Flow-aware traffic control for a content-centric network

Sara Oueslati
Orange Labs, France
sara.oueslati@orange-ftgroup.com

James Roberts, Nada Sbihi
INRIA, France
{james.roberts,nada.sbihi}@inria.fr

Abstract—The content-centric networking (CCN) paradigm proposed by PARC holds considerable promise as the architecture for the future Internet but remains incomplete. In this paper we argue that it is necessary to supplement CCN with mechanisms enabling controlled sharing of network bandwidth by concurrent flows. Traffic control is necessary to ensure low latency for conversational and streaming flows and to realize satisfactory bandwidth sharing between elastic flows. These objectives can be realized using “network neutral” traffic controls based on per-flow fair bandwidth sharing. The paper describes necessary CCN mechanisms and discusses the strategies to be implemented by end systems and routers. A simulation-based performance evaluation demonstrates the effectiveness of the proposed mechanisms and the impact of possible strategies. We describe our experience in implementing some of the mechanisms using the CCNx prototype software.

I. INTRODUCTION

Jacobson and colleagues at Palo Alto Research Center (PARC) have argued convincingly that the Internet should be re-designed to facilitate content dissemination [7]. Their proposed content-centric networking (CCN) paradigm would bring significant advantages, notably with respect to security, mobility and efficiency. Interest in CCN is growing, notably through NSF Future Internet Architecture Award project, Named Data Networking [19]. Parallel efforts, like the European projects Pursuit and Sail1, are aiming to define alternative information-centric network architectures.

The definition of CCN is incomplete. In particular, traffic control functions have hardly been considered. Such controls are essential, however, if only to ensure that packets of voice calls experience negligible delay when they share links with high speed downloads or that greedy users cannot gain an unfair share of bandwidth by requesting downloads at a rate that is too high. In this paper we propose a CCN traffic control framework based on per-flow fair bandwidth sharing, identifying original mechanisms and evaluating possible end-system and router forwarding strategies.

CCN flows can be reliably identified on-the-fly from the object name included in Interest and Data packets. As has previously been argued in the context of IP, we claim that imposed per-flow fair sharing is then sufficient to realize implicit service differentiation and meet performance requirements. The paper aims to justify this claim taking account of the particularities of CCN. As well as proposing various router mechanisms to control sharing, we postulate a CCN compatible charging scheme that provides the necessary incentive for operators to invest in transmission and storage capacity. Simulation results demonstrate that the control framework is effective and reveal the impact of possible end-system and router forwarding strategies. A small-scale testbed has been used for proof-of-concept experiments.

Of course, Internet traffic control is a vast subject that continues to stimulate a huge amount of research. It is impossible in one paper to respond completely to all the issues that have been raised. Our limited ambition is to put the case for some original CCN mechanisms and to argue that they are compatible with rational user and operator strategies. Others will surely prefer alternative traffic control frameworks where, for instance, bandwidth is shared under TCP-like congestion control and/or routers distinguish multiple classes of service. We look forward to comparing our proposals with such alternatives, when they emerge.

In the next section we recall salient features of CCN that are necessary for our discussion. Section III presents the mechanisms required to realize the proposed traffic control framework. End-system and router strategies in emitting or forwarding CCN Interests are discussed in Section IV. The results of a preliminary performance evaluation and an account of our experimental implementation are presented in Section V.

II. CCN AND TRAFFIC CONTROL

We recall some features of CCN that are important for traffic control. A full description is provided in the CCN paper [7].

A. Naming

Packets in CCN are of two types: Interests and Data. Each Data packet carries a payload of several kilobytes, preceded by a header consisting of a unique name followed by signatures and other authentication data. A Data packet is returned in response to an Interest packet bearing the same name.

---

1www.fp7-pursuit.eu, www.sail-project.eu

Fig. 1. CCN naming with the notion of object name
A summary representation of the hierarchical name structure envisaged in CCN is depicted in Fig. 1. The chunk name uniquely defines the Data packet. We suppose the chunk name can be parsed to determine an “object name” composed of prefixes that are common to all chunks relating to a given object (including version number, etc.). The object in question might be a video stream, a voice signal in a phone call or a stored document. The field noted “chunk number” in Figure 1 might for instance be the sequence number for a stream or a label specifying the part of a stored document.

B. Forwarding

Forwarding is performed using the following structures implemented in a CCN router: a forwarding information base (FIB), a pending interest table (PIT) and a content store (CS). When an Interest is received on some face \( f \), a CCN router:

1) checks the CS and if the chunk is present returns a copy via \( f \),
2) if not, checks if the PIT has an entry for this chunk and, if so, adds \( f \) to the entry,
3) if not, creates a new PIT entry containing \( f \) and forwards the Interest on the faces indicated by the FIB for that chunk name.

When a Data packet is received, the router:

1) returns the chunk over all faces indicated by the corresponding PIT entry and deletes that entry,
2) if appropriate, stores a copy of the chunk in the CS.

![CCN network highlighting path user \( U_1 \) and source \( S_2 \)](image)

Figure 2 depicts part of a CCN network. Suppose user \( U_1 \) emits Interests for a given object to its access node \( A \) which forwards them to \( B \) which in turn knows it can retrieve the required chunks from source \( S_1 \). Interests follow the path from \( U_1 \) to \( S_1 \) via \( A \) and \( B \). Each Data chunk is returned over the reverse path, i.e., from \( S_1 \) to \( U_1 \) via \( B \) and \( A \).

This is a simple forwarding example. In CCN, it is possible that \( B \) splits received Interests for a given object between two possible sources, \( S_1 \) and \( S_2 \). Similarly, the same object might be requested simultaneously by more than one user. In the figure, \( B \) receiving two Interests from \( U_1 \) and \( U_2 \) for a given chunk at about the same time would forward only one copy towards an appropriate source.

Source \( S_1 \) could be a server, another user or the CS of a CCN router. CCN naturally realizes multicast trees and brings popular objects close to users by the systematic use of caching.

Transport is mainly governed by the rate at which Interests are emitted. This rate is determined by the user but can be reduced if necessary at any intermediate router.

C. Traffic control in CCN

Congestion control in IP relies jointly on buffer management and end-system behaviour. The current CCN specification does not discuss buffer management and, in fact, the notion of a buffer that is distinct from the CS is not even recognized. End system behaviour is hardly discussed in the paper [7] either, while the prototype implementation CCNx includes only primitive algorithms to determine a “pipeline” of outstanding Interests. Both aspects of congestion control thus remain to be specified in CCN.

CCN interestingly defines a strategy layer immediately below the “chunk layer” at the hourglass waist (cf. Fig. 1 in [7]) and just above the network layer. This layer offers the possibility to adapt Interest forwarding depending on observed network congestion, among other criteria. It would be possible, for instance, to modify FIB entries depending on measured performance over alternative paths. It largely remains to specify and evaluate candidate forwarding strategies.

The envisaged systematic use of caching in CCN is intended to realize a favorable tradeoff between memory and bandwidth. It therefore will have a significant impact on the volume and distribution of network traffic. On the other hand, whatever the resulting distribution of traffic demand, it remains necessary to perform traffic control wherever congestion can arise. Caching and traffic control, as considered in this paper, are thus largely orthogonal functions.

D. Network faces and user faces

Since the users expressing Interests are known only at the access node, resource sharing objectives and buffer management policies depend on the position of the face in question. We distinguish “network faces”, situated between CCN nodes, and “user faces” through which packets are exchanged between users and their access node.

Since user faces are likely to be the main bottleneck for most CCN flows, it is desirable that bandwidth sharing here take proper account of the performance requirements of these flows. It is also possible at such faces that user-network signalling and appropriate schedulers be implemented to fulfill these requirements. However, on network faces, the only criteria for controlled bandwidth sharing must derive from the name and any associated metadata included in CCN packet headers. In this paper we only consider traffic controls applied to network faces.

III. BANDWIDTH SHARING MECHANISMS

This section presents some original mechanisms as well as some original adaptations of known mechanisms.

A. CCN flows

The hierarchical CCN naming convention described in Sec. II-A lends itself to the identification of flows. A CCN flow
consists of packets bearing the same object name. Flows are identified on-the-fly by parsing the packet headers. It is generally sufficient to identify flows locally at a given face and to maintain minimal state only when the packets of the flow are awaiting emission over that face.

We do not explicitly distinguish flows that correspond to point-to-point, multipoint-to-point or point-to-multipoint communications since we only envisage mechanisms that act independently on the single packet stream with the same object name observed at some face.

B. Separating cache and buffer

The statement in [7] that FIFO queues in routers are replaced by caches with LRU replacement is simplistic. As noted in Section II-C, buffer management is an important component of traffic control. There is a mismatch between the characteristics of a buffer and the CS that precludes performing traffic control directly via the occupation of the latter. Buffers need very rapid access and are typically small. The CS, on the other hand, can be built with slower, cheaper memory and needs to be very large.

The CS reduces network traffic by locally storing frequently demanded objects. The required memory capacity for a given traffic reduction depends on the size and popularity distribution of the object population. To simplify, we assume the applied caching policy is ideal in the sense that the cache contains only the most frequently requested objects (note that this can be achieved practically using policies proposed in [8]).

Consider two types of application that together currently generate the majority of Internet traffic: user generated content exemplified by YouTube, and peer-to-peer file sharing exemplified by BitTorrent. We assume popularity for these applications is characterized by a Zipf law: the rate of demands for the \( i \)th most popular object is proportional to \( i^{-\alpha} \). Measurements reported in [5, 6], for example, suggest this is not unreasonable and that a typical value of \( \alpha \) is around .75 for both types of content\(^3\).

Assuming a population of \( N \) objects and ideal caching, the hit rate \( h(x) \) for a cache of size \( xN \) objects, \( 0 \leq x \leq 1 \), is

\[
    h(x) = \sum_{1 \leq i \leq xN} i^{-\alpha} / \sum_{1 \leq i \leq N} i^{-\alpha} \to x^{1-\alpha} \text{ as } N \to \infty.
\]

Now consider what this implies for YouTube videos and BitTorrent swarms. We suppose there are at least 100 million YouTube videos and that their mean size is 4 MB [4]. This yields a required cache of 640 GB for a hit rate of 20% or 25 TB for a hit rate of 50%. At the time of writing, the Demonoid BitTorrent tracker site records some 400000 active swarms with an average object size of more than 7 GB. Hit rates of 20% and 50% for this application require 4 TB and 175 TB, respectively. We conclude that the CS will need to be very big to significantly reduce network traffic.

On the other hand, IP router buffers are considered huge to significantly reduce network traffic. We conclude that the CS will need to be very large.

C. Flow-aware networking principles

We believe a lightweight flow-aware networking paradigm is well-adapted to CCN. This consists essentially in realizing two functions: 1, router imposed per-flow fair bandwidth sharing, and 2, overload control.

We consider dynamic traffic where flows occur at the instants of a stationary random process (e.g., a Poisson process) and persist for a certain finite time. This time might be an intrinsic flow characteristic, as for a telephone call say, or result from the mean throughput an elastic, content retrieval flow acquires. Traffic demand is the product, flow arrival rate \( \times \) mean flow size. Overload occurs when this demand (in bit/s) approaches or exceeds link capacity leading therefore to unacceptable performance for some flows.

The significant advantages of imposing fair sharing are well-known (see, for example, [12]–[14], [16]):

1) end-systems are relieved from the obligation to implement “TCP friendly” congestion controls
2) only flows whose rate exceeds the fair rate experience significant packet loss and delay.

The expected fair rate of a link of capacity \( C \) and load \( \rho \) is approximately \( C(1-\rho) \) [2]. On normally loaded network links (e.g., a 10 Gb/s link with load less than 90%), this is much greater than the rate of conversational and streaming flows. Their packets consequently experience negligible delay and loss thus realizing implicit service differentiation.

Realizing fairness is clearly subject to being able to correctly identify flows and it is necessary to carefully examine the scope for abuse that could arise in CCN. We note, however, that the bandwidth of network faces is assumed to be large and the fair rate sufficiently high that very few flows would be able to saturate the residual capacity however users might seek to disguise them.

Fairness is less an objective in itself than an expedient to realize the above two advantages. Imposing fairness is a service allowing high bandwidth capable applications to safely exploit residual capacity without the need to implement specified end-system behaviour.

\(^3\)knowing the precise distribution is clearly important for cache sizing but not for the present “ball park” estimates
D. Implementing fair sharing

We advocate the implementation of a scheduling algorithm like DRR [15] with packets dropped from the longest queue on buffer overflow. Per-flow queues in DRR are materialized as simple linked lists referenced through a structure called ActiveList (see pseudocode in [15]). Only flows that currently have a packet in the buffer are included in ActiveList and the number of such flows determines the performance of the algorithm. The scalability of DRR derives from the fact that the number of active flows is broadly independent of link speed. By identifying flows through a hash of their object name, using so-called stochastic fairness queuing, the fact that the population of active flows changes rapidly brings no performance penalty.

It was shown in [10] that the distribution of the number of flows in ActiveList is broadly independent of link capacity C. The analysis in the cited paper is complicated but the intuition behind the affirmation that fair sharing is scalable is quite simple. The vast majority of flows have an intrinsic rate, determined by a codec or a bottleneck elsewhere on their path, that is orders of magnitude smaller than C. When all flows have such low intrinsic rates, queues arise only due to near simultaneous packet arrivals from distinct flows. Even at high load (up to 90%, say), the queue length remains relatively small (no more than 100, say, with high probability).

Suppose now, on the other hand, that all flows have an intrinsic rate greater than C and share bandwidth fairly. Under very general traffic assumptions, the probability distribution of the number of flows in progress is known and depends only on the load [2]. As above, even at high load, this number remains relatively small.

In practice, flows in progress have a mix of intrinsic rates. The analysis in [10] shows that the number of flows that need scheduling remains less than a few hundred for any mix, even when the number of flows in progress is counted in hundreds of thousands. The most likely mix is a very large number of low rate flows, together counting for a large fraction of C, with a handful of high rate flows dynamically sharing the residual capacity.

E. Overload control

Fair sharing, by whatever mechanism, is scalable only at normal loads that for present purposes we suppose are loads less than 90%. Above this, the number of flows to be managed can become large. Above a load of 90%, in any case, performance is generally unsatisfactory. Both to ensure scalability and to preserve performance, it is necessary to implement some form of overload control.

Admission control whereby, at the onset of congestion, new flows are rejected to preserve the performance of flows in progress, is onerous to implement since it is necessary to maintain the large list that identifies the flows in progress. In addition, the design of robust and efficient admission control algorithms has proven problematic, to say the least.

On the other hand, overloads should be very rare in a well-managed network. The most likely cause of overload is a failure somewhere in the network bringing a surge in demand on links of a backup path. In this case, flows that appear “new” to the overloaded link are in fact the continuation of flows that were in progress on the broken path. To reject such flows is actually to interrupt a flow in progress.

We suggest, therefore, that CCN overload control should consist simply in discarding the packets of a suitably large subset of flows in progress. The subset might be determined arbitrarily by applying a hash function to the flow identifier, say. This is a quite drastic, reactive rather than preventive control. It may be considered akin to preferring rolling blackouts to a general brownout in an overloaded power network. Alternatives may be envisaged if it is possible to identify and preferentially discard “less important” flows (e.g., not interrupting multicast TV flows viewed by thousands).

F. Economic incentives

It is important to establish a viable business model for CCN providing sufficient incentive for providers to invest in this new architecture. Satisfactory network performance depends above all on there being sufficient capacity to satisfy demand. We make a modest proposal intended to ensure adequate return on investment and compatibility with proposed traffic controls.

We suppose providers are paid for delivered Data packets. In Figure 2, user U1 pays P1 for data delivered from A; P1 pays P2 for data delivered by S1. Emitting an Interest amounts to buying the corresponding Data. Providers must install sufficient infrastructure to be able to complete the sale. Providers also have an incentive to cache content since, for example, P1 will not wish to pay P2 for repeated transmission of the same object.

This proposal just defines the direction of charging. Fees would have to cover the cost of infrastructure but their precise nature could take many forms including flat rates and peering arrangements.

An implication of this choice is that users are not penalized for requesting more data than the network can handle. On the other hand, a network provider like P1 in Fig 2 would not wish to “buy” more data from provider P2 than it can “sell” to its own customers, as determined by the current transport capacity of link BA in this example. This means providers have an incentive to participate in congestion control by regulating the upstream flow of Interests, as discussed next.

G. Interest discard

CCN offers a complementary means to control bandwidth sharing on network faces. It has the particularity that Interests
and Data visit exactly the same network nodes. Figure 3 illustrates the flow of Interests and Data in both directions over link \(AB\) of Figure 2. \(U_a, U_b\) and \(S_a, S_b\) represent users and sources, respectively, for two representative flows. In CCN, as previously noted, each flow can have multiple users and multiple sources. We only show one of each to keep the drawing simple.

Network providers have an incentive to control the flow of Interests. Suppose there is congestion between \(B\) and \(A\). \(B\) would wish to limit the flow of Interests to source \(S_a\) to avoid having to “buy” packets that it is not able to “sell” to \(U_a\) via \(A\). The diamond in the figure represents a mechanism that suitably shapes the flow of Interests as discussed below. Note that actions on Interests and scheduling take place in the same line card, facilitating implementation. Interests are dropped or delayed before being processed by CCN forwarding functions.

We have envisaged several options for selectively dropping Interests and/or pacing their emission. At one extreme, it is possible to control sharing on \(BA\) by pacing Interest flows to a “fair” rate at \(B\) (e.g., limiting the flow from \(B_{in}\) to \(B_{out}\) via \(S_a\)) and implementing a simple tail drop buffer at \(B_{out}\). This turns out to be unsatisfactory, however, since to ensure fairness on \(BA\) it would be necessary to pace an Interest flow to a rate that accounts for (unknown) flow-specific congestion between \(B_{in}\) and \(B_{out}\). Our preferred approach consists in applying fair sharing in the buffer at \(B_{out}\) and discarding Interests that are emitted at a rate in excess of the current fair rate.

Specifically, we implement a counter for each flow that is currently in the DRR \(ActiveList\). All counters are incremented by one quantum every time the DRR scheduler completes a cycle (i.e., when every flow queue in \(ActiveList\) has been emptied or received a quantum of service) up to a maximum value \(b\). The counter is decremented every time an Interest for the flow in question is sent from the line card to the CCN forwarding functions. This decrement is equal to the size of the requested Data packet. If when an Interest is received the counter value is less than this decrement, the Interest is discarded.

### H. Signalling congestion

In IP, the efficiency of TCP relies on the rapid detection of packet loss through out-of-sequence packet delivery. In CCN, where flows can have multiple sources and multiple destinations, the packet sequence is not respected and an analogy of “triple duplicate ACK” to detect loss is not possible. Current CCNx implementations use time outs to detect loss, both in the end-system and in routers that need to detect loss to remove corresponding PIT entries. It is not easy to set such time outs to a value that is small enough for timely reactions while avoiding false alerts for packets that are simply delayed.

An alternative to relying on time outs would be to explicitly signal packet loss. Our proposal is the following. When the DRR scheduler should discard a packet from the flow with the longest backlog, rather than rejecting the entire packet, only the packet payload is rejected. The header, including the name, is modified to signal the discard. This truncated packet is handled like a regular Data packet by routers on its path that therefore remove its PIT entry in a timely manner. The end-system identifies the loss immediately and can rapidly reemit the corresponding Interest and react accordingly to the congestion signal (e.g., by reducing the current transmission window). When an Interest is discarded as proposed in Section III-G, the line card would create a truncated Data packet with the chunk name to be returned directly over the reverse channel.

### IV. Interest forwarding strategies

We discuss how a user application should send Interests given the mechanisms described in the last section. We then consider different router Interest forwarding strategies with respect to flows that have multiple users or multiple sources. Lastly, we evaluate the impact of these strategies on the traffic capacity of the envisaged CCN traffic control framework.

#### A. User strategies

The resource sharing mechanisms proposed above in fact make network performance broadly independent of the way users emit Interests. It is nevertheless of interest to consider what would be the best strategy from the user’s point of view. We consider content retrieval applications where the transfer time depends on available bandwidth.

First note that the transport protocol is not required to realize any bandwidth sharing objective. Also, since transport in CCN is basically one-sided, there is actually no need for a formal communications protocol. It is necessary just to specify how the receiver should modulate the rate at which it issues Interests to meet the requirements of a given application.

One possibility is to emulate TCP by implementing additive increase, multiplicative decrease (AIMD) congestion control. The user maintains a window \(W\) of pending Interests (i.e., sent Interests for which the Data packet has not been received) and adjusts its size by adding \(\alpha/W\) to \(W\) for each Data packet received and decreasing \(W\) by a factor \(\beta\) whenever a loss is detected. The values of \(\alpha\) and \(\beta\) together with the round trip time between sending an Interest and receiving its Data determine how aggressively the user tracks available bandwidth, as is well-known from studies of TCP.

Since, according to the proposed charging scheme (Sec. III-F), users have no penalty for emitting more Interests than actually necessary, some might find it advantageous simply to emit Interests as fast as possible, at a constant rate say. The user thus ensures maximum throughput at the cost of having to manage a potentially large number of pending Interests.

Finally, we consider a simple fixed window strategy where users maintain a constant number of pending Interests. The size of the window determines complexity (the number of pending Interests to supervise) and clearly has an impact on realized throughput.

There are clearly many alternative strategies. We select the above three for a preliminary performance evaluation (see Sec. V) and to guide future research. As previously noted, given the imposition of fair sharing, the choice of transport protocol is not a critical issue for CCN.
**B. Router strategies**

We only consider router strategies in respect to the way they interact with our proposal to impose fair sharing. It seems clear that there are no specific CCN considerations to develop for flows between one source and one user. Complications may arise for multicast and multi-source flows.

1) **Multicast flows:** For real time multicasting, CCN naturally creates source rooted trees through the forwarding mechanism described in Sec. II-B. In the absence of overload, the rate of such a streaming flow is much smaller than the fair rate on any branch of the tree and the flow consequently receives excellent quality of service.

When users retrieve the same object but not exactly at the same time, it might be argued that their packets should not be considered as a single flow. For example, suppose user \( U_1 \) in Fig. 2 downloads an object from source \( S_1 \) in chunk order \( c_1, c_2, \ldots \) starting at some time \( t_1 \) while user \( U_2 \) requests the same object in the same order but starting at later time \( t_2 \). If \( t_2 \) happens before \( U_1 \) has finished downloading, there will be two streams of packets for the same object on link \( S_1 \).B. They should perhaps receive twice the bandwidth accorded to a single flow.

Fortunately, this objection to applying simple, unweighted fair queueing falls as long as router \( B \) is equipped with a cache. Even for a relatively small cache, chunks \( c_1, c_2, \ldots \) will still be present at \( B \) when \( U_2 \) requests them. There will never actually be two streams in progress on link \( S_1 \).

In Fig. 4a, users to the left of \( N_1 \) download objects stored in sources \( S_2 \) to \( S_4 \) as well as source \( S_1 \) co-located with \( N_1 \). We assume any given object is located in a number of sources (from 1 to 4). If the object is in \( S_1 \) it is downloaded directly from \( N_1 \). Otherwise \( N_1 \) spreads its Interests over all possible sources.

In Fig. 4b, objects are stored in 1 to 4 sources co-located with nodes \( N_1 \) to \( N_4 \) and are downloaded by users beyond the broad grey arrows. If the object is stored locally it is downloaded directly without using the depicted links. Otherwise, the origin node sends Interests on the two paths it sees. Paths may thus have either one or two links.

These toy networks are inspired by similar examples introduced in [9]. We discuss their traffic capacity in the next subsection.

2) **Multisource flows:** It may be advantageous for the forwarding strategy in a CCN router to split a flow of Interests between multiple sources. Work on multipath routing in IP has demonstrated that flow throughput and network traffic capacity are greater when congestion control is performed in a coordinated manner over the multiple paths [9]. Imposing fairness on each link as we propose prevents use of coordinated congestion control so it is important to understand the impact of this on CCN performance. We consider two toy network segments illustrated in Figure 4.

We assume the only throughput limitation is due to the depicted links. The sources are supposed to be coordinated so that they share the heavy load of caching popular content (cf. Sec. III-B). Objects are typically stored in multiple sources both for greater reliability and better throughput performance.

![Toy networks to illustrate use of multiple sources.](image)

Fig. 4. Toy networks to illustrate use of multiple sources.

**C. Traffic capacity**

The traffic capacity of a network is defined by its stability region, i.e., the set of constraints on demand that ensure the number of flows in progress does not go to infinity. We assume a worst case where all flows correspond to downloads.

1) **Single path forwarding:** First suppose all flows are point-to-point and follow a single path. This is the same configuration assumed for an IP network by Bonald et al. [3].

It was proved in that paper that traffic capacity is maximal under fair sharing however aggressive the behaviour of end users. This is a strong result that demonstrates the robustness of imposed fairness as a traffic control.

2) **Multipath forwarding:** We discuss capacity with respect to the toy networks of Fig. 4. Let \( r \) designate a multisource route, meaning the subset of sources and links used to download a certain object, and let \( \rho_r \) be the demand on this route (flow arrival rate \( \times \) mean flow size).

It is proved in [9] that the network of Fig. 4a is stable under the following necessary and sufficient conditions:

For all subsets of routes \( S \) we have

\[
\sum_{r \in S} \rho_r < \sum_{l \in \mathcal{L}(S)} C_l, \tag{1}
\]

where \( \mathcal{L}(S) \) is the set of links that are used by any flow in \( S \) and \( C_l \) are the link capacities. This stability region is the same as that realized by coordinated congestion control.

On the other hand, per-link fairness leads to loss of capacity for the example of Fig. 4b. First note that to use shortest paths only (in hops) reduces the network to a set of disjoint networks like that of Fig. 4a (with one link less) and thus yields maximal capacity. Allowing longer paths is advantageous at low load since throughput is then maximized. Coordinated congestion control, as described in [9], ensures a smooth transition between multipath routing at low load and shortest path routing at high load.

It is not possible to emulate coordinated congestion control when routers impose fair sharing. In this case, in heavy traffic, the fact that some flow paths have two links means each flow consumes more resources on average leading to capacity loss.
Fortunately, it is possible to remedy this problem by imposing the following congestion dependent Interest forwarding strategy. The router that is the mid-point of a 2-hop path monitors the state of the downstream hop (it can do this via the DRR scheduler). If the number of flows is above a threshold, it simply discards Interests on the 2-hop path so the network reverts to shortest path forwarding. The 2-hop path is used again when the number of flows drops below a second threshold.

We do not claim that is always possible to selectively discard Interests on long paths as in this toy network. We would rather propose that network topologies with shared content stores can and should be designed so that the advantages of fair queuing can be realized without capacity loss (e.g., using single hops as in Fig. 4a or by applying selective discard as in Fig. 4b).

V. SIMULATION AND EXPERIMENTAL RESULTS

We report simulation results on the performance of CCN enhanced with the mechanisms and strategies described above. We also discuss our preliminary implementation experience based on the CCNx prototype.

A. Fair queueing and user strategies

We simulated a simple dumbbell topology using a purpose built simulator written in Python using the package Simpy [11]. The bottleneck is a 10 Mbit/s link equipped with a 100 packet buffer. All packets are 1 KB in length. A 5 Mbit/s Poisson background traffic shares the link with a set of permanent flows that implement the different user strategies described in Sec. IV-A. The background traffic represents the superposition of many distinct low rate flows and is therefore described in Sec. IV-A. The background traffic shares the link with a set of permanent flows that implement the different user strategies described in Sec. IV-A. The background traffic represents the superposition of many distinct low rate flows and is therefore handled with priority by the DRR scheduler (i.e., background packets are served within the DRR cycle in which they arrive). The simulated permanent flows are as follows:

- **AIMD (7/8)** – the end system implements AIMD congestion control with a multiplicative decrease rate \( \beta = 7/8 \); increase rate is \( \alpha = 1 \) per round trip time (RTT) and RTT = 10 ms.
- **AIMD (1/2)** – the end system implements AIMD congestion control with \( \beta = 1/2 \) and \( \alpha = 1 \) with RTT = 200 ms.
- **CWIN (5)** – the end system maintains a constant window of 5 packets and RTT = 200 ms.
- **CWIN (100)** – the end system maintains a constant window of 100 packets and RTT = 200 ms.
- **CBR** – the end system emits Interests at a constant rate corresponding to a requested Data rate of 4 Mb/s.

Simulation results are summarized in Tables I and II. We distinguish results obtained with and without Interest discard and with loss detection by congestion notification (Sec. III-H), noted Rapid, and by a 1s time out, noted TO (1s). When Interest discard is used (Sec. III-G), parameter \( b \) is set to 10.

<table>
<thead>
<tr>
<th>Flow</th>
<th>No discard</th>
<th>Interest discard</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rapid TO (1s)</td>
<td>Rapid TO (1s)</td>
</tr>
<tr>
<td>AIMD (7/8)</td>
<td>1.20</td>
<td>1.24</td>
</tr>
<tr>
<td>AIMD (1/2)</td>
<td>0.19</td>
<td>0.19</td>
</tr>
<tr>
<td>CWIN (5)</td>
<td>1.20</td>
<td>1.24</td>
</tr>
<tr>
<td>CWIN (100)</td>
<td>1.20</td>
<td>1.24</td>
</tr>
<tr>
<td>CBR</td>
<td>1.20</td>
<td>1.24</td>
</tr>
</tbody>
</table>

**TABLE II**

<table>
<thead>
<tr>
<th>Flow</th>
<th>Loss and discard rates (L/D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIMD (7/8)</td>
<td>0.006/0</td>
</tr>
<tr>
<td>AIMD (1/2)</td>
<td>0.006/0</td>
</tr>
<tr>
<td>CWIN (5)</td>
<td>0.006/0</td>
</tr>
<tr>
<td>CWIN (100)</td>
<td>0.30/0</td>
</tr>
<tr>
<td>CBR</td>
<td>0.76/0</td>
</tr>
</tbody>
</table>

While short of a complete performance evaluation, these simulations of a simple case study justify the following observations.

Imposing fairness allows applications to implement aggressive congestion controls without undue impact on other flows. AIMD appears as an adequate control principle as long as the increase and decrease factors are sufficiently high. Interest discard effectively avoids routers requesting excess Data packets and does not adversely impact the other flows. The discard filter size \( b \) is not highly critical to performance and a value as small as 10 appears adequate. In our simulations, loss detection by timeout has a slight negative impact on performance compared to rapid loss detection. However, the main advantage of rapid detection is in reducing the time a lost packet remains as a pending Interest, both in the PIT and the end-system.

B. Multipath throughput

We apply Montecarlo simulation to evaluate the throughput performance of the toy networks of Fig. 4. Flow arrivals are Poisson and their size has an exponential distribution. Users
are assumed to emit Interests at unbounded rate so that the flow of Data packets from the servers is maximal. Fair queuing leads to the “min” allocation defined in [3] whose capacity is the same as max-min. All links have unit capacity and demand, expressed in this unit, includes flows served directly by the origin node.

Figure 5 shows realized throughput for the network of Fig. 4a under the assumption that source $S_2$ has twice as many objects as the others yielding asymmetric traffic. The results confirm that traffic capacity is indeed 5.29 units, as can be predicted by (1), and performance is excellent even for routes using busy server $S_2$ (labelled “high traffic”).

Fig. 5. Throughput v load for 2 out of 3 sources

For the square network of Fig. 4b, we assume symmetrical traffic: each flow chooses 2 out of 4 sources at random. Fig. 6 plots throughput for 3 router strategies. The strategy for the plot “shortest paths” is to use only single hop paths (some flows use two paths, others only one). Traffic capacity is then maximal (formula (1) predicts 16). The strategy for plot “dual path” systematically uses both short and long paths when the route so dictates. Its capacity is reduced to 12, a loss of 25% compared to shortest paths. On the other hand, this strategy yields 50% more throughput at low load. The third plot, “dual with blocking”, results from applying the strategy described in Sec. IV-C2. Thresholds are 2 and 1 meaning the long route is blocked if there is more than one flow on the downstream link and reopened only after that link next becomes idle. The curve demonstrates that it is possible to achieve maximum throughput and maximum capacity while implementing fair queuing schedulers.

C. Implementation in CCN

We have implemented CCN flow scheduling and the Interest discard mechanism on a simple testbed configuration running the CCN software of PARC (www.ccnx.org/). A full duplex link interconnects two Linux machines, one the server, the other the client.

Scheduling is implemented in kernel space on the server using a modified version of module $sch_{sfq}$ (part of the Linux traffic control $tc$ module) developed by L. Muscariello and P. Viotti that allows flows to be identified from a hash of the CCN name [17]. Interest discard is also implemented in the server kernel space. It requires the following updates:

- creation of counters in kernel space for each flow (actually, each sfq bucket),
- incrementing these counters on completion of each DRR cycle,
- discarding an Interest if the counter is too low,
- decrementing the corresponding counter each time an Interest is forwarded.

All Data packets in our set-up are constant size meaning we count packets and not bytes. The server implements a shaper (using $sch_{htb}$) to limit the link rate to 10 Mb/s.

We performed the following experiments. We launch the $ccnd$ daemon on the server and upload objects (large files) into the repository. We launch $ccnd$ on the client and update the FIB using $ccndc$ to locate these objects. We download objects using either the CCNx tool $ccncatchunks2$ or a simple program called $cbr$ that we developed to emit Interests at constant rate. Program $ccncatchunks2$ implements a “pipeline” allowing a certain maximum number of Interests to be pending at the client. The transmission window is adjusted when Data are received and when a loss is detected [17].

Figure 7 shows experimental results for two cases, one where the buffer is a simple FIFO queue and one where DRR is activated. The figure plots the per-second rate realized by two concurrent flows, as measured using Wire-shark (www.wireshark.org/). The top traces show that, with FIFO, the aggressive $cbr$ flow effectively interrupts the $ccncatchunks2$ flow when it starts at time 30. Equity is partially restored by DRR (bottom traces), though $ccncatchunks2$ (here with a pipeline of 3) is not sufficiently aggressive to attain the fair rate of 5 Mb/s and $cbr$ benefits from this.

For Figure 7, both DRR and Interest discard were enabled. Rate results are the same, however, when Interest discard is not used. The impact of Interest discard is in terms of Data loss and Interest discard rates. The following table shows these rates in 3 cases: without discard, with discard parameter $b = 10$ and with $b = 100$. 
We have proposed that CCN traffic control be based on router-imposed fair sharing between flows identified on-the-fly through the object name included in packet headers. A number of features and mechanisms would need to be added to the CCN paradigm to realize the proposed control framework.

It is first necessary to separately manage short, fast access buffers at router faces and the much larger and slower Content Store. Buffer management using a fair queueing scheduler like DRR is provably feasible and would ensure network performance is not vulnerable to adversarial user behaviour.

The proposed novel charging scheme, where Interests “buy” Data, ensures adequate return on investment for transmission and storage infrastructure. It defines the appropriate direction of charging, given the receiver-oriented nature of CCN flow control. Significantly, it encourages network operators to participate in congestion control by regulating the upstream flow of Interest packets through their routers. We propose a simple filter to be implemented in router line cards to perform selective Interest discard as necessary.

Since bandwidth sharing is controlled, performance does not depend critically on end-system strategies for sending Interest packets. A reasonably aggressive AIMD congestion control is shown in our simulation experiments to constitute an effective strategy, attaining the fair rate at the cost of low loss. The task of both end-system and CCN routers in maintaining the list of pending Interests is facilitated by the proposed mechanism that allows rapid detection of packet loss. This consists in returning just the header of a discarded packet.

One might suspect that fair sharing has an adverse impact on the performance of CCN flows with multiple sources and/or multiple destinations. Our analysis demonstrates that this is in fact not the case, as long the network implements identified intelligent caching and forwarding strategies.

Simulation of a simple case study illustrates the effectiveness of the proposed mechanisms and strategies. A more extensive performance evaluation is planned future work. Preliminary experience in implementing fair queueing and selective Interest discard in the CCx prototype is positive. It is the prelude to future, more wide-reaching experimentation.

ACKNOWLEDGEMENT

This work is partially funded by the French National Research Agency (ANR) through project CONNECT. We gratefully acknowledge valuable feedback from our partners in this project.

REFERENCES


<table>
<thead>
<tr>
<th></th>
<th>no filter</th>
<th>$b = 10$</th>
<th>$b = 100$</th>
</tr>
</thead>
<tbody>
<tr>
<td>loss</td>
<td>.42</td>
<td>.002</td>
<td>.005</td>
</tr>
<tr>
<td>discard</td>
<td>0</td>
<td>.45</td>
<td>.46</td>
</tr>
</tbody>
</table>

Results show Interest discard fulfills its intended role of avoiding needless Data retrieval. The $ccncatchunks2$ flow suffers no loss or discard in all three cases.

Fig. 7. Flow rates over time in sec: $cbr$ and $ccncatchunks2$ (continuous red line)