CCN & TCP co-existence in the future Internet: Should CCN be compatible to TCP ?

Stefan Braun, Massimo Monti, Manolis Sifalakis, Christian Tschudin
Department of Mathematics and Computer Science, University of Basel, Switzerland
stefan.braun@stud.unibas.ch, m.monti@unibas.ch, sifalakis.manos@unibas.ch, christian.tschudin@unibas.ch

***Abstract*—Content Centric Networking (CCN) proposes a clean-slate architecture as an alternative to current TCP/IP, which matches better current Internet use patterns. However, not much thinking has been invested yet on how this replacement could take place. We assume that the simplest and most attractive approach is a native co-existence of the two architectures in dual-stacks. In this case CCN's dynamics need to withstand TCP's aggressiveness in occupying network capacity, and at the same time not be overly aggressive itself, which would create instability to the current TCP-dominated Internet. Starting from this premise, we implemented an AIMD strategy for controlling the pipeline of Interests, which is compatible with TCP's behaviour. We test variants of this strategy against TCP on a native CCN deployment and we report on issues of such a strategy for the CCN philosophy. Our main observation is that such an approach has the potential to bring the two protocols to a (statistical) fair sharing of the capacity. However, unless strong assumptions are made, a temporal estimator (such as RTOs) for controlling the Interest pipeline is ill-suited for CCN's content multi-homing semantics. Unfortunately, such assumptions are not possible in lack of empirical usage and traffic patterns from a large-scale CCN deployment.**

I. Introduction

The Content Centric Networking (CCN) [1] architecture is an ambassador of information centric networking research [2], and proposes a clean-slate architecture that matches better the use patterns in the current Internet, than current TCP/IP does. Simple, elegant, and minimalistic, it redefines the key primitives at the "waist of the network" to include simple request-reply interactions and a publish-subscribe information distribution paradigm, which enable features such as loop free forwarding, location independence and flow balance. By casting away the explicit requirement for end-to-end communication and by enabling caching en-route, it makes content readily available from different locations and distances from the receiver. For a detailed description of the CCN protocol and how these features are embodied in its operation, we refer the reader to the seminal literature in [1], [3], [4], as well as the on-line technical forum of the CCN community.[1]

However, while most of the on-going work on CCN focuses on various operational aspects, little attention has been paid to explore how it may be adopted in today's Internet. On first thought possible options could be the following: (a) It may continue to run within TCP tunnels between CCN relays, pretty much like the current CCNx prototype does. However, this seems to defeat the purpose of introducing new features such as *flow balance in every hop*, offering little advantage over existing CDN (Content Delivery Network) technology. (b) It may be adopted natively and independently of TCP/IP in "separation domains" sliced across the current forwarding fabric, and using technologies such as OpenFlow, MPLS and differentiated services. In this case an administrative overhead is expected from the core network providers to cater for the "slices". (c) It may run natively on dual-stack configurations alongside TCP/IP (much like IPv4/IPv6 do). This last option is probably the most attractive, because it is simple (in terms of implementation), easy (does not require administrative intervention within the network to make it work) and potentially ubiquitous (existing traffic management measures may not need to distinguish between TCP and non-TCP/IP traffic). However, in this case CCN's protocol dynamics need to withstand TCP's aggressiveness in occupying network capacity, and at the same time not be overly aggressive itself, which would create instability to the current TCP-dominated Internet; at least for as long as both protocols are in use.

Focusing on this last attractive possibility, in this paper, we explore a strategy for CCN that exhibits TCP compatible dynamics. Specifically, we examine how effective it would be to adopt an Additive Increase Multiplicative Decrease (AIMD) and timeout-controlled pipeline of Interests as a strategy for receiver-driven flow control, since these are the two key-features that dictate TCP's dynamics today. We identify constraints, pitfalls and possible workarounds, and at the end of the paper we try to take a step back, see what we learned from this exploration, and draw some objective conclusions about the suitability and effectiveness of this approach.

A. Why AIMD and why timeout-based ?

By default the network is dumb and capacity sharing is (supposed to be) the result of competition among the TCP flows that collectively claim it, by regulating their rates through the well-known AIDM algorithm and based on time-estimators. But even in actual practice, whenever in-network resource management (e.g. for service differentiation) is adopted, this is tailored to influence TCP dynamics. This means that CCN "flows", in order to compete on equal terms with TCP over network capacities, should exhibit similar dynamics, which are reactive to time-estimators (and which can be similarly affected by in-the-network measures).

It is worth noting that CCN trades in-network memory (by means of caching) for bandwidth-delay product in long end-

[1]Currently accessible at https://www.ccnx.org/.

978-3-901882-50-0

to-end paths. Nevertheless, it still needs to be able to occupy and share with TCP the common path capacity to the nearest cache. At the same time it should not obstruct TCP flows from instantiating an end-to-end pipe across the common parts of their paths. In fact, as we will see later, when in-network memory is variable (in space and time), stability (and fairness) issues arise.

In our exploration in this direction, we were intrigued additionally by the fact that there are already a number of works in the current CCN literature on flow-control dynamics, which propose such an AIMD strategy even for a native CCN environment [5], [6], [7], [8].

The remaining of this paper is structured as follows: In section II, we overview the relevant reference behaviour of TCP, and we discuss some insights that underlie the engineering of our CCN strategy. In section III, we describe the base algorithm for the timeout-controlled AIMD strategy. In section IV, we introduce the topology and configuration for our tests and then in section V, we describe the tests, discuss the results and further refinements of the base algorithm. In section VI, we summarise insights derived from our observations, and refer to the relevant literature on the topic. Finally, section VII concludes the paper.

II. Considerations for engineering a TCP-like behaviour

TCP uses a congestion-based approach, whereby it temporarily causes congestion and losses, which allow it to claim quickly its capacity share and to sense its limits, respectively. Using an AIMD adaptive window of un-acknowledged segment transmissions, it maintains a continuous flow of data at the maximum possible throughput. Because of its end-to-end semantics, it relies on transmission acknowledgements (or lack thereof) and timeouts recorded at the sender for adjusting the transmission window. Any skews or noise in the time-based estimators that TCP uses can cause to abuse or underutilise a flow's capacity share. Such skews are relatively rare in wired environments and in cases where the path length does not change too often. Nevertheless, when skews do happen and lead to loss of synchronisation between sender and receiver, the *Fast-Retransmit/Fast-Recovery* mechanism tries to recover the lost synchronisation. When this is not possible, falling back to the *Slow-Start* phase serves as a last-resort "reset" button.

First of all, because in CCN the *flow balance* is maintained in every hop, there is no need for end-to-end flow-control semantics [1]. Congestion avoidance measures within the network analogous to those tailored for influencing TCP's transmission window scaling (and respectively its flow rate) may interact either indirectly, or explicitly, with the CCN receiver. In CCN a role similar to that of TCP's transmission window can be played by the pipeline of Interest[2] requests – dispatched and pending-for-content. A CCN receiver may use it to regulate its capacity occupancy/claim (against other CCN content transfers and possibly TCP flows). This was confirmed in early experiments that we conducted in [9].

Moreover, because by contrast to a transmission window the Interest pipeline is not emulating a "channel buffer", which preserves the ordering of transmissions and receptions, it does not suffer from synchronisation problems. There is therefore no (profound) need for pipeline synchronisation mechanisms, between the receiver and the content stores (CS).

On the other hand, omitting *Slow-Start* (as a synchronisation "reset" function), we need a way to compensate for the aggressiveness of TCP during their admission phase of a flow. Luckily, in regard to this, unlike TCP, where lost packets have to travel again the entire end-to-end path, in CCN when an Interest times out at a PIT (Pending Interest Table) along the delivery path, the recovery of a missing content chunk is usually only "delayed a little" and will most likely be found at an intermediate cache. This means that a re-request will have the chunk much faster delivered and the re-transmitted Interest will not have to travel again the entire path to the source. Taking advantage of this feature, we can enable a fast rump-up (or recovery) of a reduced pipeline after a timeout, and it allows us to be more time-elastic within a timeout period for re-transmitting Interests.

The challenge that still remains is how to regulate the scaling of the Interest pipeline, using a suitable (and analogous to TCP) timeout-based estimator. As we will see in the following, this estimation turns out to be not so trivial because of the dynamically varying path length to the content, as a result of caching and content multi-homing.

III. AIMD-driven Interest transmission strategy

We proceed to introduce the core parts of out timeout-driven AIDM flow-control strategy for the Interest pipeline of the CCN receiver.

1) Pipeline control: To implement the interest pipeline we use two data structures: One (the *sentList*) keeps a record of dispatched Interests that pend content reception (its length is the active size of the pipeline). The second one (*missingChunkList*) tracks the timed-out Interests. When, in the *sentList*, a timer for a specific Interest awaiting for its Content chunk expires, the respective Interest is moved to the *missingChunkList*. Following this event, the pipeline increment freezes and its nominal size is reduced according to multiplier *MDcoef* (AIMD logic). As other Interests in the *sentList* are satisfied, its size will reduce to match this nominal size. After that, the pipeline size can start increasing again stepwise (*AIstep*) at every successful reception, allowing new Interests to be dispatched. At this point however, previously expired Interests from the *missingChunkList* are prioritised for re-transmission over new Interests. The logic behind this is that they will be satisfied faster (because of previously en-path cached content) and therefore the pipeline size will rump-up faster. If in the meantime content-chunks of some expired Interests still arrive delayed, we simply remove them from the *missingChunkList*, so as not to congest the network with unnecessary re-transmissions.

[2]We employ the convention of capitalising the first letter in reference to Interest/Content, when we wish to denote the respective CCN packet types.

Algorithm 1 : Functions controlling the Interest pipeline

```
function INTERESTEXPIRED(seqNr)
    if mustBackOff.isTrue() then
        sentList.size = ceil(sentList.length * MDcoef)
        if sentList.size > 1 && interestsToSend.size > 0 then
            missingChunkList.push(seqNr)
        else
            sendInterest(seqNr)
    else
        missingChunkList.push(seqNr)
        sendNextInterest()
function RECEIVEDCONTENT(seqNr)
    if seqNr == lastSeqNr then
        finish()
    else
        updateTimeOutEstimator(RTTmeasure(seqNr))
        if expected(seqNr)&&!timer[seqNr].expired() then
            sendNextInterest()
            if sentList.isEmpty() then
                for i = 0 → AIstep do
                    sendNextInterest()
function SENDINTEREST(seqNr)
    timer[seqNr].set(requestTO)
    sentList.push(seqNr)
    interestTX(seqNr)
function SENDNEXTINTREST()
    if ! missingChunkList.empty() then
        sendInterest(missingChunkList.first())
    else
        if nextSeqNr++! = lastSeqNr then
            sendInterest(nextSeqNr)
function UPDATETIMEOUTESTIMATOR(rtt)
    err = rtt − sRtt
    sRtt += 0.125 * err
    rttVar += 0.125 * (abs(err) − rttVar)
    to = sRtt + 4 * rttVar
    requestTO = TOcoef * to
```

2) Time-out estimation: Associated with every Interest transmission, there is a timer set, upon expiration of which the request is considered timed out and moved from the *sentList* to the *missingChunkList*. The timer expiration value *reqTimeOut*, as in TCP, is based on a round trip time estimation (RTT) and takes into account the normalised over time RTT average (*sRtt*), its variance (*varRtt*), and a scale factor (*TOcoef*).

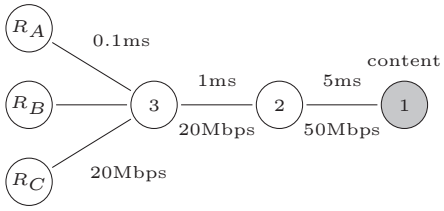
3) Counteracting TCP Fast Recovery: The TCP Fast-Recovery mechanism essentially "numbs" the window-adaptation for at least one RTT (from the reception of three duplicate ACKs and retransmission of a missing segment, until the first non-duplicate ACK). In that period, TCP prevents subsequent scaling-down of its transmission window. If everything goes well after that period, the window will re-start the additive increase, or otherwise a second reduction may occur immediately after Fast-Recovery. In our algorithm however, since individual chunk requests-receptions are independent of each other, subsequent timeouts can lead to subsequent back-offs of the transmission pipeline (until the correct RTT estimate is found). Although this does not pose problems next to other CCN flows, if we want to withstand the aggressiveness of TCP, we need to account for this. We thus provide an optional feature to de-activate the pipeline back-off for the duration of one estimated RTT immediately after a timeout.

The main components of the resulting AIMD strategy for controlling the Interest pipeline in CCN are given in pseudo-code in Algorithm 1. The parameters *AIstep*, *MDcoef* and *TOcoef* are used for tuning the sensitivity and responsiveness of the AIMD algorithm. Parameter *TOcoef* is used to influence the Interest time-out expiration timer in multiples or fractions of the standard TCP timeout estimator. The default value is set to 2, which corresponds to the time that three duplicate ACKs in TCP need to reach the sender. Parameter *MDcoef* is the factor for the multiplicative decrease of the Interest pipeline in fractions of the current pipeline. Last, parameter *AIstep* sets the step (number of interests) for the additive increase of the Interest pipeline. The empirically acquired optimal values for these parameters are given in Table I.

IV. Experimental Set-up

For our experiments we use the *OMNet++* network simulator with the INET framework (TCP/IP protocol suite) and the CCN Lite implementation of the CCNx protocol[3].

The employed network topology (below) contains a bottleneck link (l_{32}) over which flows from three client hosts must compete. All hosts support both TCP/IP and native CCN functionality. In all cases content is requested by the receiver hosts R_A, R_B and R_C. In case of TCP, data are retrieved from host 1 only, while for CCN multi-homing is possible by spreading content over any of the remaining nodes in the topology.



We fix the content chunk size in CCN as well as the MSS in TCP to 1400 bytes, and each file transfer consists of 5000 content chunks, or TCP segments. For TCP flow-control we use the TCP New Reno algorithm and the advertised window is set to its maximum of 65535 bytes. The link-layer queue size is set to 20 frames. This configuration guarantees that TCP's window variations are dictated only by network's traffic conditions, which result from the competition between the existing data flows. The bandwidth-delay product for one direction from host C to a receiver host equals: $BDP = 0.1ms{*}20Mbps{+}1ms{*}20Mbps{+}5ms{*}50Mbps = \underline{272Kbit}$

This set-up is simple enough to comprehend its characteristics, and at the same time it is sufficient to expose the problems and demonstrate the effectiveness of the engineered solution. Follow up tests also with more complex topologies confirmed the effectiveness of the refined version of the presented strategy.

[3]CCN Lite is a lightweight native implementation of the CCNx protocol developed by the University of Basel.

Parameter	Default Value
CCN: Content Chunk Size	1400 bytes
TCP: Maximum Segment Size	1400 bytes
TCP: Advertised Window Size	65535 bytes
TCP: Selective ACKs	enabled
TCP: Nagle's Algorithm	enabled
Buffer Size	20 frames
Bandwidth Delay Product	33.5 KByte
MDcoeff	0.4–0.5
AIstep	1
TOcoeff	2

TABLE I: Simulation parameters.

V. Test results and Analysis

In the following, we report on four sets of experiments. In the first three, we carry out an incremental validation of the CCN strategy (nominal behaviour, competition between CCN and TCP in an end-to-end environment, competition between CCN and TCP in a setting that includes multi-homing). In this process, we expose an important challenge that a receiver-based, timeout-controlled, AIMD strategy faces in case of multi-homed content. In the fourth set of experiments we describe and show how we addressed this challenge. Then, in Section VI we discuss in a more general context the suitability and compatibility of such flow-control strategies with other in-network measures in presence of multi-homed content.

As we are primarily interested in the behaviour of the two protocols, our comparison plots show (a) fluctuations of the TCP's transmission window and CCN's Interest pipeline, which depict the sensitivity and reactions of the algorithms, and (b) the performance of the strategy attained in terms of throughput – as sliding window averages.

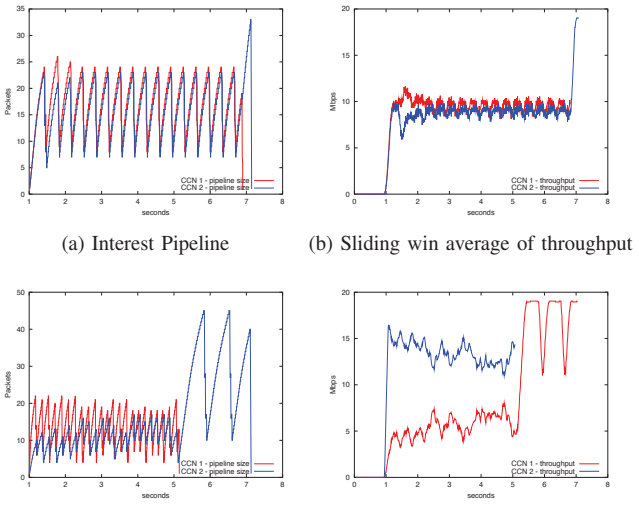
All plots show results of indicative individual tests (rather than averages over a number of runs), representing typical behaviours irrespective of the number of times a test is repeated.

A. CCN vs. CCN

A preparatory round of tests aims at confirming the assumption that the proposed AIMD strategy (introduced in Section III) sufficiently reproduces among CCN communications the same reference behaviour of competing TCP flows: in Figure 1.(a,b) we see the scaling of the transmission pipeline and respectively the throughput of two CCN communications across equal length paths. In Figure 1.(c,d), we see analogous plots when the two communications span across different path lengths (one acquiring its content from a nearer cache). As expected, the shorter path attains higher throughput since its Interest pipeline reacts faster to the availability of capacity.

B. TCP vs. CCN – Content available at a single point

In the second set of experiments we seek to see if the same set of parameters, that make CCN behave as TCP, can serve the "*fair*" capacity sharing when CCN competes against TCP. In Figure 2.(a,b) we report results when a CCN communication and a TCP flow acquire content from the same source (i.e. they have the same path length). There is an impressive alignment (synchrony) in the behaviour of the algorithms, which is



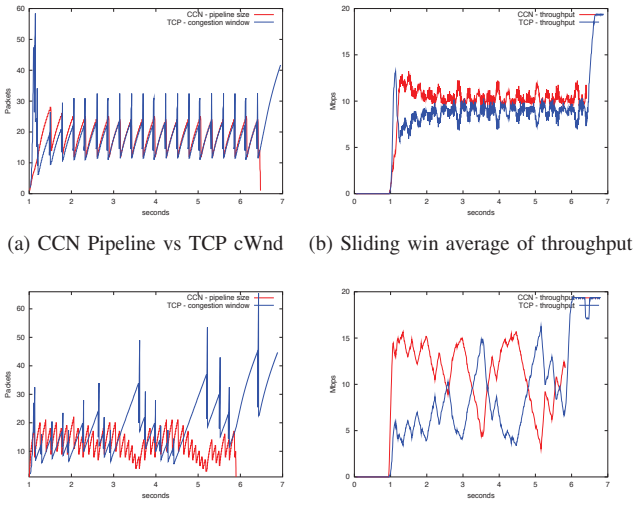
(a) Interest Pipeline (b) Sliding win average of throughput

(c) Interest Pipeline (d) Sliding win average of throughput

Fig. 1: Two CCN flows. In (a,b) both flows receive data from host 1. In (c,d) flow CCN 2 receives data from host 2, flow CCN 1 from host 1. (for *MDcoeff* = 0.5).

reflected in the variation of the transmission window/pipeline and the throughput (although not completely smooth, they still converge to equal shares).

When the path length is different (Figure 2.(c,d)), we observe a fierce competition between the two protocols, whereby they literally take in turns a throughput advantage rather than converging on stable proportional shares (even though in the final bill the overall shares are proportional)! However, we can recover the expected stability by setting the pipeline back-off parameter to 0.4.



(a) CCN Pipeline vs TCP cWnd (b) Sliding win average of throughput

(c) CCN Pipeline vs TCP cWnd (d) Sliding win average of throughput

Fig. 2: One CCN, one TCP flow. In (a,b) both flows receive data from host 1. In (c,d) CCN flow receives content from host 2, TCP flow from host 1. (*MDcoef=0.5*).

C. TCP vs. CCN – Multi-homed content

The last set of experiments involved the presence of multi-homed content in the network: For CCN, content may be delivered to a receiver from different locations, where it may have been previously cached (either along the main path or from intersecting paths to the source). In this case, the path lengths (and bandwidth-delay products) to the different content stores may differ substantially, such that a single estimator for Interest request timeouts is better expressed by a probability distribution, rather than a *continuous* average.

In order to understand how "fragile" the Interest timeout estimation is, with regard to the caching strategy, we conduct two additional tests with multi-homed content. In the first test we distribute the 5000 chunks of the CCN content among the CSs at nodes 1, 2, and 3, in batches sizes of 500 consecutive chunks (shown graphically in Figure 3). In the second test, we use the same content distribution pattern (repeated many times) but with batch sizes of 50 consecutive chunks. The estimation of the timeout will involve 8 and 80 respectively, abrupt changes in the RTT measurement. In both cases, TCP receives content only from node 1 (its RTO estimation being influenced only by the CCN traffic pattern).

The results in Figure 4.(a,b) refer to the 500 chunk batches and 4.(c,d) refer to the 50 chunk batches. Worse than anticipated, we observe that by using the current timeout estimator, CCN cannot benefit from the fact that 40% of the content is nearer to the receiver than for TCP. Moreover, the batch size (consecutive chunk ranges) has a dramatic impact: in Figure 4 (c,d) TCP outperforms CCN with much better flow completion time and higher throughput.

By requesting chunks in sequence, when switching over from one cache to another, the variation of the measured RTT gets often dominated by large steps and high frequency components. These cannot be easily filtered out and introduce substantial noise in the smoothed average. When switching over to a cache with a longer or more congested path, the last timeout estimation will lead with a very high probability to a timer expiration and thereby a back-off. If the window size falls under a lower limit, the network capacity can not be utilised any more. In the opposite case, when switching over to a cache closer to the receiver, the current pipeline may be too large for the bandwidth delay product of the new path. A congestion is then likely, causing losses and thus again a back-off. In either case, the situation is exploited by the TCP flow, which aggressively occupies the availed capacity immediately. The key factor seems to be *how often* a changeover between caches occurs during the retrieval of content (reflecting the *entropy* of the content chunks in the network), and irrespective of the absolute proportions of content among CSs. This means that the performance of a timeout-based flow-control strategy is tightly connected to the existing caching strategy in the network (and is likely a bad choice for scalability).

D. A more "deterministic" timeout estimation

An important difference between TCP's transmission window and CCN's Interest pipeline regards the ordering of

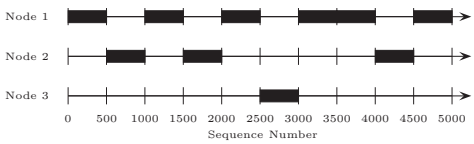
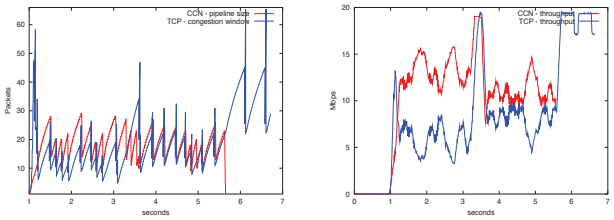
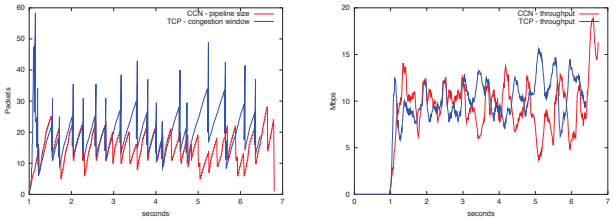


Fig. 3: Distribution of content-chunks per CS, shown as ranges of 500 chunks (called batch size). This distribution assigns 30% of the content to host 2, 10% to host 1 and 60% to host 3.



(a) CCN Pipeline vs TCP cWnd (b) Sliding win average of throughput



(c) CCN Pipeline vs TCP cWnd (d) Sliding win average of throughput

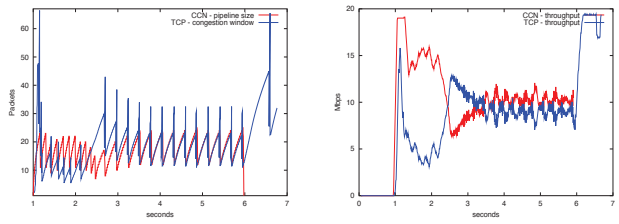
Fig. 4: One CCN and one TCP. CCN flow receives multi-homed content from different hosts (Fig. 3). In (a,b) the batch size is 500 chunks. In (c,d) the batch size is 50 chunks. TCP flow always receives data from host 1 (*MDcoef=0,5*).

requests. Because TCP models a stream, the transmission window slides along the sequence number space, such that segment requests are ordered. In CCN on the other hand, every Interest request is independent of every other and therefore the order in which chunks are requested and delivered does not matter. If one knows roughly where a batch of chunks is delivered from, then the Interests for those chunks can be associated (independently from all other) with a common RTT measurement and timeout estimation. Per batch RTTs are expected to have a smooth variance and timeout estimators are expected to produce more stable outputs over time. Building on this idea we extended our AIMD strategy as follows:

1) Every CS marks the Content packets it serves from its local cache with a unique identifier (per CS). Relayed content maintains the identifier of the originating CS.
2) When a CS sends the first chunk, from a continuous range of chunks (that belong to a part of content that it has cached), it stores in the packet the number of subsequent and consecutive chunks it has available.
3) The receiver keeps track of all CS identifiers it has seen and the respective content chunk ranges, and maintains an independent RTT measurement (and timeout estimation) for each. This information is sufficient for predicting with high probability a suitable timeout value

per Interest (depending on the cache store it is expected to be found). If a new Interest relates to a chunk that does not appear in a known range, then an averaged RTT measurement (over all known CSs) is used for the timeout estimation. The average is weighted by the number of chunks received from each content store.

Figure 5 presents the results for the extended AIMD strategy when applied in the previous scenario, with 50-chunk batch size ranges. This time the CCN communication is able to benefit from the shorter sub-paths (to nodes 1 and 2) and attains a shorter completion time and better throughput, than TCP. The algorithm exhibits a (statistical) determinism, which is captured in the pipeline scaling and throughput plots: between t=1s-2.3s most content is received from node 3 (nearest), between t=2.3s-3s most content is retrieved from node 2, and after t=3s the remaining content from node 1 is received.



(a) CCN pipeline vs TCP cWnd (b) Sliding win average of throughput

Fig. 5: One CCN and one TCP. CCN flow receives multi-homed content from different hosts (Fig. 3) using multiple RTT estimators. TCP flow always receives data from host 1 (*MDcoef=0,5*).

VI. Discussion and Related work

CCN promises novelty not only at the network level but also at the strategy layer (corresponding transport level). Flow control is not entirely end-to-end, but partly related to the content request strategy at the receiver end [5], [8], [7], and partly related to decisions made within the network (traffic shaping [10], [7], [11], and node adaptive forwarding [8]).

Similar to our work in this paper, the authors in [5], [8], [7] propose analogous timeout-driven AIDM flow-control strategies for the CCN receiver's Interest pipeline. In the experiments in [7] the authors have employed an AIMD strategy that uses TCP's RTO as a timeout estimator; similar to our baseline algorithm. As we show in this paper, however, this approach is not particularly effective against TCP in face of multi-homed content. In [5] on the other hand, the authors propose a weighted average timeout estimator based on a "recent" history of RTT measurements. It would be interesting to empirically test the effectiveness of this approach and compare its performance against our "deterministic" solution. It should be noted that none of these other approaches has been evaluated in the context of interaction with TCP and the one in [5] has not been tested empirically.

Our results, on one hand, entail some promise for our set-out goal (i.e. a strategy that empowers harmonious co-existence of CCN and TCP in a future Internet). On the other hand, as we observed, pursuing such a goal by homogenising the transport behaviour of the two protocols is not problem-free and complicates other engineering aspects whose aim is to substantiate the caching benefits in CCN [12], [6].

More specifically, the effects of multi-homed content are rather difficult to predict and control in a network where any router can be a cache, and thus a host of content. As we saw, in this case the *entropy* of content in the network plays an important role in the smoothing of RTT measurements, and thereupon in the robust estimation of timeouts. When the path delays to the different CSs vary substantially, the timeout estimation for Interests might not converge to a *continuous* average. Additionally, as multi-homed content out-of-main-path to the source will have direct impact on routing convergence, this implies a causal coupling between routing efficiency and the dynamic behaviour of receivers.

Based on the same line of thinking, in-network congestion avoidance measures for CCN that rely on adaptive queue management (AQM), like those proposed in [10], [7], should also be regarded with scepticism. Because such measures increase the delay variance across CSs, they can impact the performance of such timeout-dependent strategies much faster and stronger in CCN than in TCP. This problem is affirmed by the authors in [11], who propose a congestion avoidance measure that tries to avoid packet drops (queue overflows), and at the same time normalise delays in content delivery (i.e. smooth delay variance in the network). To achieve these goals they assume a back pressure mechanism based on which the network tries to proactively "shape" the Interest pipeline at the receiver. Independent preliminary experiments in [9] with back-pressure, based on active pacing [13], and explicit signalling [14], seemed quite effective and confirmed that such a receiver-network "collaboration" is probably the most effective congestion prevention and traffic shaping practice for CCN (and closer to its design philosophy).

VII. Conclusion

CCN's volatile strategy layer provides an option to engineer protocols with customised traffic dynamics, for serving different needs and service profiles. In this paper we hypothesised that the need to be served is the competition between CCN and TCP flows, as par example in a future large scale native deployment of CCN next to the current Internet protocols. Based on this assumption we designed a timeout-driven AIMD strategy for flow-control in CCN, which enables a CCN receiver to retrieve content by utilising the network capacity *ala*-TCP. The results of our experiments revealed a success potential for our quest, but at the cost of increasing implementation complexity in order to compensate for the fact that in CCN content may originate from different points in the network. This complexity is on one hand algorithmic, e.g. multiple estimators, and on the other hand resource-related, in terms of memory for protocol state, and processing at the end-node. In this direction the follow-up step is to profile this complexity as a function of the topology, and the distribution of content in the network.

REFERENCES

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. 5th international conference on Emerging networking experiments and technologies*, ser. CoNEXT '09, 2009, pp. 1–12.

[2] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26 –36, July 2012.

[3] V. Jacobson, D. K. Smetters, N. H. Briggs, M. F. Plass, P. Stewart, J. D. Thornton, and R. L. Braynard, "Voccn: voice-over content-centric networks," in *Proc. ACM Workshop on Re-architecting the Internet*, ser. ReArch '09. New York, NY, USA: ACM, 2009, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/1658978.1658980

[4] D. Smetters, "Securing network content," *Architecture*, pp. 1–7, 2009. [Online]. Available: http://www.parc.com/content/attachments/securing-network-content-tr.pdf

[5] G. Carofiglio, M. Gallo, and L. Muscariello, "ICP: Design and evaluation of an interest control protocol for content-centric networking," in *Proc. 1st IEEE Int'l Workshop on Emerging Design Choices in Name-Oriented Networking*, March 2012, pp. 304 –309.

[6] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: line speed publish/subscribe inter-networking," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, Aug 2009.

[7] S. Oueslati, J. Roberts, and N. Sbihi, "Flow-aware Traffic Control for a Content-Centric Network," in *Proc. IEEE INFOCOM Int'l Conference*, 2012.

[8] C. Yi, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, "Adaptive forwarding in named data networking," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 3, pp. 62–67, Jun. 2012.

[9] S. Braun, *Protocol Coexistence of CCN and TCP, Sharing the same network resources*. University of Basel, May 2012.

[10] G. Carofiglio, M. Gallo, and L. Muscariello, "Joint hop-by-hop and receiver-driven interest control protocol for content-centric networks," in *Proc. 2nd ACM Workshop on Information-Centric Networking*, ser. ICN '12, 2012, pp. 37–42.

[11] N. Rozhnova and S. Fdida, "An effective hop-by-hop interest shaping mechanism for ccn communications," in *Proc. 1st IEEE Int'l Workshop on Emerging Design Choices in Name-Oriented Networking*, March 2012, pp. 322 –327.

[12] S. Arianfar, P. Nikander, and J. Ott, "On content-centric router design and implications," in *Proc. ACM Workshop on Re-Architecting the Internet*, ser. ReARCH '10, New York, NY, USA, 2010.

[13] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the Performance of TCP Pacing," in *IEEE INFOCOM*, vol. 3, Mar. 2000.

[14] A. Charny, D. Clark, and R. Jain, "Congestion control with explicit rate indication," *CoRR*, vol. cs.NI/9809082, 1998.