hybrid ICN (hICN) [29] architecture of Cisco, which has already seen partial deployment in the current Internet. In the case of partial deployment, the INRPP nodes must ensure reliable transfer of data through the underlay network. This can be accomplished through different ways; one possibility is to establish TCP tunneling between the INRPP overlay nodes. A more detailed investigation of the partial deployment of INRPP is the subject of future work.

C. Security of INRPP

INRPP relies on explicit network feedback in the form of slow-down and cancel notifications piggybacked with ACK packets to guide the transmission rates. Non-cooperative nodes can ignore feedback to obtain unfair advantage over cooperative ones, which is a problem common to many transport protocols. We refer interested readers to the work of Wilson *et al.* [30] for a detailed discussion on attacks against fairness by uncooperative behaviour in a transport protocol that relies on explicit network feedback similar to INRPP.

At the same time, it is important to secure INRPP against malicious behaviour by the end-points that target either other end-points or the network infrastructure. In one attack vector, malicious end-points launch a Denial-of-Service (DoS) attack by injecting slow-down notifications to unnecessarily slow down on-going flows. However, in order for an injection attack to successfully slow down a target flow, the end-points of the flow must echo the nonces in the injected ACK packets to their data packets. For echoing to happen, the malicious nodes must somehow guess the correct sequence numbers of the target flows when injecting nonces. Therefore, the chance of success for an injection attack in INRPP is similar to the one in TCP.

Another type of attack involves spamming the network with bogus nonces as part of a state exhaustion attack targeted at the upstream routers' limited storage for nonces. To prevent the nonce storage from overflowing, routers can simply implement their nonce storage as a cache with an eviction policy such as Least-Recently-Used (LRU). However, cache storage alone does not solve the problem with poisoning of nonce caches with bogus nonces from malicious nodes. Currently, in the INRPP protocol, the endpoints do not originate slow-down notifications because the flow control is performed end-to-end. Therefore, the correct implementation of the protocol does not allow nonce origination from the end-points. Authenticating the notifications between directly neighboring routers would be sufficient to detect bogus nonces, and this can be achieved through a lightweight mechanism such as a MAC-based authentication [31]. We are not aware of other potential attacks against INRPP and leave a more detailed analysis of the protocol's security to future work.

VI. Related Work

The common practice among ISPs to move the bottleneck to the edge of the network (*i.e.*, DSLAM/GPON to user) restricts users from taking up as much bandwidth as possible. As we move towards a FTTx environment, however, the bottleneck will inevitably move to the core of the network potentially causing severe congestion events. Overprovisioning of core links will not be an option anymore and therefore alternative solutions will need to be sought. Multi-Path TCP [11] has received wide attention recently due to its ability to take advantage of multiple *e2e* paths. However, the requirement for *multihoming* of MPTCP (and mTCP [32]) have driven adoption of MPTCP to controlled, data-centre environments mainly [33]. INRPP extends the *Resource Pooling Principle* [12], [34], natively integrated in MPTCP, to also utilise *midpath multipath* without the *equal-cost* requirement of ECMP [35].

Multipath routing on the other hand has been studied in the context of traffic engineering in the core of the network [14], [36]–[38], mainly for load-balancing reasons [14], [35]. Despite extensive studies on multipath routing [39]–[41] and multipath congestion control [42]–[44], these two arguably complementary areas remain remarkably decoupled. There has been no previous attempt to combine the benefits of multipath routing and congestion control into a common framework in order to improve overall resource utilisation.

With the In-Network Resource Pooling Protocol we make a first attempt to bring the worlds of multipath congestion control and multipath routing closer together. Although much

of INRPP's mechanisms can be replaced or redesigned to fit to specific network needs, our detailed design and evaluation shows that the proposed set of mechanisms work smoothly together. At the same time, the significant performance gains observed prove the need for a combined multipath routing and congestion control framework.

VII. Summary and Conclusion

We have proposed a radically new way of circumventing the bottleneck caused by end-to-end transport techniques' limitations in wide area networks. The proposed scheme relies on in-network storage to progressively move data along the path from source to destination. Intermediate routers/caches act as custodians for received data, that is, they are not allowed to drop data. Therefore, packet losses due to congestion are eliminated and data migrates hop-by-hop in caches according to local resource availability (*i.e.*, faster than the *e2e* path's slowest link).

INRPP makes better use of in-network link resources and as a result achieves up to 50% faster data transfers. This is especially beneficial for short flows.

Our extensive performance evaluation shows that INRPP does not risk network stability since it gets in a closed-loop mode when network conditions deteriorate. When the network is less congested, INRPP takes immediate advantage of all the available bandwidth on both the main path and any detour available along the path and completes file transfers up to two times faster than conventional transport protocols. End-host clients need only minor modifications, while routers need to be equipped with caches and implement the detour and back-pressure mechanisms. The proposed hybrid SRAM-RLDRAM cache design keeps the cost down by mostly using cheap DRAMs and only very few expensive SRAMs per updated interface. We believe that, given the performance gains of INRPP, the required changes are not prohibitive.

References

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. ACM CoNEXT*, Dec. 2009, pp. 1–12.

[2] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[3] N. Gvozdiev, S. Vissicchio, B. Karp, and M. Handley, "Low-latency routing on mesh-like backbones," in *Proc. ACM HotNets*, New York, NY, USA, Nov. 2017, pp. 136–142.

[4] N. Feamster, "Revealing utilization at internet interconnection points," *CoRR*, vol. abs/1603.03656, pp. 1–10, Sep. 2016.

[5] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *Proc. IEEE INFOCOM*, vol. 4, Mar. 2004, pp. 2514–2524.

[6] N. Dukkipati, "Rate control protocol (RCP): Congestion control to make flows complete quickly," Ph.D. dissertation, Dept. Elect. Eng., Stanford Univ., Stanford, CA, USA, Oct. 2007.

[7] I. Psaras, L. Saino, and G. Pavlou, "Revisiting resource pooling: The case for in-network resource sharing," in *Proc. ACM HotNets*, Oct. 2014, pp. 1–7.

[8] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor sharing flows in the internet," in *Proc. IWQoS*, 2005, pp. 271–285.

[9] C. M. D. Pazos and M. Gerla, "A rate based back-pressure flow control for the internet," in *Proc. IFIP HPN*, 1998, pp. 555–573.

[10] S. Sarkar and L. Tassiulas, "Back pressure based multicast scheduling for fair bandwidth allocation," in *Proc. IEEE INFOCOM*, Apr. 2001, pp. 1123–1132.

[11] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proc. USENIX NSDI*, 2011, pp. 1–14.

[12] D. Wischik, M. Handley, and M. B. Braun, "The resource pooling principle," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 5, pp. 47–52, Sep. 2008.

[13] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with Rocketfuel," *IEEE/ACM Trans. Netw.*, vol. 12, no. 1, pp. 2–16, Feb. 2004.

[14] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the tightrope: Responsive yet stable traffic engineering," in *Proc. ACM SIGCOMM*, 2005, pp. 253–264.

[15] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the internet," *Commun. ACM*, vol. 55, no. 1, pp. 57–65, Jan. 2012.

[16] S. Iyer, R. R. Kompella, and N. McKeown, "Designing packet buffers for router linecards," *IEEE/ACM Trans. Netw.*, vol. 16, no. 3, pp. 705–717, Jun. 2008.

[17] J. Garcia-Vidal, M. March, L. Cerda, J. Corbal, and M. Valero, "A DRAM/SRAM memory scheme for fast packet buffers," *IEEE Trans. Comput.*, vol. 55, no. 5, pp. 588–602, May 2006.

[18] F. Wang, M. Hamdi, and J. K. Muppala, "Using parallel DRAM to scale router buffers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 5, pp. 710–724, May 2009.

[19] *NS-3, a Discrete Event Simulator*. Accessed:Nov. 20, 2021. [Online]. Available: http://www.nsnam.org

[20] *Kickass for NS-3—V.1.0*. Accessed:Nov. 20, 2021. [Online]. Available: http://users.eecs.northwestern.edu/~mef294/projects/kickass/code/README-kickass-ns3

[21] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 281–292, Aug. 2004.

[22] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 59–62, 2006.

[23] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," 1998, *arXiv:cs/9809099*.

[24] E. Gavaletz and J. Kaur, "Decomposing RTT-unfairness in transport protocols," in *Proc. LANMAN*, May 2010, pp. 1–6.

[25] M. Zukerman, T. Neame, and R. Addie, "Internet traffic modeling and future technology implications," in *Proc. INFOCOM*, vol. 1, Mar. 2003, pp. 587–596.

[26] J. R. Iyengar, P. D. Amer, and R. Stewart, "Receive buffer blocking in concurrent multipath transfer," in *Proc. IEEE GLOBECOM*, Nov./Dec. 2005, pp. –5.

[27] T. Dreibholz, M. Becke, E. P. Rathgeb, and M. Tüxen, "On the use of concurrent multipath transfer over asymmetric paths," in *Proc. GLOBECOM*, Dec. 2010, pp. 1–6.

[28] J. T. Araujo, R. Landa, R. G. Clegg, G. Pavlou, and K. Fukuda, "A longitudinal analysis of internet rate limitations," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 1438–1446.

[29] G. Carofiglio, L. Muscariello, J. Augé, M. Papalini, M. Sardara, and A. Compagno, "Enabling ICN in the internet protocol: Analysis and evaluation of the hybrid-ICN architecture," in *Proc. ACM ICN*, Sep. 2019, pp. 55–66.

[30] C. Wilson, C. Coakley, and B. Y. Zhao, "Fairness attacks in the explicit control protocol," in *Proc. 15th IEEE Int. Workshop Quality Service*, Jun. 2007, pp. 21–28.

[31] B. Raghavan and A. C. Snoeren, "A system for authenticated policy-compliant routing," in *Proc. ACM SIGCOMM*, 2004, pp. 167–178.

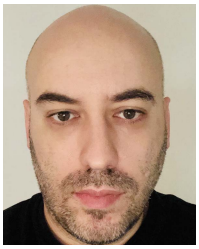
[32] M. Zhang *et al.*, "A transport layer approach for improving end-to-end performance and robustness using redundant paths," in *Proc. USENIX ATEC*, Jun. 2004, pp. 99–112.

[33] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM Conf.*, Aug. 2011, pp. 266–277.

[34] M. Honda, E. Balandina, P. Sarolahti, and L. Eggert, "Designing a resource pooling transport protocol," in *Proc. IEEE INFOCOM Workshops*, Apr. 2009, pp. 13–18.

[35] C. Hopps, *Analysis of an Equal-Cost Multi-Path Algorithm*, Standards IETF RFC 2992, 2000.

[36] J. He and J. Rexford, "Toward internet-wide multipath routing," *IEEE Netw.*, vol. 22, no. 2, pp. 16–21, Mar./Apr. 2008.
[37] X. Liu and L. Xiao, "A survey of multihoming technology in stub networks: Current research and open issues," *IEEE Netw.*, vol. 21, no. 3, pp. 32–40, May 2007.
[38] C. Lumezanu, D. Levin, and N. Spring, "PeerWise discovery and negotiation of faster paths," in *Proc. 6th ACM HotNets*, Nov. 2007, pp. 29–37.
[39] D. G. Andersen, A. C. Snoeren, and H. Balakrishnan, "Best-path vs. multi-path overlay routing," in *Proc. ACM SIGCOMM IMC*, 2003, pp. 91–100.
[40] R. Kokku, A. Bohra, S. Ganguly, and A. Venkataramani, "A multipath background network architecture," in *Proc. IEEE INFOCOM*, May 2007, pp. 1352–1360.
[41] S.-J. Lee, S. Banerjee, P. Sharma, P. Yalagandula, and S. Basu, "Bandwidth-aware routing in overlay networks," in *Proc. IEEE INFOCOM*, Apr. 2008, pp. 1732–1740.
[42] P. Key, L. Massoulié, and D. Towsley, "Path selection and multipath congestion control," *Commun. ACM*, vol. 54, no. 1, pp. 109–116, Jan. 2011.
[43] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec, "MPTCP is not Pareto-optimal: Performance issues and a possible solution," in *Proc. CoNEXT*, Aug. 2012, pp. 1651–1665.
[44] C. Raiciu, M. Handley, and D. Wischik, *Coupled Congestion Control for Multipath Transport Protocols*, Standards RFC 6356 (Experimental), Oct. 2011.



Sergi Rene received the B.S. degree in electrical engineering and the M.S. degree in network engineering from the Technical University of Catalonia (UPC), Barcelona, Spain, in 2006 and 2010, respectively, and the Ph.D. degree from the Department of Network Engineering, UPC, in 2015. In 2015, he joined the Department of Electronic Engineering, University College of London (UCL), carrying out research on information-centric networks, congestion control, and mobility.

Onur Ascigil received the Ph.D. degree from the Department of Computer Science, University of Kentucky, USA, in 2014. From 2015 to 2021, he worked as a Research Associate with University College London, U.K. He is currently a Lecturer with the School of Computing and Communications, Lancaster University, U.K. His research focuses on problems related to decentralized content and service delivery in the internet, resource management in edge/fog computing, future internet architectures, and congestion control.

Ioannis Psaras (Member, IEEE) is currently a Research Scientist at Protocol Labs. He is a part of the Resilient Networks Lab, where he is working on identifying and addressing future challenges that the IPFS, libp2p, and Filecoin protocols are bound to come across. He is particularly interested in content routing design and optimization, the performance of libp2p's pubsub protocol, and content naming, generally architectural extensions to support the resilience and growth of the IPFS and Filecoin networks. Before joining Protocol Labs, he has spent more than ten years in academia, the majority of which as a Lecturer at University College London, U.K., where he investigated resource management techniques for current and future networking architectures with a particular focus on "function-centric networks" to realize distributed and decentralized edge computing. He held a Prestigious EPSRC Early Career Fellowship in the area of "content-oriented and service-centric edge-computing architectures." He has received five best conference paper awards. He has also been heavily involved in the effort to shift the internet toward an information-centric networking environment.

George Pavlou (Fellow, IEEE) received the Diploma degree in engineering from the National Technical University of Athens, Greece, and the M.S. and Ph.D. degrees in computer science from University College London, U.K. He is currently a Professor of communication networks with the Department of Electronic and Electrical Engineering, University College London. His research interests focus on networking and network management. In 2017, he was elected as a Fellow of the IEEE for contributions to network resource management and content-based networking. In 2011, he received the IEEE/IFIP Dan Stokesbury Bi-Annual Award for "distinguished technical contributions to the growth of the network management field."