Journal of Network and Computer Applications 221 (2024) 103772

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

End-to-end active queue management with Named-Data Networking

Miguel Rodríguez-Pérez *, Sergio Herrería-Alonso, J. Carlos López-Ardao, Raúl F. Rodríguez-Rubio

atlanTTic research center, Universidade de Vigo, Maxwell s/n, 36310, Vigo, Spain

ARTICLE INFO

Keywords:
Information-centric networking
Named-data networking
Active queue management
Congestion control
CoDel

ABSTRACT

The innovative information-based Named-Data Networking (NDN) architecture provides a good opportunity to rethink many of the design decisions that are taken for granted in the Internet today. For example, active queue management (AQM) tasks have been traditionally implemented in the routers to alleviate network congestion before their buffers fill up. However, AQM operations could be performed on an end-to-end basis by taking advantage of NDN features. In this paper, we provide an implementation of an AQM algorithm for the NDN architecture that we use to drive a classical AIMD-based congestion control protocol at the receivers. To accomplish this, we take advantage of the 64-bit *Congestion Mark* field present in the link layer of NDN packets to encode both rate and delay information about each transmission queue along a network path. In order to make the solution scalable, this information is delivered stochastically, guaranteeing that receivers get accurate and updated information about every pertinent queue. This information is enough to implement the well-known controlled delay (CoDel) AQM algorithm. Simulation results show that our client-located CoDel implementation is able to react to congestion when the bottleneck queuing delay surpasses the 5 ms target set by the usual in-network CoDel implementation and, at the same time, get a fair and efficient share of the available transmission capacity.

1. Introduction

The current global network infrastructure, commonly known as the Internet, and fundamentally based on the TCP/IP architecture, has almost replaced all previous dedicated wide area networks. It has been claimed that its success comes from its simplicity and versatility, as it places almost no restrictions on both the underlying networking technology, nor in the switched data itself. In the end, applications using TCP/IP are provided with a point-to-point channel that can be used to transmit arbitrary information.

While this approach has served quite satisfactorily for the past forty years, it is not well suited for many of the new communication paradigms currently being used on the Internet. Nowadays, vast amounts of traffic belong not to point-to-point communications but to the transport of *information*. In these scenarios, the user is not really interested in establishing a communication with a particular server to then get some information from it. In fact, as long as the user can attest the information authenticity and integrity, the user does not care about its actual location on the network. The current use of load-balancers and overlaid caching networks is a band-aid solution intended to solve this mismatch between the network communication model and the needs of modern applications. This mismatch has its own serious shortcomings, such as the continuous recentralization of the network and, consequently, its diminishing reliability. Information-Centric Networks (ICN) (Kutscher et al., 2012), and, in particular, Named-Data Networks (NDN) (Zhang et al., 2014), provide a service where users can directly address *information* pieces, regardless of their location. This is more suited to the most common network uses, while it can still be employed to provide a conventional point-to-point communication model on top (Trossen et al., 2015). Additionally, some partially unsolved problems of the TCP/IP architecture—like multipath transmission, multicast, and mobility—are naturally supported by NDN. In the NDN architecture, applications request information from the network by sending an *Interest* packet that identifies the procured content. The network drives this packet towards a node that holds a (cached) copy of the *named* content or to an appropriate *producer*. Whatever the case, the content is sent back to the requesting application (*consumer*) as the payload of a *Data* packet.

Certainly, ICN networks, and NDN in particular, present both new opportunities and challenges for application traffic management. Among the challenges, a multi-producer setup, with information *chunks*

* Corresponding author.
E-mail addresses: miguel@det.uvigo.gal (M. Rodríguez-Pérez), sha@det.uvigo.es (S. Herrería-Alonso), jardao@det.uvigo.es (J. Carlos López-Ardao), rrubio@det.uvigo.es (R.F. Rodríguez-Rubio).

https://doi.org/10.1016/j.jnca.2023.103772
Received 2 May 2022; Received in revised form 19 June 2023; Accepted 17 October 2023
Available online 24 October 2023

disseminated across the network, hinders traditional congestion control algorithms that rely on relatively smooth round trip time (RTT) measurements and stable bottleneck locations. On the other hand, a new architecture is a good opportunity to try new approaches, like hop-by-hop congestion control or increased assistance by in-network devices. Additionally, as data is only entered into the network as a response to consumers' demands, it is the downstream equipment the one in charge of regulating the transmission rate. This is beneficial, as this equipment can receive and/or generate congestion information as it occurs in the network and react accordingly faster than a producer. These new capabilities can be used to address ongoing network problems with a different set of tools. For instance, in the last few years, the excessive amount of buffering done by most network devices, usually known as bufferbloat, has received ample attention from the research community as it is a source of excessive latency and subpar performance. A key ingredient of a comprehensive solution to this problem relies on implementing proper active queue management (AQM) inside the network, thus helping to (indirectly) drive the congestion control algorithms of the hosts. However, proper tuning of AQM is still an active research topic (Ye et al., 2021b), limiting its widespread usage.

1.1. Contributions

In this paper we present a novel approach for simultaneously performing both congestion control and AQM operations at the end hosts of an NDN network. This would enable faster deployment and experimentation with the tuning and development of new AQM algorithms.

For the approach to work, we designed a new mechanism that increases the congestion information that end hosts receive from the network. This mechanism takes advantage of the existing possibility for NDN routers to use up to 64 bits of information per packet to encode congestion information between connected devices. As eight bytes is a very tight space to encode accurate information about every queue in the network path, we designed a scalable stochastic algorithm that ensures that consumers get information from all the relevant queues. The information conveyed is the instantaneous throughput and the queuing delay, which should be enough for proper AQM and congestion control operations.

We show how the aforementioned information can be used to implement existing AQM algorithms at the end hosts. In particular, and in order to test the validity and accuracy of our scalable algorithm, we chose to provide a sample implementation of the controlled delay (CoDel) AQM algorithm. Then, we use the congestion information generated at the host by CoDel to drive a typical window-based TCP Reno-like congestion control algorithm. This algorithm modulates the Interest sending rate and, thus, indirectly, the data rate. Using Codel (Nichols et al., 2018) as a proof-of-concept also serves additional purposes. Firstly, it is a practical and modern AQM algorithm known to work quite well nowadays. So, it is important for us to prove that our approach allows for its client-side implementation. Additionally, it has a non-trivial state machine, making the client-side implementation both more interesting and challenging. We feel that a correct client-side implementation of CoDel is further proof of the validity of our approach.

Simulation results show that our CoDel implementation is able to limit the amount of traffic in the network queues while obtaining a fair and efficient share of the bottleneck link capacities at the same time.

The rest of the paper is organized as follows. Section 2 describes the related work. Then, in Section 3 we describe how to deliver bottleneck information to the end hosts in an efficient and scalable manner. Section 4 proposes a method to implement CoDel exclusively in the end hosts, using only the information gathered about the bottlenecks. The results shown in Section 5 validate our proof-of-concept CoDel implementation. We provide a discussion about the merits of the proposal in Section 6. Finally, the conclusions are laid out in Section 7.

2. Related work

Proposals for ICN congestion control algorithms can be divided into two main categories: end-to-end solutions and in-network protocols. The first try to adapt well-known Internet techniques to the multi-producer scenario, while the latter use the increased processing capabilities of ICN routers to both adapt forwarding rates and, when possible, spread the load among several producers and routers.

2.1. End to end approaches

Multipath-aware ICN Rate-based Congestion Control (MIRCC) (Mahdian et al., 2016) is a rate-based end-to-end multipath aware ICN congestion control mechanism that uses information provided by the routers themselves to select the best producer for each chunk while regulating, at the same time, the transmission rate via Interest pacing. The Path-specified Transport Protocol (PTP) works in a similar vein (Ye et al., 2018). Like MIRCC, PTP is able to control the traffic rate on each path independently, but it should be more easily scalable as it avoids performing direct rate estimations at the routers. A different alternative is explored in Wu et al. (2021). This technique tags the different paths used to obtain the chunks so that the consumer can use a different request rate for each producer. Then, a window-based end-to-end congestion control algorithm is applied along each *subpath*. In Qin et al. (2020) the consumer gets one bit of congestion information from the last eight routers along the downstream path. It then uses this information to both select the best path and perform an additive increase-multiplicative decrease (AIMD) congestion control algorithm to control the traffic rate. The authors of Hu et al. (2021) adapt the BBR (Bottleneck Bandwidth and Round-trip propagation time) congestion control algorithm (Cardwell et al., 2016) to NDN networks. Because data packets can come from different network nodes, they needed to adapt the RTT estimation procedure at the heart of BBR. Their results showed higher data rates than those obtained with classic congestion control algorithms and good performance in wireless environments.

In Ye et al. (2021a) the authors propose a network utility maximization model to formulate multi-source and multipath transmission in NDN with in-network caches, and implement a congestion control protocol that is able to monitor the RTT of each sub-flow independently. Their congestion estimation module measures the number of backlogged packets for each sub-flow—following the design principle of MultiPath TCP (MPTCP) wVegas (Cao et al., 2012) and TCP BBR. Then, it uses path switching as the underlying forwarding plane so that consumers can decide the forwarding path of each Interest packet. The path-specified congestion control enables content consumers to separate the traffic control on each path, which consequently facilitates fair and efficient bandwidth sharing among all consumers.

Remote Adaptive AQM (RAAQM) (Carofiglio et al., 2013) estimates at the consumer the AQM state of a random early drop (RED) managed queue at the bottleneck to generate synthetic congestion marks and drive a reactive congestion control algorithm. However, as it has to rely on indirect queue measures (mainly RTT variations) the range of possible AQM algorithms is limited. In contrast, our proposal is able to obtain direct queue length measures from all the involved routers to mimic and create a broader range of AQM algorithms.

2.2. In-network congestion control

Network-mediated congestion control protocols have also been proposed for ICN networks. They try to act as close as possible to the congestion by either rerouting around it and/or adapting the Interest forwarding rate. One of these in-network proposals is the Hop-By-Hop Interest Shaping mechanism (HoBHIS) (Rozhnova and Fdida, 2012). HoBHIS is a rate-based hop-by-hop congestion control mechanism that calculates the available capacity of each ICN router in a distributed manner to adjust its session Interest rate and, therefore, dynamically

regulate its data rate and transmission buffer occupancy. In Thibaud et al. (2020) the nodes exchange the requisites of different applications and the network constraints to drive the traffic along the most appropriate paths and then regulate the traffic rate via the proper pacing of Interest packets. The problem with these proposals, that only modify the forwarding rate inside the network, is that they are not able to completely eliminate congestion, because its ultimate cause is that the request rate of the consumers is just too high. Li et al. (2023) goes one step further and couples consumer rate-adjustment and in-network traffic diversion to overcome congestion and improve resource utilization.

A Deep Reinforcement Learning (DRL) technique is applied in the edge routers in Yang et al. (2022). The edge routers obtain accurate local information about congestion—maximum queue length of Data packets—inspecting the passing Data packets as they move downstream traveling through intermediate nodes. An *intelligent* rate adjustment mechanism in such edge routers classifies the collected path information according to their congestion degrees to provide suitable inputs for the DRL technique, and then outputs a reasonable transmission rate for Interest packets.

A different approach is presented in Nikmard et al. (2022). The authors propose a Dynamic Cache Placement (DCP) method to provide congestion avoidance by dynamically relocating the content of in-cache routers according to the traffic volume pattern and the link capacity. The DCP method distributes the popular data to the network regions with less traffic load and more accessible routers to balance the traffic load of congested routers. If buffer occupancy exceeds some predefined threshold, the provider nodes will notify appropriate neighbors in order to shift the traffic load, and even move the cached contents.

A way to ensure fairness among the different NDN flows is shown in Zafar et al. (2020). It works by shaping the rate of Interest packets in intermediate nodes. This avoids that applications that fail to obey congestion status obtain an excessive amount of resources.

The Yellow AQM in ICN routers (Zeng et al., 2021) adapts existing AQM algorithms to the non peer-to-peer nature of ICN flows. While in the TCP/IP architecture most communications are between just two hosts, in NDN Interest packets can be answered by any node containing a copy of the requested content. The paper provides a mechanism for routers to discern between cached and forwarded traffic adapting the AQM mechanisms.

The cooperation between caching and congestion control is further studied in Qu et al. (2023). The authors modify NDN so that a single Interest packet can provide several Data packets, and uses information added to the Data packets to improve caching efficiency. To overcome congestion, they estimate both delay and link capacity at every router and transmit this information attached to the Interest packets. Producers and caching nodes use this information to limit the sending rate of the data itself.

2.3. Hybrid alternatives

There are also alternatives that try to deal with congestion using both in-network and end-to-end techniques. For instance, in Ye et al. (2020) the authors propose a mechanism that implements an AQM algorithm in the routers without access to the underlying link characteristics, as this *link* can be a rate-varying channel, such as a WebSocket or a TCP tunnel. This mechanism uses RTT measurements between neighboring NDN nodes to drive the AQM algorithm and adds explicit congestion marks in Data packets to notify downstream consumers.

A different alternative is presented in Hashemi and Bohlooli (2021). A new congestion control module is added to the routers. This module calculates the forwarding probabilities to each possible outgoing interface, and communicates with the consumers to help them to manage the congestion window size. This requires both modifying the behavior of NDN routers and the format of NDN packets. Their added explicit-feedback mechanisms, both to the consumers and between NDN nodes, result in a multi-source and multi-path congestion control algorithm that is able to provide a fair share of resources to the consumers.

WinCM (Wang et al., 2018) uses a CoDel AQM algorithm to derive a congestion signal that is then sent to the consumers. However, this signal is modified so that it carries information about the actual excess of queuing delay. Consumers can use this extra information to adjust their congestion windows in a more precise way. This provides better stability when compared with approaches that use the conventional binary congestion signal. This information is also used by downstream NDN routers to adjust their forwarding decisions and avoid congested routes.

Finally, a well-known hybrid approach is also provided by PCON (Schneider et al., 2016). PCON capable routers add a *Congestion Mark* to Data packets that can be filled by any router to indicate congestion along the path. This congestion mark can then be employed both by downstream routers, to inform the path-selection algorithm when forwarding Interest packets and by the consumers themselves to pace their Interest producing rate. Song and Zhang (2022) works similarly, but in this case routers use queue size as the congestion feedback so downstream routers can adapt forwarding decisions and modulate their Interest sending rate.

Our proposal is focused on exploring the feasibility of moving queue management from the network core to the end hosts. For this, we only take advantage of only two NDN features: (a) nodes controlling the actual data rate are on the receiving end of traffic, and (b) there is more available space to carry state information in NDN packets than in IP packets. As such, we will design a mechanism for single-producer scenarios.

3. Delivering queue state to the edges

The peers in an end-to-end transmission need to obtain accurate and prompt information about the state of the network queues if they are to carry out the AQM operations themselves. In the NDN architecture, the consumer is ultimately responsible for the transmission rate, as it can modulate the amount of data in transit by controlling the number of outstanding Interest packets. So, our mechanism has to make sure that every consumer gets enough information to simultaneously perform AQM and congestion control operations in a timely manner. This information must be transmitted in the Data packets, as they are the only ones received by the consumers.

The current version of the NDN link protocol, NDN link protocol version 2 (NDNLPv2 (Junxiao et al., 2017)), defines a *Congestion Mark* tag that can carry up to eight bytes of information.[1] The only reserved value is 0, used to communicate no congestion. The rest of the values can be used, without restriction, to represent different levels of congestion, leaving the exact meaning to the congestion control strategy of the consumers.

We propose to include both the queuing delay and the transmission rate in the *Congestion Mark* tag, as they provide a rather complete assessment of the congestion experienced by packets in the outgoing link. Note that we prefer to include the link delay instead of the packet queue size since it usually provides a better characterization of congestion and, additionally, does not depend on the actual link capacity. In any case, the queue size can be obtained indirectly by simply multiplying the link rate by the delay.

Certainly, it is not efficient to transmit information about every queue along the path in every Data packet. There needs to be a way to coordinate the different routers so that they can take turns to transmit their congestion-related information.

[1] Clearly, 8 bytes are much more than what TCP/IP currently uses to signal congestion information (2 bits in the IP header, and two additional flags in the TCP header). However, we must consider that with the added information hosts can do both congestion control and AQM. Additionally, an 8-bytes field is not bigger than the size of other conventional options, like, for instance, the timestamp field in TCP, which occupies 10 bytes. In any case, it does not represent even 1 % of the typical Ethernet MTU.

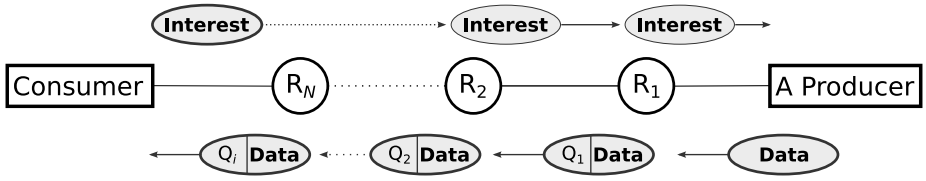

Fig. 1. Diagram showing the information flow from a producer to the consumer through routers R_1 to R_N. The queue state information of just one of the involved links (Q_i) is transmitted within the Data packet.

We have represented in Fig. 1 a simplified version of the proposed algorithm workflow. The consumer application regulates the transmission rate with the number of pending Interest packets. These Interest packets are forwarded to an appropriate producer (just one in the figure, for simplicity) that responds with Data packets. Then, the routers along the return path may include their outgoing queue state in the Data packet header. Note that while the first router is always able to include this information, the rest of the routers in the path will determine if they replace this information with the one of their own outgoing queue. For this they use the coordination method described in the next subsection. Finally, the consumer gets information from a link $i \in [1, N]$ and uses it to perform both AQM and congestion control.

3.1. Router coordination

We employ a stochastic approach for choosing which router should include the state information about its outgoing link in Data packets. Ideally, all the routers along the transmission path should get an equal chance to include a congestion mark. Since normal transmission windows can easily account for hundreds of packets, the consumer should receive several updates from each router every RTT. Keep in mind that, in order to perform AQM and not just congestion avoidance, we need status information about all non-empty queues along a path, and not just from the bottleneck one.

We propose a simple method that has several important properties:

- Each Data packet carries exactly one congestion mark.
- A given consumer receives, on average, the same number of congestion marks for every queue along the transmission path.
- Routers do not need to store extra state about ongoing transmissions.
- The coordination information can be completely contained in the Data packets themselves.

The algorithm is stated as follows. Every time a Data packet is about to depart the transmission queue:

1. If there is no congestion mark, add a new congestion mark.
2. If there is a congestion mark, replace it with probability $\mathrm{P_r}(i) = 1/i$, where i is the distance, in hops, from the producer to the current queue.

To analyze why such a simple probability function produces the desired results just consider, without loss of generality, that the ordered set of routers $\mathcal{R} = \{R_1, \ldots, R_N\}$ forms a path between a producer and the requesting consumer.

Lemma 1. *If $\mathrm{P_r}(i) = 1/i, \forall R_i \in \mathcal{R}$, the probability that the congestion mark from a particular router $R_j \in \mathcal{R}$ arrives to the consumer is $\mathrm{P_a}(j) = 1/N$.*

Proof. The probability $\mathrm{P_a}(j)$ that the congestion information about the outgoing link of router R_j gets to the consumer is directly the probability that router R_j replaces the previous congestion tag and that no subsequent router replaces its congestion tag. So

$$\mathrm{P_a}(j) = \mathrm{P_r}(j) \times \prod_{i=j+1}^{N} \left(1 - \mathrm{P_r}(i)\right) = \frac{1}{j} \times \prod_{i=j+1}^{N} \left(1 - \frac{1}{i}\right) = \frac{1}{N}. \quad \square \tag{1}$$

Please note that two consumers downstream of the same router, but with different path distances N and N', will receive a congestion mark from it with respective probabilities $1/N$ and $1/N'$. Also notice that the only additional information needed by the coordination algorithm is the current distance to the producer, which can be encoded using a single byte of the *Congestion Mark* tag. We will refer to this byte as the *count* value.

3.2. Encoding congestion information

Consumers need to store information about all the queues in the transmission path. To do this, they need to be able to identify all of them. To this end, the *Congestion Mark* embedded in the Data packet has to include a permanent unique identifier (ID) of the queue entering the congestion information. Successive congestion state information from the different routers enables the different consumers to track congestion dynamics. This ID, however, does not need to have any associated meaning, and consumers should treat it as an opaque bitstream. A straightforward implementation consisting of assigning a sequential ID to each router would require no further coordination among the routers—recall that the *count* value is transmitted in the *Congestion Mark*. However, it would fail as soon as a single router is shared by two or more transmissions originating from different producers. The alternative, also without explicit coordination, is to randomly select each queue ID from a sufficiently large population of different identifiers.

The length of the `queue ID` field limits the allowed population size. The size of the `queue ID` field is a compromise between the ID collision probability and the space left for encoding the *count*, *rate* and queuing *delay* values. We need to account for the fact that paths with more than ten routers are common nowadays on the Internet. In any case, paths consisting of sixty routers, although very rare, also have to be considered.[2] The probability that at least two queues along a path share the same random ID is an instance of the well-known *birthday paradox* problem. The collision probability for a path with L queues and 2^n possible identifiers is given by

$$\mathrm{P_c}(L, n) \approx 1 - \mathrm{e}^{-L^2/(2^{n+1})}. \tag{2}$$

Table 1 provides the values for some combinations of n and L values. As we can see, having less than 24 bits produces unacceptable high collision probabilities even for conservative path lengths. In fact, even using 24 bits for the ID would result in more than 1 in every 100 000 transmissions being unable to fully distinguish information from involved queues. We feel that 32 bits provide the right level of protection, with negligible collision chance for usual path lengths (< 20 hops) and very small probability for larger paths. In any case, note that this is the worst case scenario in which every link along a path becomes the bottleneck at least once during the communication and thus needs to perform AQM operations. In practice, very few links ever become a bottleneck, so it is enough to avoid collisions only between IDs pertaining to bottleneck queues. For multipath transmissions, we do not need to ensure that all the router IDs of the different subpaths

[2] Many popular operating systems currently employ 64 as the initial value of the Time To Live (TTL)/Hop-limit field of IP datagrams.

Table 1
Collision probability for different path length (L) and ID bit length (n) combinations.

	ID bit length (n)					
L	4	8	12	16	24	32
8	0.860	0.120	7.78×10^{-3}	488×10^{-6}	1.91×10^{-6}	7.45×10^{-9}
16	1.00	0.393	30.8×10^{-3}	1.95×10^{-3}	7.63×10^{-6}	29.8×10^{-9}
32	1.00	0.865	0.118	7.78×10^{-3}	30.5×10^{-6}	119×10^{-9}
64	1.00	1.00	0.393	30.8×10^{-3}	122×10^{-6}	477×10^{-9}
128	1.00	1.00	0.865	0.118	488×10^{-6}	1.91×10^{-6}

are different, just that the bottleneck IDs of the different subpaths are different. 32 bits should ensure this easily as long as the number of simultaneous subpaths is not too large.

Using 32 bits for the `Queue ID`, and 8 for encoding the *count* value, leaves just 24 bits to simultaneously encode the *queuing delay* (δ) and the transmission *rate* on the queue (r). The range of possible delay values goes from nanoseconds (it just takes 120 ns to transmit a 1500 bytes frame on an 100 Gb/s Ethernet link) to seconds. The possible transmission rates also encompass a similar magnitude range. Clearly, 24 bits are not enough to simultaneously encode both values with satisfactory precision. We propose that routers randomly select whether to send rate or queuing delay information and use one bit of the *Congestion Mark* to indicate the kind of information being transmitted (bit R). Using a random procedure should avoid any undesired correlation between traffic patterns and the information received by each consumer. This ensures that every consumer gets, approximately, the same amount of information about both the bottleneck delay and its current transmission rate. The other 23 bits are used to encode the rate (in bits per second) or the queuing delay (in picoseconds). As 23 bits are just enough to represent almost 7 orders of magnitude, we employ a floating-point representation, using 15 bits for the significand and 8 for the exponent. In this way we can represent any conceivable rate or queuing delay value.

The final encoding of the *Congestion Mark* is represented in Fig. 2, where the bit R is set to 1 when the transmitted information is a rate and 0 if it carries a delay value.

Algorithm 1 Router marking procedure.

```
1:  queue_id ← random(0, 2^32 − 1)
2:  function MarkOutgoingPacket(pkt)
3:      if pkt has congestion mark then
4:          i ← pkt.Count + 1
5:      else
6:          i ← 1
7:      if random(0, 1) < 1/i then
8:          pkt ← ReplaceCongestionMark(pkt)
9:      pkt.Count ← i
10:     return pkt
11: function ReplaceCongestionMark(pkt)
12:     pkt.Queue ID ← queue_id
13:     R ← round(random(0, 1))
14:     pkt.R ← R
15:     if R = 0 then
16:         pkt.Value ← delay
17:     else
18:         pkt.Value ← rate
19:     return pkt
```

A description of this procedure in algorithmic form is shown in Algorithm 1. The random(a, b) function provides a uniform number in the interval (a, b). At startup each router generates its own *unique* identifier (*queue_id*). Afterwards, before transmitting each Data packet, the MarkOutgoingPacket function is called, and it modifies the packet according to the rules detailed previously in the text.

4. A client-located CoDel implementation

The CoDel (Nichols and Jacobson, 2012; Nichols et al., 2018) active queue management algorithm tries to keep bottleneck links fully busy while bounding the average queuing delay. To this end, it keeps some statistics about the queue length (in time units) during the last measuring interval as well as the status of the algorithm itself. When it detects that the queue length is persistently greater than a predefined target value, it enters the *drop state* and starts marking some packets with congestion indications. The actual operation algorithm can be seen in Fig. 3. Every time a new packet j is transmitted, its sojourn time (S_j) is compared to the *target* value. When this value is greater than the target, it enters the drop state. If the queue has stayed in the drop state for more than the *interval* time (a parameter of the CoDel algorithm), the outgoing packet is either dropped or marked with a congestion indicator. Additionally, the interval time is reduced with a power low to ensure a rapid congestion response if the congestion is persistent.

Whereas in current networks this procedure has to be carried out by the routers, and as such, the initial values of both the *target* and *interval* parameters are decided by the network admin, in our proposal it is up to the consumers to mimic this behavior. Recall that, using the previously proposed coordination protocol, the routers provide the consumers with the necessary information in the congestion tag: a sample of the sojourn time of packets as they exit the network queues.

However, consumers must be aware that they are usually sharing the queue with other consumers. In a classical AQM implementation, the router will not notify every consumer about the congestion situation, but only the one whose packet has been selected (in the case of CoDel, at most one packet after each *interval*). If all the consumers were to react simultaneously to the congestion indication, their synchronized response would likely be a cause of severe network instability. To avoid this problem, consumers in our proposal implement a modified version of the marking procedure where the *total* and *interval* variables are always updated (to keep the state of the AQM algorithm), but the packet is only marked with some probability $\mathrm{P_m}$. This probability should be the same as the probability that a packet from this consumer would have been dropped or marked by the AQM mechanism in the router. Setting

$$\mathrm{P_m} = \frac{\text{consumer rate}}{\text{link rate}} = \frac{\widehat{RTT} \times cwnd}{\text{link rate}}, \tag{3}$$

where $\widehat{RTT}$ is the consumer's estimation of the round-trip delay and $cwnd$ is its current congestion window, satisfies this requirement as long as the sizes of packets crossing the bottleneck are similar. Note that all this information is already available to consumers.

In addition to the usual information related to congestion control, consumers have to store additional information about the state of each queue to perform AQM operations. Let us define

$$\mathcal{Q}_i = \{C, r, \delta\}, \tag{4}$$

the state information of the queue identified by the queue id i, where r is the last measure of its throughput, δ is the queuing delay, and C is the state of the CoDel algorithm:

$$C = \{state, interval, total\}, \tag{5}$$

with $state \in \{\text{No Drop}, \text{Drop}, \text{Mark}\}$.

Now, every time a new Data packet is received with a congestion tag, the consumer extracts the queue identifier i and updates $\mathcal{Q}_i$: the rate or queuing delay is updated to the received value and the CoDel state C is recalculated following Algorithm 2. The UpdateCodelState function is called with the current values of the CoDel state C, that is updated with its output, and the most recent measure of the sojourn delay ($\mathcal{Q}_i[\delta]$). Finally, if after returning from the function, $\mathcal{Q}_i[C] =$ Mark, a congestion signal is fed to the congestion control algorithm with probability $\mathrm{P_m} = \widehat{RTT} \times cwnd / \mathcal{Q}_i[r]$. The complete procedure is detailed in pseudocode form in Algorithm 3.

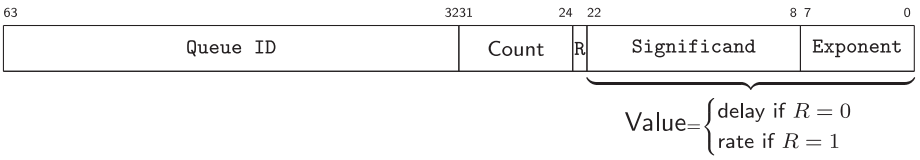

Fig. 2. Information encoding in the congestion tag. Bit R controls if the *Congestion Mark* carries a delay or a rate.

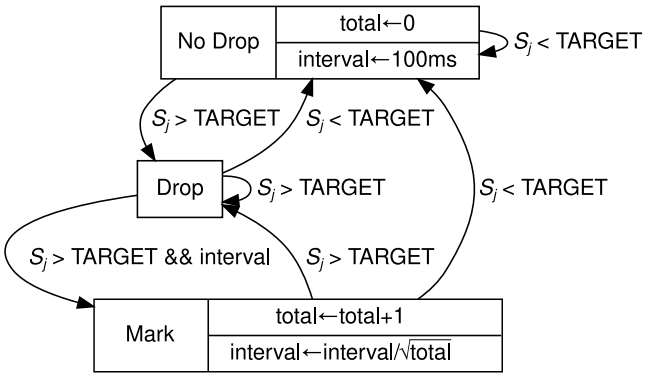

Fig. 3. Modified CoDel algorithm finite-state machine.

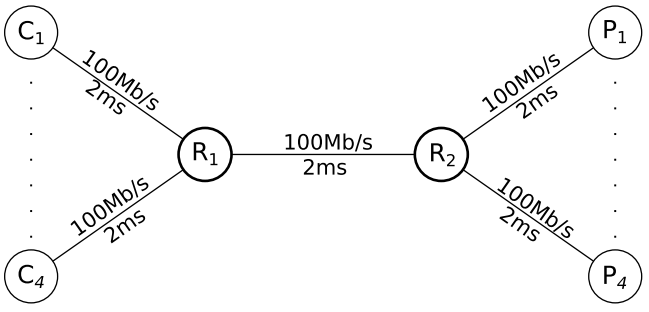

Fig. 4. Dumb-bell topology for the multi-consumer scenario. C_1 to C_4 are the consumers, R_1 and R_2 are two intermediate routes and, finally, P_1 to P_4 is the set of producers.

Algorithm 2 CoDel algorithm. In the code, *now* is a variable that always has the current time. This code is called for every Data packet arrival with the latest available queuing delay value δ.

function UpdateCodelState($\{state, interval, total\}$,$\delta$)
  if $state$ = No Drop **then**
    $total \leftarrow 0$
    $interval \leftarrow 100$ ms
    if $\delta >$ TARGET **then**
      $state \leftarrow$ Drop
      $status\_change \leftarrow now$
  else if $state$ = Drop **then**
    if $\delta >$ TARGET **and** $now \geq status\_change + interval$ **then**
      $state \leftarrow$ Mark
    else if $\delta <$ TARGET **then**
      $state \leftarrow$ No Drop
  else if $state$ = Mark **then**
    $total \leftarrow total + 1$
    $interval \leftarrow interval/\sqrt{total}$
    if $\delta >$ TARGET **then**
      $state \leftarrow$ Drop
      $status\_change \leftarrow now$
    else
      $state \leftarrow$ No Drop
  return $\{state, interval, total\}$

Algorithm 3 Complete procedure executed on arrival of Data packets.

function SignalsCongestion(pkt)
  $i \leftarrow$ pkt.Queue ID
  $val \leftarrow$ pkt.Significand $\times 2^{\text{pkt.Exponent}}$
  if pkt.R = 0 **then**
    $Q_i[\delta] \leftarrow val$
  else
    $Q_i[r] \leftarrow val$
  $\{state, interval, total\} \leftarrow$ UpdateCodelState($Q_i[C], Q_i[\delta]$)
  $Q_i[C] \leftarrow \{state, interval, total\}$
  if $state$ = Mark **and** random$(0, 1) < \widehat{RTT} \times cwnd/Q_i[r]$ **then**
    return true
  return false

5. Results

We have implemented a proof of concept version of the previous algorithm as a modification to the `GenericLinkService` class of the NDN Forwarding Daemon (NFD) (Anon, 2022). Then we run it with the help of a new Consumer application under the NDN Simulator (Anon, 2020). The source code for the simulated applications and the associated code implementing both the consumer application and the modifications to NFD are available to download at Rodríguez Pérez (2023).

The Consumer application uses the information gathered from the routers to generate the congestion signals it should receive if there were a bottleneck in the network running the CoDel AQM algorithm with a target delay of 5 ms. Then, it uses these synthetic congestion signals to drive a simplified Reno-like congestion control algorithm to manage the amount of pending Interest packets.

We have set up three simulation experiments to assess the behavior of the algorithm. We have employed producers with an infinite amount of pending data to be transmitted to a set of consumers requesting orthogonal sets of the data. As we are solely interested in the ability to move the queue management to the network terminals, we have ignored other information centric characteristics of the architecture, such as native multicast delivery or in-network caching. The payload size was set to 1024 bytes and the NDN packets were not encapsulated by any legacy protocol, but transmitted directly over point-to-point links.

5.1. Simple topology

In the first experiment, we used a simple dumb-bell topology to gain confidence in the proper behavior of the algorithm and its implementation. The precise network topology is the one represented in Fig. 4. All the links in this network have the same characteristics, that is, a 100 Mb/s capacity and a propagation delay of 2 ms. In this experiment, four consumers (C_1, C_2, C_3, C_4) request data from the corresponding four producers (P_1, P_2, P_3, P_4) while sharing a bottleneck (the $R_2 \rightarrow R_1$ link). To observe the dynamic behavior, the consumers are not started simultaneously, but instead they wait 30 s after one another. In the same way, they do not stop at the same time. Instead, C_i stops at instant $t_i = 120 + 30i$ seconds, so only in the period from 90 s to 150 s all the consumers are active.

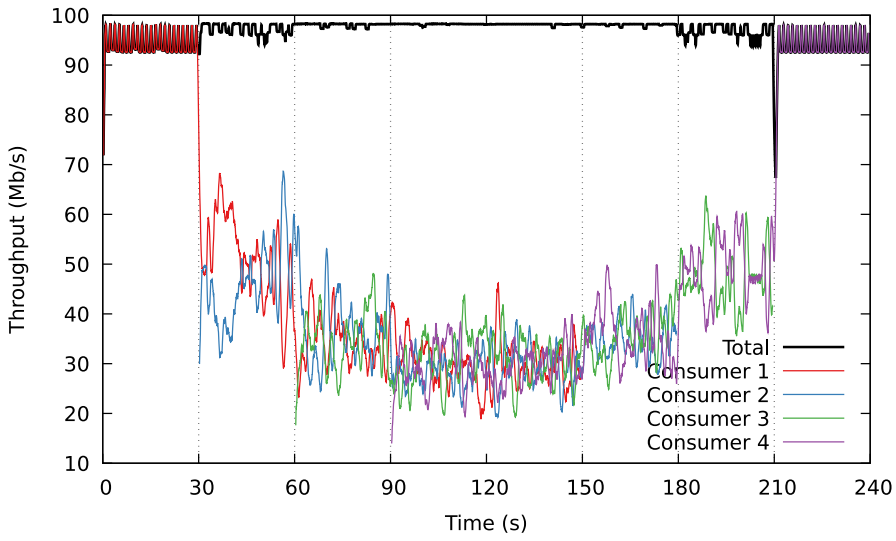

Fig. 5. Instantaneous throughput obtained by each consumer and its sum under the dumb-bell topology.

Table 2
Average throughput obtained by each consumer (C_1 to C_4) in every 30 s interval (in Mb/s).

Interval	C_1	C_2	C_3	C_4
0 s–30 s	91			
30 s–60 s	46	47		
60 s–90 s	31	34	31	
90 s–150 s	22	22	27	24
150 s–180 s		31	34	29
180 s–210 s			45	49
210 s–240 s				91

Fig. 5 shows the instantaneous throughput of each consumer in colored lines, while the total throughput across the bottleneck link is shown in black color. It can be seen how the total traffic stays very close to 100 Mb/s during the whole experiment, with just a brief interruption when the penultimate flow abandons the network while the last flow augments its rate. The individual rates show that, on average, each consumer gets close to its expected fair share of the available capacity, even if the results are rather noisy, as is to be expected from an AIMD congestion control algorithm. We have included the average throughput for each consumer in each interval in Table 2.[3] The results confirm that our solution is able to reach a fair allocation of the bottleneck capacity among the competing flows.

The delay experienced by each consumer during each transmission is plotted in Fig. 6. The expected average round-trip delay has been represented as a black line at 17 ms—twice the propagation delay (6 ms) plus the 5 ms target for the CoDel algorithm. Even though the round-trip delay of each individual packet varies with quite a large amplitude, its averaged value (represented with a wider line) stays close to the target value most of the time. The exception is when the number of flows is too small. We see that, when less than three flows share the bottleneck, the AIMD reaction to congestion indications keeps the average (but not the instantaneous) round-trip delay below the target. In any case, recall that the link capacity is still fully employed.

To gain more confidence in the AQM results, we also check both the instantaneous and the average length of the bottleneck queue, represented in Fig. 7. As before, we have employed a black line to represent the expected maximum queue length: in this case, almost 60 packets, equivalent to 5 ms of queuing delay.[4] As expected, the figure mirrors the delay shown in Fig. 6. When the number of flows is small, as soon as the delay grows over the target, the flows react and the queue size is kept well below its target. However, as the number of flows grows, we can appreciate that the consumers manage to keep the average delay controlled around the AQM activation value.

5.2. Cascade topology

In our second experiment we wanted to test our proposal in a more challenging environment with changing bottleneck links. For this, we have employed the network topology depicted in Fig. 8. As before, there are four consumers with the same round-trip delays (so that their expected rates stay identical), but now there are three different downstream bottlenecks: one located in the output queue of R_2 to R_1, the second one from R_3 to R_2 and, finally, one on the outgoing producer interface. In this experiment, C_1 is active during the whole simulation, C_2 from 50 s to 300 s, C_3 from 100 s to 250 s and C_4 from 150 s to 200 s. In this way, the bottleneck *moves* between R_1, R_2 and R_3 throughout the experiment. Also notice that the expected individual rates when there are two or more competing communications are always 50 Mb/s. We have also employed this topology to compare the performance of our approach to that of the canonical CoDel implementation in a TCP/IP network. For this, we have established four TCP connections that always have data ready to be sent from node P to each of the client nodes. The actual TCP version used was NewReno as implemented in ns−3.

Fig. 9 shows the transmission rate obtained by each consumer as well as the total one, represented again by a black line. The total rate almost coincides, as expected, with the capacity of the bottleneck and, as in the previous experiment, it can be easily appreciated that each consumer approximately gets the same share of the bottleneck capacity.

For comparison, the results obtained by the TCP version are plotted in Fig. 10. Clearly, the obtained results are analogous to those of our implementation, except for the fact that in the NDN case the reaction to congestion takes just half an RTT, as it is the consumers that manage the transmission rate. This is more noticeable in the first (and last) 50 s of the simulation, where the amplitude (frequency) of the rate variations is smaller (higher) in Fig. 9 than in Fig. 10.

[3] We disregarded the first 5 s in each interval to give some time to the congestion control algorithms to reach a steady state.

[4] A 100 Mb/s transmission during 5 ms accounts for 58.3 packets, considering 1024 bytes of payload per packet, plus 48 bytes of headers.

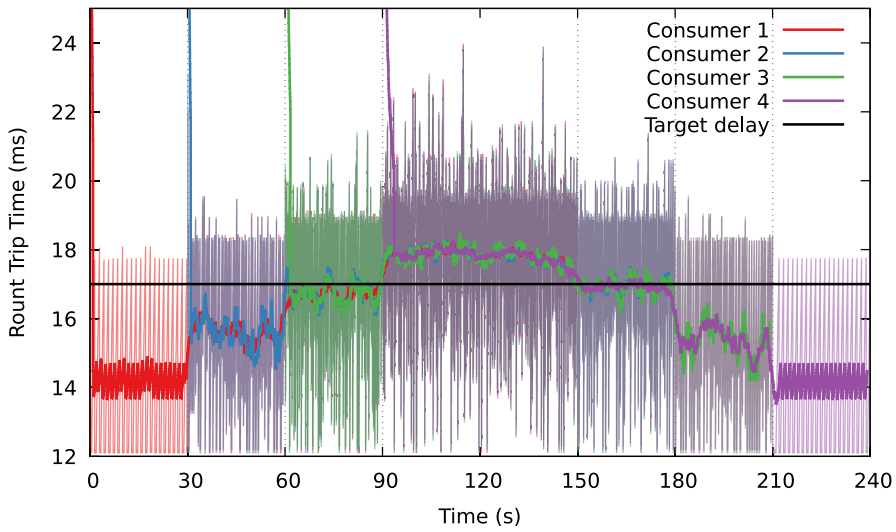

Fig. 6. Instantaneous delay for each consumer under the dumb-bell topology. The wider colored lines represent short-term averaged values.

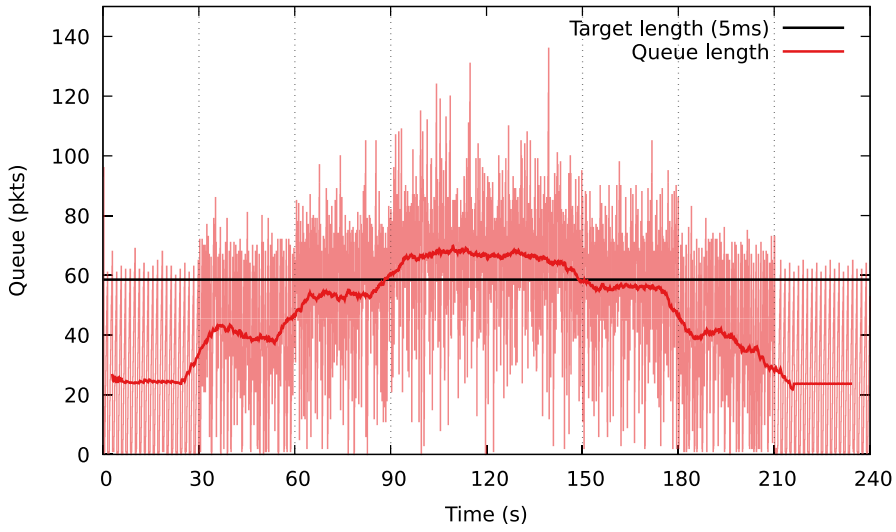

Fig. 7. Bottleneck queue size for the dumb-bell topology. The wider colored line is the short-term averaged queue size.

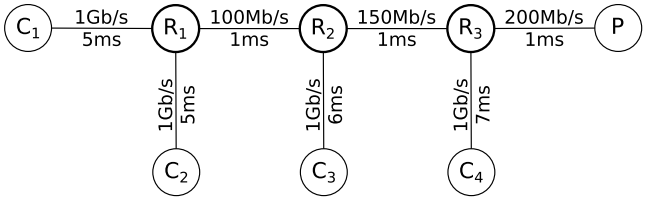

Fig. 8. Network topology for the cascade network. There is one producer (P), four clients (C_1 to C_4) and three intermediate routers (R_1 to R_3).

The sizes of the queues during the simulation are depicted in Fig. 11. The figure includes, for each bottleneck queue, its short term average overlaid using a bolder line. The black line represents the expected queue length at every given interval. Notice that, as the bottleneck moves during the simulation, the expected bottleneck length, when measured in data packets (or bytes), should vary to account for the different link capacities but, in any case, it corresponds with 5 ms of queued traffic. We omit, for the sake of brevity, the results from the TCP scenario, since they are quite similar to those obtained in the NDN case.

Recall that, from 0 s to 100 s and from 250 s to 350 s, only consumers behind R_1 are active and, therefore, only a queue builds up on the link from R_2 to R_1. However, for 100 s to 150 s and for 200 s to 250 s, the 150 Mb/s capacity of link R_3 to R_2 is utilized and so the bottleneck queue forms on that link. In the figure we can see how the bold red line diminishes, albeit it does not completely disappear, while the blue line oscillates around its target. Finally, from 150 s to 200 s, all the consumers are active and the bottleneck queue forms on the outgoing link from P to R_3. Consequently, the two other queues diminish, while the new one oscillates around its target.

We also take a look at the information gathered by the consumers themselves about the queuing status. We will use consumer C_1 for this, as it is active during the whole experiment, but the results should extrapolate to the other consumers. As stated in Section 4, consumers need to obtain information about the throughput and the queuing delay of the bottleneck link if they are going to implement CoDel by themselves. As the routers themselves ignore where the bottleneck resides, the consumers get information from all the routers along the path. Fig. 12 shows the information received by consumer C_1 about the throughput at the routers. From 0 s to 100 s and from 250 s to 350 s all routers send approximately 100 Mb/s worth of data. Again, the result is quite noisy, but the short term average stays relatively constant. In the

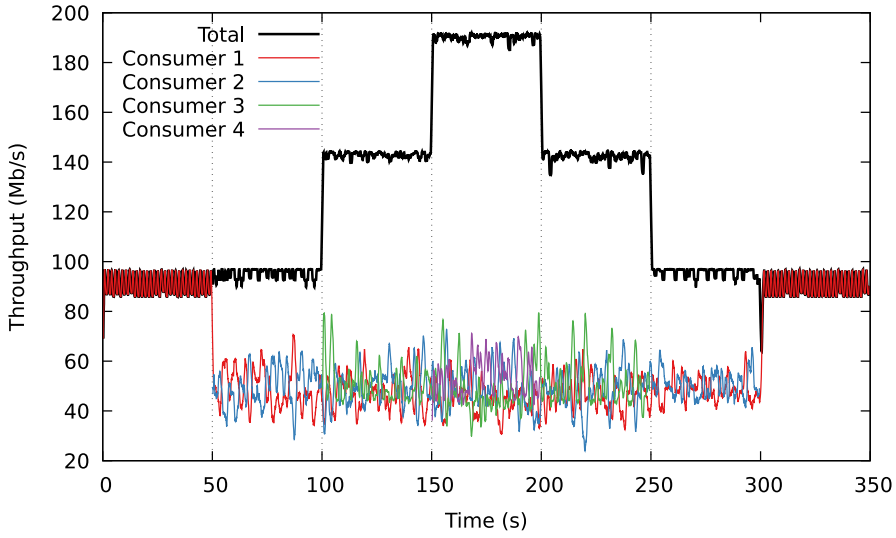

Fig. 9. Instantaneous throughput obtained by each NDN consumer and its sum under the cascade topology.

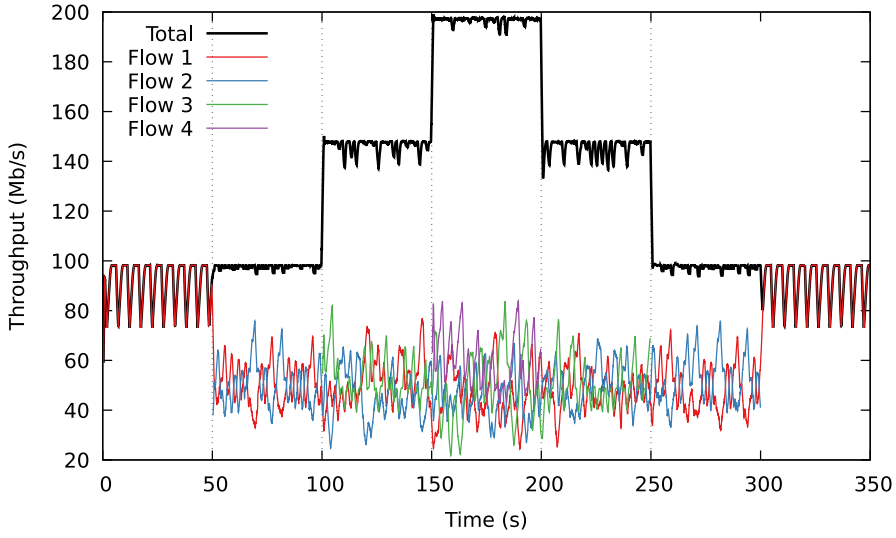

Fig. 10. Instantaneous throughput obtained by each TCP flow and its sum under the cascade topology (in-network CoDel scenario).

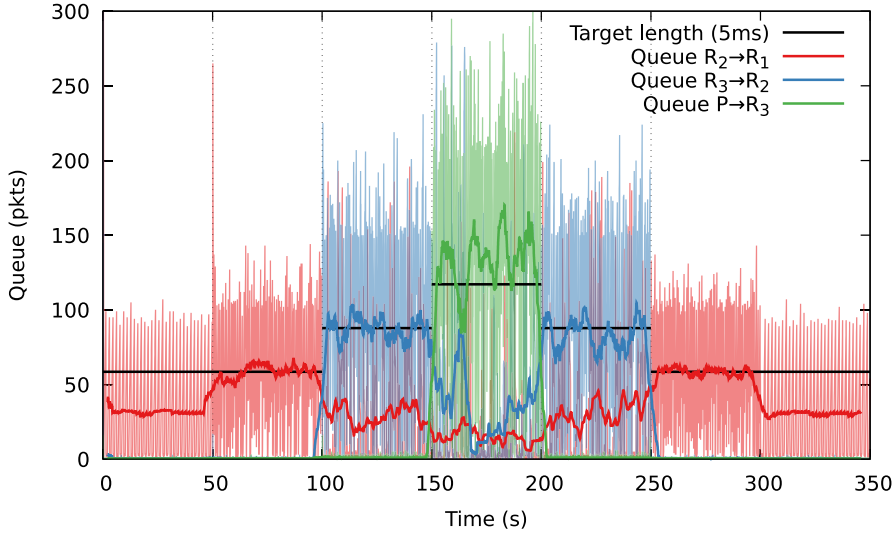

Fig. 11. Bottleneck queue size for the cascade topology. The wider colored lines represent short-term averaged values.

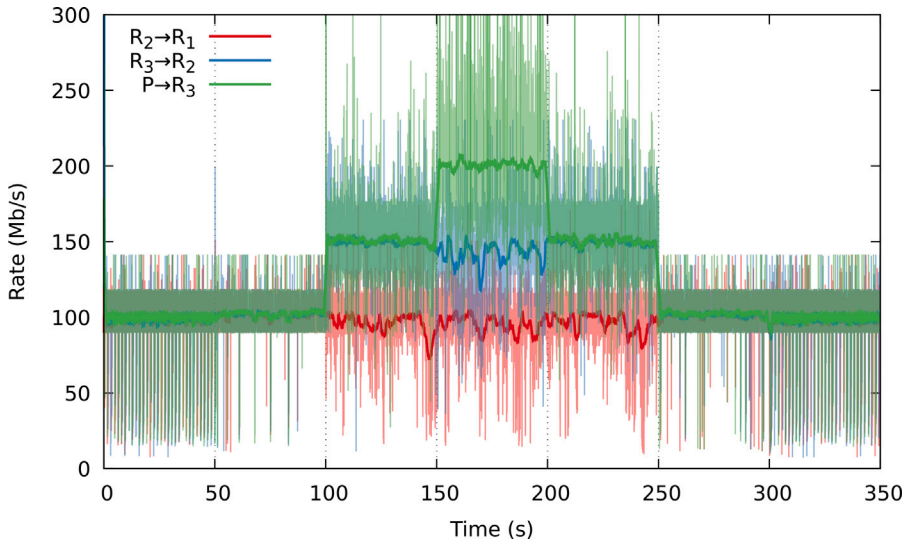

Fig. 12. Throughput on each queue as known to the consumer C_1. The wider colored lines represent short-term averaged values.

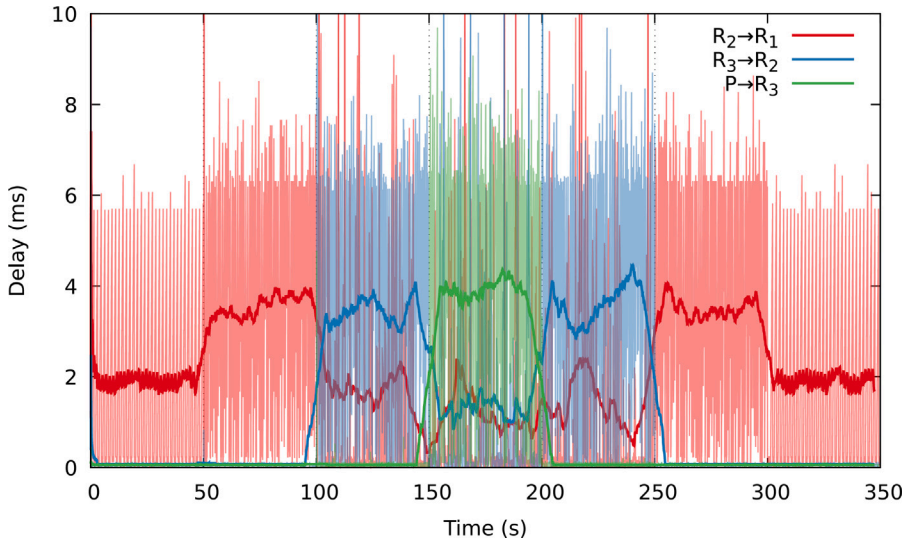

Fig. 13. Queuing delay on each queue as known to the consumer C_1. The wider colored lines represent short-term averaged values.

intervals from 100 s to 150 s and from 200 s to 250 s we can distinguish the different throughput of link R_2 to R_1 and those of P to R_3 and R_3 to R_2, as they no longer overlap. Finally, between 150 s and 200 s each bottleneck is observing a different throughput and thus the figure shows the three different rates. As expected, consumer C_1 is able to obtain the right information about each bottleneck rate for the whole duration of the experiment.

Finally, Fig. 13 shows the information gathered by consumer C_1 about the queuing delay at each router. It clearly mirrors Fig. 11, albeit in this case the information is directly provided in time units.

5.3. Multiple bottlenecks topology

Our final set of experiments involves testing the proposal in a parking-lot network topology that exhibits multiple simultaneous bottlenecks. The actual topology is represented in Fig. 14. There are three simultaneous transmissions from P_1, P_2 and P_3 to C_1, C_2 and C_3, respectively, that have to go through the whole network. At the same time, transmissions from P_i to C_i, $i \in [4, 16]$, ensure that the outgoing link from each router R_{j+1} to R_j stays congested.

Table 3 summarizes the achieved rates from both groups of clients. It is easy to see that clients C_1 to C_3 should get the same rate, and, in fact, their fairness level, as measured with Jain's index (Jain et al., 1984), is almost 1. The same happens with clients C_4 to C_{16}. They all get the same RTT and compete against the first three clients. Again, both the small standard deviation and the fairness index very close to 1 confirm it. There is, however, a very big difference between the rates obtained by both sets of clients that is caused by two simultaneous effects. On the one hand, the first three clients experience a much higher propagation delay, and hence, even bigger RTTs when accounting for the queues. Secondly, as almost all links are congested, the expected loss rate—in our case, the rate of congestion notification by the AQM mechanism—has to be different. If p_1 is the loss probability on a single queue, then the loss probability for the first three clients grows to $p_{13} = 1 - (1 - p_1)^{13}$.[5] In our experiments, we have measured that clients C_4 to C_{16} experience an RTT around 17 ms. Then, according to Mathis's formula (Mathis et al., 1997),

$$p_1 = \left(\frac{1.22 \cdot MSS}{RTT \cdot rate}\right)^2 = \left(\frac{1.22 \cdot (8 \cdot 1024)\,\text{bits}}{17\,\text{ms} \cdot 0.8225\,\text{Mb/s}}\right)^2 \approx 47.72 \times 10^{-6}, \qquad (6)$$

[5] Note that, according to the aggregate rate obtained by clients C_1 to C_3, there should be no queue on any output of R_1.

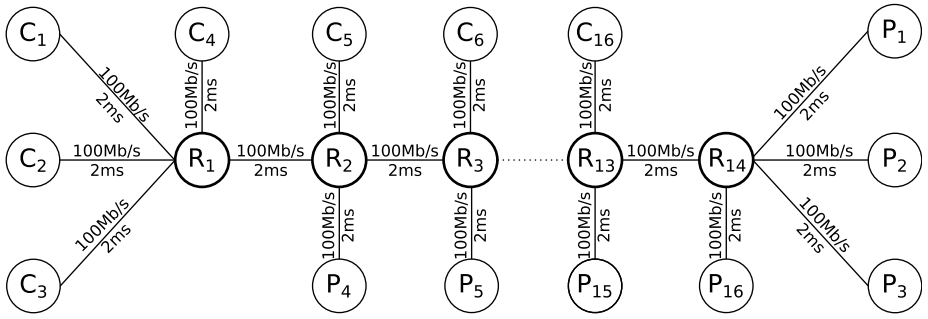

Fig. 14. Network topology for the multi-bottleneck network. There are sixteen pairs of producers and consumers (consumer C_i requests data from producer P_i). Nodes R_1 to R_{14} are intermediate NDN nodes.

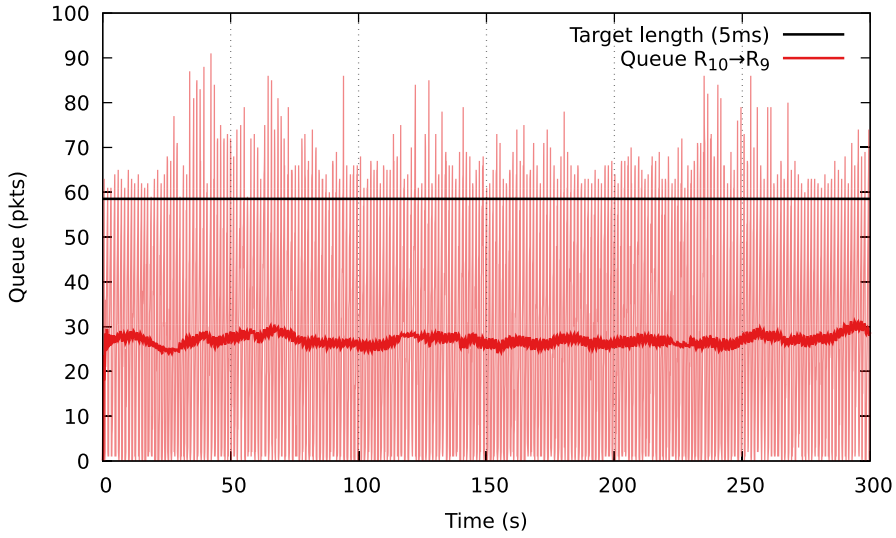

Fig. 15. Bottleneck queue size for the multi-bottleneck topology on the $R_{10} \rightarrow R_9$ link. The wider colored line represents short-term averaged values.

Table 3
Average client rate, standard deviation and fairness index for the two different sets of clients in the multi-bottleneck scenario.

Clients	C_1–C_3	C_4–C_{15}
Av. rate	5.23 Mb/s	85.1 Mb/s
Rate std. dev.	0.823 Mb/s	0.497 Mb/s
Fairness	0.983 76	0.999 97

where MSS is the maximum segment size. So, $p_{13} \approx 620.2 \times 10^{-6}$ and the three first clients should achieve a rate about 4.180 Mb/s when we take into account their measured RTT of 96 ms and 1024 bytes packet sizes. This rate value is, clearly, in the same range as that shown in Table 3.

Fig. 15 shows the instantaneous behavior of one representative queue, in particular, that of link between R_{10} and R_9. We can see that the AQM mechanism is able to react when the queue occupation grows larger than our 5 ms target delay. As there is just one flow using most of the bandwidth, it behaves similarly to that of the dumb-bell scenario with just one client (see the first 20 s of Fig. 7), with an average occupation between 20 and 30 packets. Table 4 shows the average queue sizes and standard deviation for the rest of the queues—recall that no queue forms in any outgoing link of R_1. As expected, all queues show very similar results.

6. Discussion

Shifting queue management operations to the network terminals, in line with the end-to-end philosophy that was the basis of the original Internet architecture, has both advantages and disadvantages. Such a mechanism opens up new avenues for experimentation, as modifications and tuning would not depend on network operators or equipment manufacturers. However, widespread adoption requires standardization and coordination among the network users. Such large-scale deployment is only in the hands of organizations controlling the major operating systems. We have shown that it is possible to entirely mimic the behavior of a modern AQM algorithm at the end hosts. We believe that it is an alternative worth exploring, even if it is restricted to new network architectures, relatively isolated networks, or even for when Internet providers are hesitant to set up proper AQM mechanisms in their networks.

We have shown that it is possible to obtain and transmit the needed information to the hosts in an efficient and scalable manner. In fact, in our particular scenario in an NDN network, the proposed modifications do not need any change in the current packet format. The added state in the routers—just their average rate and queue occupancy—scales linearly with the number of interfaces. Consumers, in contrast, have to store information about every router in their paths, but that is a modest amount of memory for current network devices.

Table 4
Average queue size and standard deviation on each router (in packets) for the multi-bottleneck scenario. R_1 is omitted as no queue is formed in its outgoing interfaces.

	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}
Av.	29.5	28.4	29.6	28.2	27.9	27.7	29.3	29.1	28.9	29.4	29.9	29.8	29.4
Dev.	21.3	21.5	22.3	21.4	21.6	21.8	21.6	21.5	21.4	23.3	21.7	25.5	25.5

This information makes it possible to implement advanced AQM algorithms while still providing the congestion control capabilities common to end hosts in the current Internet. This opens the door to both experimentation and rapid deployment of new AQM schemes.

7. Conclusions

In this paper we have tested the feasibility of performing joint congestion control and active queue management in an end-to-end way. To this end, we have implemented these capabilities as a NDN application with minimal changes to the NDN router implementation. In particular, we have leveraged the existing 64-bit congestion information field available in the link layer of the NDN architecture to transmit real-time and accurate information about both the queuing delay and the link throughput from the routers to the end hosts in an efficient and scalable manner. Moreover, all routers inside a given path are able to transmit this information to the downstream consumers with the same probability and with no need for explicit coordination.

We have also demonstrated how this information can be used to mimic a modern AQM algorithm (CoDel) and perform TCP-compatible congestion control simultaneously at the receivers. We have proved via simulation that our CoDel implementation is able to control the average queue sizes in all the bottlenecks and that each consumer obtains a fair share of the network capacity.

We plan to expand this initial proof-of-concept work to explore the convergence of this idea with the multipath capabilities of NDN networks, and, in particular, with the possibility to identify subpaths thanks to the congestion information obtained from the different bottlenecks. This should help with multipath transmissions, as it would facilitate the estimation of the RTT for each subpath separately. Additionally, we also plan to test our proposal in a multicast setup.

CRediT authorship contribution statement

Miguel Rodríguez-Pérez: Conceptualization, Software, Writing – original draft, Funding acquisition. **Sergio Herrería-Alonso:** Validation, Supervision, Funding acquisition, Writing – review & editing. **J. Carlos López-Ardao:** Writing – review & editing. **Raúl F. Rodríguez-Rubio:** Writing – original draft, Investigation.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Miguel Rodríguez Pérez on behalf of all the authors.

Data availability

Data will be made available on request.

Acknowledgments

This work has received financial support from grant PID2020-113240RB-I00, financed by MCIN/ AEI/10.13039/501100011033, and by the Xunta de Galicia (Centro singular de investigación de Galicia accreditation 2019–2022) and the European Union (European Regional Development Fund—ERDF). Funding for open access charge: Universidade de Vigo/CRUE-CISUG.

References

Anon, 2020. [ndnSIM]: NS-3 based NDN simulator. https://github.com/named-data-ndnSIM/ndnSIM.

Anon, 2022. Named data networking forwarding daemon. https://github.com/named-data/NFD.

Cao, Y., Xu, M., Fu, X., 2012. Delay-based congestion control for multipath TCP. In: 2012 20th IEEE International Conference on Network Protocols (ICNP). (ISSN: 1092-1648) pp. 1–10. http://dx.doi.org/10.1109/ICNP.2012.6459978.

Cardwell, N., Cheng, Y., Gunn, C.S., Yeganeh, S.H., Jacobson, V., 2016. BBR: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. Queue 14 (5), 20–53. http://dx.doi.org/10.1145/3012426.3022184.

Carofiglio, G., Gallo, M., Muscariello, L., Papali, M., 2013. Multipath congestion control in content-centric networks. In: 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). pp. 363–368. http://dx.doi.org/10.1109/INFCOMW.2013.6970718.

Hashemi, S.N.S., Bohlooli, A., 2021. 3CP: Coordinated congestion control protocol for named-data networking. IEEE Trans. Netw. Serv. Manag. 18 (3), 3918–3932. http://dx.doi.org/10.1109/TNSM.2021.3086437, conference Name: IEEE Transactions on Network and Service Management.

Hu, Y., Serban, C., Wang, L., Afanasyev, A., Zhang, L., 2021. BBR-inspired congestion control for data fetching over NDN. In: MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM). (ISSN: 2155-7586) pp. 426–431. http://dx.doi.org/10.1109/MILCOM52596.2021.9652898.

Jain, R.K., Chiu, D.-M.W., Hawe, W.R., 1984. A quantitative measure of fairness and discrimination for resource allocation in shared computer system. Tech. Rep. DEC-TR-01, Digital Equipment Corportation, https://www1.cse.wustl.edu/jain/papers/ftp/fairness.pdf.

Junxiao, S., Afanasyev, A., Pesavento, D., Newberry, D., Schneider, K., Liang, T., [NDNLPv2] — NFD. https://redmine.named-data.net/projects/nfd/wiki/NDNLPv2.

Kutscher, D., Ohlman, B., Oran, D., 2012. Information-Centric Networking Research Group. Internet Research Task Force, https://irtf.org/icnrg.

Li, Z., Shen, X., Xun, H., Miao, Y., Zhang, W., Luo, P., Liu, K., 2023. CoopCon: Cooperative hybrid congestion control scheme for named data networking. IEEE Trans. Netw. Serv. Manag. http://dx.doi.org/10.1109/TNSM.2023.3262198, Conference Name: IEEE Transactions on Network and Service Management.

Mahdian, M., Arianfar, S., Gibson, J., Oran, D., 2016. MIRCC: Multipath-aware ICN rate-based congestion control. In: Proceedings of the 3rd ACM Conference on Information-Centric Networking, ACM-ICN '16. Association for Computing Machinery, New York, NY, USA, pp. 1–10. http://dx.doi.org/10.1145/2984356.2984365.

Mathis, M., Semke, J., Mahdavi, J., Ott, T., 1997. The macroscopic behavior of the TCP congestion avoidance algorithm. ACM SIGCOMM Comput. Commun. Rev. 27 (3), 67–82. http://dx.doi.org/10.1145/263932.264023.

Nichols, K., Jacobson, V., 2012. Controlling queue delay. Commun. ACM 55 (7), 42–50. http://dx.doi.org/10.1145/2209249.2209264.

Nichols, K., Jacobson, V., McGregor, A., Iyengar, J., 2018. Controlled delay active queue management. RFC 8289, http://dx.doi.org/10.17487/RFC8289.

Nikmard, B., Movahhedinia, N., Khayyambashi, M.R., 2022. Congestion avoidance by dynamically cache placement method in named data networking. J. Supercomput. 78 (4), 5779–5805. http://dx.doi.org/10.1007/s11227-021-04080-0.

Qin, J., Xing, Y., Wei, W., Xue, K., 2020. Edge computing aided congestion control using neuro-dynamic programming in NDN. In: GLOBECOM 2020-2020 IEEE Global Communications Conference. (ISSN: 2576-6813) pp. 1–6. http://dx.doi.org/10.1109/GLOBECOM42002.2020.9322365.

Qu, D., Wu, J., Zhang, J., Gao, C., Shen, H., Li, K., 2023. Efficient congestion control scheme based on caching strategy in NDN. J. Netw. Comput. Appl. 216, http://dx.doi.org/10.1016/j.jnca.2023.103651.

Rodríguez Pérez, M., 2023. Simulation scenarios for a joint AQM and congestion control algorithm. https://github.com/ICARUS-ICN/jaqmcc.

Rozhnova, N., Fdida, S., 2012. An effective hop-by-hop Interest shaping mechanism for CCN communications. In: 2012 Proceedings IEEE INFOCOM Workshops. pp. 322–327. http://dx.doi.org/10.1109/INFCOMW.2012.6193514.

Schneider, K., Yi, C., Zhang, B., Zhang, L., 2016. A practical congestion control scheme for named data networking. In: Proceedings of the 3rd ACM Conference on Information-Centric Networking, ACM-ICN '16. Association for Computing Machinery, pp. 21–30. http://dx.doi.org/10.1145/2984356.2984369.

Song, S., Zhang, L., 2022. Effective NDN congestion control based on queue size feedback. In: Proceedings of the 9th ACM Conference on Information-Centric Networking. ACM, Osaka Japan, pp. 11–21. http://dx.doi.org/10.1145/3517212.3558088.

Thibaud, A., Fasson, J., Arnal, F., Sallantin, R., Dubois, E., Chaput, E., 2020. Cooperative congestion control in NDN. In: ICC 2020-2020 IEEE International Conference on Communications (ICC). pp. 1–6. http://dx.doi.org/10.1109/ICC40277.2020.9149034.

Trossen, D., Reed, M.J., Riihijärvi, J., Georgiades, M., Fotiou, N., Xylomenos, G., 2015. IP over ICN — The better IP? In: 2015 European Conference on Networks and Communications (EuCNC). pp. 413–417. http://dx.doi.org/10.1109/EuCNC.2015.7194109.

Wang, M., Yue, M., Wu, Z., 2018. WinCM: A window based congestion control mechanism for NDN. In: 2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN). pp. 80–86. http://dx.doi.org/10.1109/HOTICN.2018.8606039.

Wu, F., Yang, W., Sun, M., Ren, J., Lyu, F., 2021. Multi-path selection and congestion control for NDN: An online learning approach. IEEE Trans. Netw. Serv. Manag. 18 (2), 1977–1989. http://dx.doi.org/10.1109/TNSM.2020.3044037.

Yang, J., Chen, Y., Xue, K., Han, J., Li, J., Wei, D.S.L., Sun, Q., Lu, J., 2022. IEACC: An intelligent edge-aided congestion control scheme for named data networking with deep reinforcement learning. IEEE Trans. Netw. Serv. Manag. 19 (4), 4932–4947. http://dx.doi.org/10.1109/TNSM.2022.3196344.

Ye, Y., Lee, B., Flynn, R., Murray, N., Fang, G., Cao, J., Qiao, Y., 2018. PTP: Path-specified transport protocol for concurrent multipath transmission in named data networks. Comput. Netw. 144, 280–296. http://dx.doi.org/10.1016/j.comnet.2018.08.002.

Ye, Y., Lee, B., Flynn, R., Xu, J., Fang, G., Qiao, Y., 2021a. Delay-based network utility maximization modelling for congestion control in named data networking. IEEE/ACM Trans. Netw. 29 (5), 2184–2197. http://dx.doi.org/10.1109/TNET.2021.3090174.

Ye, Y., Lee, B., Qiao, Y., 2020. Hop-by-hop congestion measurement and practical active queue management in NDN. In: GLOBECOM 2020-2020 IEEE Global Communications Conference. (ISSN: 2576-6813) pp. 1–6. http://dx.doi.org/10.1109/GLOBECOM42002.2020.9322585.

Ye, J., Leung, K.-C., Low, S.H., 2021b. Combating bufferbloat in multi-bottleneck networks: Theory and algorithms. IEEE/ACM Trans. Netw. 29 (4), 1477–1493. http://dx.doi.org/10.1109/TNET.2021.3066505.

Zafar, H., Abbas, Z.H., Abbas, G., Muhammad, F., Tufail, M., Kim, S., 2020. An effective fairness scheme for named data networking. Electronics 9 (5), 749. http://dx.doi.org/10.3390/electronics9050749.

Zeng, L., Ni, H., Han, R., 2021. The yellow active queue management algorithm in ICN routers based on the monitoring of bandwidth competition. Electronics 10 (7), 806. http://dx.doi.org/10.3390/electronics10070806.

Zhang, L., Afanasyev, A., Burke, J., Jacobson, V., Claffy, K., Crowley, P., Papadopoulos, C., Wang, L., Zhang, B., 2014. Named data networking. Comput. Commun. Rev. 44 (3), 66–73. http://dx.doi.org/10.1145/2656877.2656887.

Miguel Rodríguez-Pérez received the M.Sc. and Ph.D. degrees in telecommunication engineering from the University of Vigo, Spain, in 2001 and 2006, respectively. He is currently an Associate Professor with the Department of Telematics Engineering, University of Vigo, where he is also an Affiliated Member of the co-located Networking Laboratory. He has published a book and coauthored over 45 conference and journal papers. His research interests include congestion control and traffic engineering, with a strong focus on energy efficiency.

Dr. Sergio Herrería Alonso received the M.Sc. and Ph.D. degrees in telecommunication engineering from the University of Vigo, Spain, in 2001 and 2006, respectively. He is currently an Associate Professor with the Department of Telematics Engineering, University of Vigo, where he is also an affiliated member of the colocated Networking Laboratory. His research interests include quality of service in the Internet, the performance analysis of computer networks and energy-efficient networking. He has authored over 30 papers in peer-reviewed international conferences and journals, most of them with a high impact factor (Q1 and Q2 quartiles) in the "WOS-Journal Citation Report". He has an h-index value of 11 (Scopus).

J. Carlos López Ardao has been a professor at the School of Telecommunications Engineering of the University of Vigo since 1990 (University Holder since 2000). His field of research has always been related to the modeling and performance analysis of communications networks and traffic engineering, although in recent years he has also begun to work in the field of subjective quality in video transmission applications and in the application of technology towards educational innovation (Social Learning, Learning Analytics, Gamification, Flipped Learning, etc.). From 1993 to date he has been a collaborating researcher in 8 consecutive projects of the National R & D Plan, two of them coordinated. He is coauthor of 22 scientific articles published in journals indexed in the Journal Citation Report (of Q1 and Q2 ranges, the majority) and of more than 50 communications to congresses. Currently, he is a member of the doctoral program in information and communication technologies at the University of Vigo, and he has directed or co-directed 3 doctoral theses. He is a co-founder of the technological spin-off of the University of Vigo, SocialWire Labs S.L., dedicated to the development of educational innovation technologies on social networking environments.

Raúl Rodríguez Rubio received the M.Sc. and Ph.D. degrees in telecommunication engineering from the University of Vigo, Spain, in 1991 and 2000, respectively. He is currently an Associate Professor within the Department of Telematics Engineering, University of Vigo, where he is also an affiliated member of the co-located Networking Laboratory. His research interests include quality of service in Internet, performance analysis of computer networks, and Cybersecurity. He has authored [over 20 papers in peer-reviewed international conferences and journals], some of them with a high impact factor (Q1 and Q2 quartiles) in the "WOS-Journal Citation Report".