An Empirical Study of Receiver-based AIMD Flow-Control Strategies for CCN

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—The Content Centric Network (CCNx) protocol introduces a new routing and forwarding paradigm for the waist of the future Internet architecture. Below CCN, a volatile set of transport and flow-control strategies is envisioned, which can match better different service requirements, than current TCP/IP technology does. Although a broad range of possibilities has not been explored yet, there are already several proponents of strategies that come close to TCP’s well-known flow control mechanism (timeout-driven, window-based, and AIMD-operated). In this paper, we carry out an empirical exploration of some proposed strategies in order to assert their feasibility and efficiency. Our contributions are twofold: First, we establish if receiver-based, timeout-driven, AIMD operated flow-control on Interest transmissions is sufficiently effective for CCN in a future Internet deployment, where it may co-exist with TCP. In this process we compare the performance of three different variants of this strategy, in presence of multi-homed content in the network (one of them proposed by the authors). Second, we provide indicators for the general efficiency of timeout-based flow-control at the CCN receiver, in presence of in-network caching, and exhibit some of the challenges faced by such strategies.

I. INTRODUCTION

The Content Centric Network (CCN) architecture [1], based on the CCNx protocol, is ambassador architecture among information centric networking projects [2]. It proposes a clean-slate architecture as a future alternative to or replacement for TCP/IP, which matches better the current Internet usage patterns. Simple, elegant, and minimalistic, it redefines the operation at the “waist of the network” to enable features such as loop free forwarding, location independence and flow balance. By casting away end-to-end communication and by enabling caching en-route, it makes content readily available from different locations simultaneously.

The CCN architecture specification leaves open the aspect of transport and flow-control mechanisms as the “strategy layer”. In this way it encourages the development of a multiplicity of transport protocols that can match various needs and the evolution of different services. Several initial attempts to propose functions for this layer have already appeared, as in [3], [4], [5], [6], [7], [8], [9] and among them, the most popular proposition is for receiver-based, timeout-driven AIMD (Additive Increase Multiplicative Decrease) flow-control, analogous to that of TCP (albeit adapted for the receiver).

We can see a few reasons that justify such a tendency. First of all, it is a strategy that has passed the test of time for its effectiveness, comes with well-established models for describing its dynamics, and also with a set of algorithms that are efficient and reasonably well understood to date. Second, if CCN reaches a level of maturity to attain broad adoption in the Internet, the currently foreseeable deployment options are the following three: (a) It may continue to run within TCP tunnels that connect CCN relays in an application overlay network (as the current CCNx prototype does). However, this seems to defeat the purpose of introducing new features such as flow balance in every hop, and offers little advantage over existing CDN (Content Delivery Networks) technology. (b) It may be deployed natively and independently of TCP/IP in “separation domains” sliced across the current forwarding fabric, by using technologies such as OpenFlow or MPLS. In this case, catering for the slices implies an administrative overhead for infrastructure network providers. (c) It may run in a mixed environment, on dual-stack configurations with TCP/IP (much like IPv4/IPv6 do). This last option is probably the most attractive, because it is simple, easy and ubiquitous, and does not require “separation policies” and intervention by ISPs. In this last case CCN’s protocol dynamics need to withstand TCP’s aggressiveness for occupying the shared network resources, and at the same time cash out the advocated advantages of in-network caching; but without instability to the current TCP-dominated Internet (so long as both protocols are in use). This provides a good enough reason for looking at timeout-driven AIMD flow-control in CCN, since in this case both CCN and TCP could exhibit compatible behaviours and similar rate-based dynamics (and which can be similarly affected by traffic shaping measures within the network).

In [10] we proposed such a receiver-based flow-control strategy for CCN that relies on RTT (Round Trip Time) measurements, Interest timeouts, and an AIMD controlled pipeline of Interests (requests for content). This strategy turned out to be compatible with TCP in the sense that a fair use of the capacity could be achieved along the common paths to content. Analogous comparable algorithms have been proposed in the literature, e.g., [3], [5], [7], [4], [8], [9], albeit not with the explicit intend to match/compare with TCP’s performance.

On one hand we consider this lack of reference comparison an important omission, for motivating a usage and paradigm shift from current TCP/IP technology. On the other hand, our initial observations in [10] made us curious about the efficiency of timeout-driven flow-control in presence of multi-homed content. This motivated the follow up work in this paper in an attempt to make this exploration more conclusive: (a) We compare the effectiveness and performance of three
receiver-based timeout-driven AIMD flow-control approaches in presence and in relation to TCP, in a network with multi-homed (cached) content. (b) We propose a set of reference measurements for assessing the performance/efficiency of such timeout-relying approaches. Based on these measurements we show the fragileness of timeout-driven flow control in face of multi-homed content in CCN (which needs to be taken into account before further research is invested in this direction).

The remaining of this paper is structured as follows: we start in the next section by introducing the AIMD strategy and the variant algorithms that we employ in our experiments. In Section III we explain the experimentation environment that we used. In Section IV we incrementally unfold our findings through a set of experiments. In Section V, we discuss our findings and their implications in the general context of other related research in this topic, and finally in Section VI we summarise and conclude this paper.

II. AIMD WINDOW-BASED, TIMEOUT-CONTROLLED INTEREST TRANSMISSION STRATEGIES

Why consider AIMD window-based, and timeout-driven transmission strategy? The foremost reason is that existing rate-control measures within the network are currently tailored to influence TCP’s flow-rates, by biasing transmission timeouts. It is pragmatic that these mechanisms can interact also with CCN receivers with analogous effects (i.e. AIMD scaling of transmissions).

TCP uses a flow and congestion control approach, whereby it repeatedly causes congestion and losses in order to sense “the boundaries”. The AIMD controlled window of in-order pending acknowledgement transmissions emulates an adaptive, in-network, virtual buffer for synchronizing the sender with the receiver. By doing so, it maintains a continuous flow of data at a throughput rate that always remains within the sensed boundaries. Because of end-to-end semantics, it relies on transmission acknowledgements (or lack thereof) and timeouts recorded at the sender to adjust the transmission window. Skews or noise in the time-based estimators can lead to abuse or underutilisation of a flow’s capacity share, and perturb the synchronisation between sender and receiver. In this case, the Fast-Retransmit/Fast-Recovery mechanism tries to recover the lost synchronisation and when this fails, falling back to the Slow-Start phase serves as a last-resort “reset” button.

By contrast in CCN, because content originates from in-network caching points, there are no end-to-end flow-control semantics [1]. Nevertheless some sort of flow-control can be ventured between the receiver and the engaged points of cached content. A role similar to TCP’s transmission window may be held by a pipeline of Interest requests – dispatched and pending-for-content– which a CCN receiver may use to regulate its capacity occupancy/claim (against other CCN content transfers and possibly TCP flows). However, by contrast to a transmission window, the Interest pipeline is not emulating one sole “channel buffer”, which needs to preserve the order of transmissions and receptions, and does not pertain so strictly to the two-end synchronisation problems. As a result there is no (profound) need for mechanisms that try to recover synchronisation (such as Fast-Retransmit/Recovery in TCP) or to “reset” synchronisation (by analogy to Slow-Start). The sole challenge that remains is how to regulate the scaling of the pipeline in face of the dynamically varying bandwidth-delay product (pipe-capacity) due to variable path lengths to content stores (as a result of en-path caching and content multi-homing) in CCN. This last aspect is of key importance for optimal utility as well as stability, because the assumptions underlying its use in TCP’s algorithms is that (a) the path-length may (but is unlikely to) vary often/fast, and (b) delay measurements are samples, with very high probability, from a single distribution (which reflects the stochastic traffic conditions in reaching one specific point in the network). These assumptions rarely hold in CCN: content originates from several Content Stores (CS).

Based on these insights, in the following sections we describe a basic operation of a receiver-based AIMD flow-control strategy (also in pseudocode in Algorithm 1), with three different timeout-estimators for controlling the Interest pipeline. Two of them bear similarities to other proposals in the literature, while the third one was proposed in [10] and is further consolidated here.

A. Basic AIMD strategy

1) Pipeline control: To implement the interest pipeline we use two data structures: One (sentList) keeps a record of dispatched Interests pending content reception (its length is the active size of the pipeline). A second one tracks the timed out Interests (missingChunkList). When, in the sentList, a timer for a specific Interest awaiting for its 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 reduces to match this nominal size. Then the pipeline size can start increasing again stepwise (Astep), at every successful reception, allowing new Interests to be dispatched. At this point, previously expired Interests from the missingChunkList are prioritised for re-transmission over new Interests, as they are likely to be satisfied faster (because of previously en-path cached content) and therefore the pipeline size will ramp-up faster. If in the mean time content-chunks of expired Interests still arrive delayed, we simply remove them from the missingChunkList, so as not to congest the network with unnecessary re-transmissions.

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) – see function UPDATE_TIMEOUTESTIMATOR. Note that unlike a typical TCP configuration, the gain parameter for the RTT variance smoothing is kept small (.125 instead of the typical
in order to attain a more symmetric increase and decrease rate in the time-out adjustment. In TCP the timer needs to account for a fast "pile-up" of delay due to congestion that will be slowly alleviated, while here we want to account for steep and abrupt delay changes that are anticipated in both directions due to inter-cache switchovers.

3) Counter-acting 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 restart the additive-increase, or otherwise a second reduction may occur immediately after Fast-Recovery. In our algorithm however, since individual chunk requests-receptions are seen as independent of each other, subsequent timeouts can lead to an avalanche of back-offs of the Interest pipeline (until the RTT estimate is corrected). To give time for the RTT estimation to be corrected (refreshed) before follow-up timeouts admit further back-off steps, therefore we de-activate the pipeline reduction process for the duration of one estimated RTT following a timeout.

B. Weighted Average Timeout Estimator

In case of multi-homed content there are a number of CSs delivering content, each of which is reachable with a different bandwidth-delay product (BDP). The "quality" of the timeout estimator relates to how accurately it captures and averages the variability Δp between these BDPs. In [3] initially, and through further improvement in [8], the authors seem to affirm that it is possible to provide a reliable enough estimator based on a weighted average of a history of the last N (empirically 20) RTT measurements. To test if this approach is robust to irregular sudden steps in the delay, and when multiple frequency components contribute to the RTT distribution from "inter-cache switching", we developed a variant of the basic algorithm that abides to this logic. A weighted average timeout estimator is computed as follows

1) Every CS marks the Content packets it serves from its local cache with an identifier (unique for the tuple [CS, content name]). If a CS receives a Content packet that satisfies an Interest in the PIT (Pending Interest Table), it leaves the identifier of the originating CS unchanged. Additionally, whenever it caches content chunks, the identifier is erased.

2) The receiver does accounting for all such identifiers it has seen, and maintains chunk counters, RTT measurements and time-out estimates (function UPDATE_TIMEOUTESTIMATOR) for each. Whenever a new Interest is sent, the timeout estimator is calculated as a weighted average over all seen CS identifiers, weighted either (a) by number of chunks received from each of them, or (b) by freshness of measurement. In this case we can choose to assign higher importance (weights) either based on the most recent or the most frequently seen identifiers.

The authors in [8] store the entire path traversed by the content packets (as opposed to the origin CS ID only), so as to additionally distinguish between multiple paths to the same CS. This further improves the granularity of the different delay components considered in the computation of the average, but since in the tests that follow no such multi-paths are considered this ramification does not affect the comparison.

C. A "predictive" Timeout Estimator

In CCN every Interest request is independent of any other, and thus there are no ordering constrains in the operation of the pipeline. By identifying clusters of chunks (we call them batches) that are likely to originate from the CSs with similar path-delay, then the respective Interests can be sent back-to-back and independent of all other, using a timeout estimation that is based on RTT measurements only from the respective CSs. Such per-batch estimators are expected to have more consistent variance (uniform noise) since it is a measurement to a single point and therefore can lead to more reliable timeout prediction for that batch of content chunk retrievals. Furthermore, grouping together batches of Interests served by CSs that have similar BDP, and sorting/ordering the Interest

---

1 An explanation of gain parameter settings and their effects can be found in appendix D of [11]
requests not based on chunk number but based on delay-to-CS, can give low-frequency/intensity variations $\Delta p$ of the pipe size and allow the pipeline more time to adjust. Building on this idea the third variant of our basic algorithm as follows

1) Content packets are marked and handled with a unique per-CS identifier as before. In addition to that, when a CS sends the first of a continuous range of chunks (for some part of content that it has cached), it also stores in the packet a value stating how many subsequent chunks are locally available. This value needs to be sent only once with the first chunk in a range, or every time the range of available chunks changes.

2) The receiver does accounting per CS identifier it has seen as before (Section II-B), and additionally stores for each CS the content chunk ranges it has cached. Based on this information it can maintain distinct timeout estimators per CS (function UPDATE_TIMEOUT_ESTIMATOR) and also probabilistically predict for each Interest which of those timeout estimator to use (knowing in which CS’s range it belongs). Additionally, Interests that are likely to be consumed by similar distance CSs are grouped and transmitted together. In case of an Interest in an unknown (“not advertised”) range, the weighted average estimator from Section II-B is used.

The performance of this algorithm is expected to gradually improve in time, as the receiver “learns” more chunk ranges or receives more chunks from the same CS.

III. EXPERIMENTAL SETUP

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.

Instead of validating the results on complex topologies and providing “black-box” type of observations, we have chosen to present experiments and demonstrate our findings with a rather simple setup. This case the introspection of relationships between phenomena and their underlying causes, while sufficiently demonstrating the “unfortunate conditions” that could happen (in relation to features of each algorithm). The experimentation topology we use is shown below, where the assigned BDPs of the pipe size and allow the pipeline more time to adjust. Building on this idea the third variant of our basic algorithm as follows

**Example Diagram**

---

IV. TESTS AND ANALYSIS OF RESULTS

To incrementally unfold our findings we structure our experiments as follows: We first provide a unit test of the three algorithmic variants in a setup where en-path caching is disabled for CCN and in presence of a competing TCP flow. This serves to assess the relative correctness of the AIMD strategy, performance and compatibility with TCP, in terms of their nominal behaviour. In the next step, we compare the performance/behavior of the three algorithms in presence of multi-homed content, in terms of their effectiveness to withstand TCP's aggressiveness. Again, TCP also serves as a reference for performance improvement, since multi-homing should give a variable advantage over end-to-end data delivery.

In the last set of tests, we pick the best performing algorithm and subject it to more comprehensive tests, with different distributions of content across the path to the source. This allows us to explore the implications of caching for the performance of a timeout-controlled strategy.

The main evaluation metrics are the pipeline stability and sensitivity, the flow/transfer completion time, and the throughput (sliding window averages). In the last set of experiments we also look at the Interest retransmissions and the protocol efficiency as defined in RFC 6349.

The phenomena we report are consistently reproducible at varying intensities, across configurations of varying complexity (number and combination of flows). For ease of illustration and discussion, we have chosen to present them here through a series of simple examples.

A. No caching - content retrieved from a single point

First, we seek to assert the nominal behaviour of the three variants of the AIMD strategy, by relevance to TCP. This means that excluding the advantages of caching, each of the three algorithms should react consistently to the available or reduced capacity. For this, we arrange that clients $R_i$ receive CCN content (under the AIMD request strategies) or TCP data,
initially from the same and then from different remote hosts. In Fig. 1 we report results that confirm this hypothesis.

Specifically, all three variants exhibit almost identical behaviour (we only illustrate the results of the base algorithm in Fig. 1). This agrees with our expectation because, in absence of multi-homed content, the timeout estimation reduces to the same calculation for all three versions. More important, the system of CCN-TCP flows is analogous to that of having 2 competing TCP flows: when the paths to the content have the same pipe size the throughput tends to split evenly (Fig. 1.(a,b)), while when the paths have different pipe sizes (Fig. 1.(c,d)) the throughput tends to divide proportionally.

The window variations shown on plots 1.(a,c) additionally inform us that the two flows mix nicely (in both cases), and the windows scale almost in synchrony. Although we have not explored in depth the flow mixing aspect, we did not observe the “flows taking turns” behavior, which is often emerging with competing TCP flows [12], [13] (and often has adverse effects for the interacting flows).

![Fig. 1: CCN versus TCP. In (a,b) both receive data from host 1 \((MDcoef=0.5)\). In (c,d) CCN receives content from host 2, TCP from host 1 \((MDcoef=0.4)\).](image)

B. Multi-homed content

A more interesting round of experiments served a performance comparison of the three variants, when multi-homing is enabled. In addition, to put these tests in a more realistic context and quantify the improvement of enabled caching over end-to-end data delivery, we test each variant against a reference TCP flow that retrieves content over a longer path. We therefore pre-distributed fragments of the content (in batches of 50 chunks) along the end-to-end path to the source following the distribution pattern shown in Fig. 2. The same pattern is repeated 10 times resulting in 80 cache switch-overs and “abrupt” changes in the RTT measurement. The challenge for each version of the CCN strategy is to cope effectively with the intense RTT variations due to the different locations of the content. Fig. 3 reports the performance for each of the three algorithms against the reference TCP flow that transfers content from node 1 (its RTO estimator being thus influenced by the variations in the CCN traffic pattern).

![Fig. 2: Distribution of content-chunks per CS, shown as ranges of 50 chunks (called batch size). This distribution assigns 30% of the content in host 2, 10% in host 3 and 60% in host 1.](image)

In Fig. 3.(a,b), we observe that the base algorithm cannot benefit from the fact that 40% of the content is nearer to the receiver than for TCP (which attains much better flow completion time). A similar test with this algorithm in [10] but with 500-chunk batches showed opposite results, which affirms that the performance of this algorithm is dependent on the size of the content fragments (pre-cached batch size).

The same phenomenon, but less intense, is observed in Fig. 3.(c,d) for the weighted average version of the algorithm. The emphasis of the weights on the mostly used CSs, which favors the respective RTT measurements as more dominant, smooths the variance and filters the high frequency components. It is worth mentioning that assigning higher weights to the most recently accessed CSs (i.e. most recent RTT measurements [3]), instead of the mostly accessed, has slightly worse performance under this content distribution pattern, but performs better under other content distribution patterns, where each CS is accessed for prolonged periods. The performance of this algorithm thus depends on the content distribution pattern, making in this way its tuning challenging.

In both cases, we have requested the chunks in sequence order. When “switching over” from one CS to another, often, the variance of the measured RTTs gets dominated by large steps and high frequency components, which are not always easy to filter out and introduce substantial noise in the timeout estimation. When switching over to a CS with longer delay or more congested path, the previous RTT measurements can lead with very high probability to timeouts and thereby back-offs. If the pipeline size falls below a limit point, the network capacity gets under-utilised (and taken over by the TCP flow). In the opposite case, when switching over to a CS closer to the receiver, it is likely that the current pipeline is too large for the bandwidth delay product of the new path. Congestion is likely, causing losses and thereby again back-offs. This situation advantages flows with more stable paths (and the more aggressive TCP ones), which gracefully take over the availed capacity. The main insight in these results is that by mixing measurements of RTTs towards different destinations, the computed mean does not regress with a BDP that maximises goodput to any of the destinations (that is because the underlying distribution of mixed RTTs is not guaranteed to be Poisson-like as the those of individual
On the other hand, the predictive version of the algorithm turns out to be the most robust to these varying phenomena (see Fig. 3.(e,f)), and outperforms TCP “as it should” in terms of flow completion times (when caching is in place). In fact, as captured in both the pipeline scaling and the throughput sharing plots, the algorithm exhibits a (statistical) determinism: 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 point to note in this case, is that the content retrieval must involve a sufficiently large number of chunks in relation to the number of CSs involved, in order to “learn” the different RTT/timeout profiles of each CS and correctly adapt the pipeline to each in turn.

Leaving aside the competition against TCP for the moment, we examine in more depth two key parameters that reflect this dynamism and affect the AIMD pipeline operation. One is the frequency of CS switch-overs, that affects the adaptation time and stability of the Interest pipeline (we remind that the pipeline adapts in steps at the rate of RTT measurements). The other parameter is the variations Δp of the BDP between two CSs during “switch-overs”. As explained before the probability that a timeout happens depends on the difference of the estimated BDP (based on past RTT measurements) versus the actual one along the path to the next CS. We refer to the combined effects of these two parameters as entropy of the content in the network. Timeouts tend to happen more often as the entropy increases due to invalidated estimates, which in turn may affect the throughput. However, the underpinning belief in all approaches discussed so far is that timeout estimation at the receiver can be sufficiently robust to the entropy of content, so that a receiver strategy remains invariably effective.

We have added node C in the topology and we distributed batches of content chunks (continuous ranges) based on the pattern in Fig. 4. Using the same pattern (repeatedly) and varying the chunk batch sizes in every CS, we can change the frequency of CS switch-overs during a content retrieval. Employing the predictive version of the AIMD strategy (as it is the most robust), we set the $R_t$ nodes to retrieve content from the network, based on the different batch sizes. In Fig. 5, we report how the throughput is affected by the CS switch-over frequency, for three of the tests. On the positive end, one can see that still this version of the algorithm tends to be “reliable” as the pipeline adapts fast enough to match the maximum available capacity. On the negative side however, as the switch-over frequency increases, so does the jitter. Adaptation becomes noisier, and throughput turns out to be sub-optimal because the algorithm does not have time to group requests from the same CS together!

![Graphs showing CCN versus TCP window size and throughput](image)

*Fig. 4: Distribution of content-chunks per CS, shown as ranges of 50 chunks. This distribution assigns 50% of the content in host 2, and the other 50% in host C.*

Next, in order to account for the effects of Δp (second parameter), we additionally vary the bandwidth-delay characteristics of link $l_{1C}$ such that we can emulate a CS at different distances from the rest of the topology. The variance of the BDP between CS2 and CS_C is $Δp = \text{BDP}_{AC} - \text{BDP}_{A2}$. Then, we repeat the tests while varying both the CS switch-over frequency and $Δp$. In Fig. 6, we report the results for three performance metrics: transfer time ratio (flow completion), Interest retransmissions, and protocol efficiency (as defined in RFC 6349), as a function of $Δp$ for different batch sizes.

One can clearly see that the performance of the timeout-based AIMD strategy drops very fast as the content entropy increases ($Δp$ increase and high CS switch-over frequency). Caching then becomes counter-beneficial! In fact, more packet drops occur at the queues before link $l_{32}$ than at link $l_{1C}$.
delay product’s worth pipeline of CS decreases (the AIMD flow-control tries to fit the bandwidth-C. In (b,c) the batch size is 500 chunks, and in (d,e) 50 chunks distributed based on Fig. 4. In (a,b) all 5000 chunks come from node path bottleneck. The more severe penalty is thus paid when Fig. 5: CCN with predictive algorithm receives multi-homed content in the network. A single (weighted) average timeout estimation varies depending on the content distribution pattern algorithm of this paper. As we showed, although reasonably estimator similar to the one we employed in the second variant in [8] on the other hand, uses a weighted average timeout to TCP). The strategy proposed in [3] and later improved nevertheless the authors in both cases implement a timeout estimator and performs considerably better than the other two, by maintaining multiple timeout estimates (requiring more information from the network) and re-ordering content chunk retrievals to smooth pipe scaling during CS-switchovers. In [9], the authors are motivated by a similar insight and follow a counterpart packet marking process so as to obtain network-side information. This enables them to similarly calculate multiple timeout estimators and in addition to operate parallel and synchronised transmission windows (resp. Interest pipelines) to each serving CS\(^4\). This approach effectively goes so far in emulating (multiplexed) TCP sessions as to adopt features like Slow-Start and request-delivery order dependency. The design and implementation complexity proposed is significantly higher (both for the receiver and the network side), than the “predictive” algorithm discussed here, but we expect its performance to be comparable. An exploration motivating further comparison, involves the role/impact of a Slow-Start phase: in our case we have assumed that in CCN the effects of a Slow-Start phase are at least unnecessary (no need for synchronisation reset) and at most potentially harmful by unpredictably-prematurely sizing the Interest pipeline (depending on which CS among several is hit first by a timeout).

In relevance to network-side flow/rate control measures, the authors in [15] and [5] introduce in-network congestion avoidance based on adaptive queue management (AQM) for drop-tail queues. This seems to affirm a tendency to borrow existing knowledge and models from the current TCP-dominated Internet for engineering transport logic and mechanisms in CCN. However, these “direct imports” should be faced with scepticism: congestion measures that may increase the BDP variability across CSs (drop-tail queues that add delays in this case) will only make things worse for the efficiency of timeout-based strategies. By contrast, in [6], the authors present a different approach for in-network flow and congestion control in CCN, where the flow-control logic is entirely inside the network, and a “logic-free” receiver only executes actions in response to explicit back-pressure from the network. Interestingly in this case, the authors foresee that frequent sudden delay variations related to multi-homing will be problematic, and try to avoid packet drops and timeouts. They proactively shape the Interest pipeline (based on observations of Content packets flow, and taking advantage of the per-hop flow-balance) so as to smooth the jitter inside the network and thus improve the quality of RTT measurements at the receiver. This approach would be more “receiver/timeout-strategy friendly” than the respective in [15] and [5], if both receiver-based and network-assisting measures were to be deployed.

Finally in regard to receiver-based timeout-driven flow con-

\(^4\)This work appeared after the initial submission of this paper and so we did not have the opportunity to consider it in the comparison testing we presented.
Transfer time ratio, interest retransmissions and efficiency are affected differently depending on the batch size (5000 = no multi-homing).

Re-transmission rate is independently parameterised at each responsive to the receiver’s intentions. Interest propagation and dynamic adaptation of the Interest rate has practical limitations: since a CCN communication is proxied at several intersections with other delivery paths, will often be forced to repeatedly “overwrite” parts of content of one path with that from others. Further study in this direction is deemed appropriate, to establish the effects of traffic dynamics and content replacement practices but needs careful consideration as existing models from Content Delivery Networks (CDN) are not necessarily applicable in the context of CCN: caches are not sitting at the network periphery, and therefore are more subject to router dynamics rather than server dynamics! Second, receiver-based flow-control approaches that rely on dynamic adaptation of the Interest rate has practical limitations: since a CCN communication is proxied at several places along a path, one cannot assume that the caches are responsive to the receiver’s intentions. Interest propagation and re-transmission rate is independently parameterised at each cache and does not abide to the behavior of Interest senders.

VI. CONCLUSION

In this paper we focus on receiver-based timeout-driven AIMD flow-control in CCN, and particularly in relevance to TCP. We compared three algorithms with different approaches to estimating timeouts for adapting the Interest pipeline, explored their performance and assessed their efficiency in sharing the network with TCP flows. We showed that when content is retrieved from a single point in the network and thus end-path caching is disabled, an AIMD strategy is as effective as TCP between two points. The same results however are not as promising when content multi-homing is in place. One of the algorithms only turned out to be robust against TCP and cash out the benefits of multi-homed content. Regarding the efficiency of timeout-based flow-control in CCN in general, contrary to current belief, we showed that it is not as compatible to the CCN philosophy, and its performance varies depending on the content entropy in the network.

REFERENCES