Adaptation and Evaluation of Widely Used TCP Flavours in CCN

Asanga Udugama, Jinglu Cai, and Carmelita Göerg

University of Bremen, Germany
{adu,jlc,cg}@comnets.uni-bremen.de

Abstract. Content Centric Networking (CCN) is a paradigm shift from the way how networks of today work. The focus of networking in CCN is on the content and not on the hosts that are involved in a communication. One of the key cornerstones of today's communication model is the use of flow and congestion control to pipeline data and take appropriate action when congestion is perceived to exist in a network. TCP of the Internet protocol suite has shown us how application performance is enhanced in different communication situations. An interesting area of research is how TCP-like flow and congestion control can be adapted for CCN. The work presented here adapts the most widely used TCP flavours of NewReno, Compound and Cubic to operate in CCN. Due to the architectural differences that CCN has over IP based networks, this work identifies a number of additional algorithms to cater to the issues associated with these differences. Finally, the performance of these adapted TCP flavours and the algorithms are evaluated in an OPNET based simulator.

Keywords: Future Internet, Content Centric Networking, Flow and Congestion Control, Simulations.

1 Introduction

Network use has evolved to be dominated by content distribution and retrieval, while networking technology still speaks only of connections between hosts. Accessing content and services requires mapping from the "what" that users care about to the network's "where". Content Centric Networking (CCN) is a new paradigm in networking which treats content as a primitive - decoupling location from identity, security and access, and retrieving content by name. Using new approaches to routing named content, derived heavily from TCP/IP, CCN achieves simultaneous scalability, security and performance [1,2].

CCN retrieves content using interests that propagate all throughout the network. This is unlike networks of today, which are mainly based on TCP/IP. These networks have made TCP as the main transport protocol to communicate data between producers and consumers of content. One of the key advantages of TCP is its algorithms to handle reliable delivery of data, end-to-end. CCN on the other hand has a hop-by-hop content delivery mechanism and a receiver oriented communication architecture where delivery of content cannot be guaranteed by the sender. Flow and Congestion Control (FC-CC) algorithms in TCP have worked well in current networks. There are a number of different TCP flavours in use today. The most widely used flavours are *NewReno*, *Compound* and *Cubic* [3,4,5]. These flavours which are based on the original TCP [6], attempt at addressing the issues associated with different network conditions. The work presented in [1] explains that the architecture of CCN has FC-CC built into it, mainly through the concept of flow balance, i.e., one *Interest* packet retrieves at most only one *Data* packet. It further states that the segment numbers in *Interest* packets act as the sequence number of TCP, CCN *Interest* packets acts as the TCP ACK [6] that acknowledges receipt of data and that TCP SACK is intrinsic due to *Interest* packets being re-sent for unreceived data.

There is research being done currently to introduce TCP-like FC-CC for CCN [7,8,9]. All of these efforts focus on developing new algorithms to perform FC-CC in CCN. One area that has lacked concentration is how the algorithms that currently exist in TCP would operate in CCN when they are adapted to operate in CCN. The work presented here adapts the algorithms of the 3 most widely used TCP flavours (*NewReno*, *Compound* and *Cubic*) to operate in CCN. Since there are a number of architectural differences that CCN has, compared to TCP/IP based networks, a couple of new algorithms have been identified to operate together with these adaptations. The rest of this paper describes our work as follows. The Section 2 details similar work done by others in the area of FC-CC together with a comparison of the work done by us. The Section 3 provides a description of the adaptation and the new algorithms we have identified for the selected TCP flavours to operate in CCN. The Section 4 details the OPNET based simulator we have built to evaluate our work, the performance results and the analysis of the results. Finally, Section 5 provides a summary of the work, conclusions and future work to be considered.

2 Related Work

Flow and Congestion Control (FC-CC) has been a topic of interest in CCN since the seminal work presented in [1]. The way in which FC-CC can be operated in CCN is fundamentally different from the way it is operated in current TCP/IP based networks. Therefore, a number of aspects must be addressed in order to make FC-CC possible in CCN. These aspects fall into 4 categories.

End-System Congestion Control Algorithm - There are multiple ways to manage the FC-CC congestion window (referred to as window hereafter) used to decide the amount of *Interest* packets that have to be sent at any given time. This window may be increased in size for successful receipts of *Data* packets or decreased when perceived packet losses or congestion is detected in the network.

Packet Loss Detection - FC-CC requires the knowledge of whether a requested *Interest* packet has been replied to with the corresponding *Data* packet. If a packet is lost or a *Data* packet arrives out-of-order, this is an indication of losses or congestion in the network.

Fairness Realisation - The architecture of CCN does not posses the end-to-end notion as in current TCP/IP based networks. This means that packets travel hop-by-hop and each hop determines how the CCN packets are forwarded. Since each hop has limited resources, there is a necessity to adopt specific fairness controls at each hop to give a faire share to each of the content flows that travel through that hop.

Flow Identification - As indicated above, fairness requires the identification of content flows to provide fair sharing of resources. In CCN, requests for content may originate from many different sources and a hop in the middle is unaware of the originator of the requests. Therefore, a flow identification method has to be adopted to assign resources fairly for the competing flows.

The authors of [7] investigates the performance of FC-CC using algorithms based on Additive Increase/Muliplicative Decrease (AIMD), constant and Constant Bit Rate (CBR) based window management. They utilise the Retransmission Timeout (RTO) and 3 out-of-order *Data* packet receipts to detect packet losses. Fairness is realised by maintaining per-flow queues in the internal buffer. When a buffer overflows, the packets in the longest queue are dropped based on Deficit Round Robin (DRR).

The authors of [8] based their window management using a CBR window. Fairness is realised by introducing a hop-by-hop *Interest* shaping algorithm that anticipates the drop of *Data* through buffer overflows. The *Interest* shaping rate is calculated using the delay from the *Interest* to the corresponding *Data*, the buffer size, the available bandwidth to send the *Interest* and *Data* packets, the number of queued *Data* packets for each flow and the number of conversations flowing through the same CCN node. In this work, the *Interest* shaping ensures that the buffer is equally distributed to each conversation.

The authors of [9] utilise an AIMD and a CBR based window management algorithm. The packet losses are detected using RTO. Fairness control is applied to the *Interest* flows. Each bottlenecked *Interest* flow maintains a queue for fair distribution of the buffer. The bandwidth distributed to the bottlenecked *Interest* flows is considered as the fair rate.

Flow identification in all these works [7,8,9] are done using the content name.

In contrast to these works, the work done by us focus on utilising the window management algorithms used in the most widely used TCP/IP flavours, viz., *NewReno*, *Cubic* and *Compound*. A further aspect considered is the handling of out-of-order packets. In CCN, packets may be out-of-order, not only due to losses or delays but also due to arrivals from different caches. Therefore, it is important to distinguish this difference before making window adjustments to avoid inappropriate adjustments that may result in performance degradations. Therefore, we have considered this aspect as an important factor in FC-CC and introduced an algorithm to handle this situation.

In all the above mentioned works [7,8,9], the focus has been on fairness controls that consider the flow of *Interest* packets. Our work too, considers fairness in the same manner, but goes beyond by also considering the *Data* packet flow, in addition to the *Interest* packets. In CCN, the payload of each data segment is carried in the *Data* packets. Since *Interest* packets do not carry this information, assigning a fair share of bandwidth must also consider the *Data* packets.

In CCN, the same content can be requested by multiple CCN nodes and due to the nature of CCN, intermediate nodes are unable to distinguish between different flows. Since the identification of a flow is quite important to handle fair sharing, we identify a flow by not only the Content Name and segment number (as in the other works [7,8,9]) but also by introducing a random nonce at the originator of *Interests* for every independent flow.

3 Adaptations

The architecture of CCN essentially uses a multicast or broadcast based mechanism to propagate *Interest* packets for content and the content (i.e., *Data* packets) flow over the paths that were created due to the propagation of the *Interest* packets. It is somewhat of a new way of communicating for mainstream communications of today, considering that unicast is the norm of current networks. The draft CCN specifications [1,10] discuss how CCN is architecturally close to TCP (flow balance, SACK, etc.). But when considered from a TCP point of view, we see that there are a number of issues that make FC-CC for CCN different from TCP FC-CC. Therefore, adaptation of the *NewReno*, *Compound* and *Cubic* flavours of TCP require considering the following aspects.

Control/Communication Orientation - In TCP, the sender of data in a communication session is in control of the session rather than the receiver. This means that the sender continually sends data to the receiver based on the senders view on how the network performs. On the other hand, no data will traverse the network unless the data has been requested for in CCN and therefore, in CCN, the receiver is in control. This means that the receiver is in charge of the data flow by controlling how *Interest* packets are sent.

Data Acknowledgment - In TCP, data are acknowledged by a message that travels in the opposite direction, to the sender. This acknowledgement (ACK) system is one of the fundamental pillars of TCP, used by the sender to make a number of decisions on how the rest of the data must be transmitted. But in CCN, there is no concept of content ACKs. Therefore, the receipt of *Data*itself is considered as the acknowledgement in CCN.

Congestion Window Management - One of the key aspects of TCP is the use of congestion windows to control the flow of data between the sender and the receiver. In TCP, this is maintained at the sender due to the sender orientation of TCP. On the other hand, since CCN is receiver oriented and the *Interest* sending is considered as the means by which the flow of data is controlled, the congestion windows must be maintained at the receiver.

Congestion (Packet Loss) Detection - TCP uses the RTO and the receipt of 3 duplicate ACKs as the basis for considering congestion in the network [6]. In the case of CCN, the RTO can be built in the same way as in TCP by considering the RTT associated with the *Interest*-*Data* cycle. But, CCN does not have the concept of 3 duplicate ACKs and further, unlike TCP, CCN has the additional problem of determining whether any out-of-order *Data* receipts are due to congestion (or packet loss) in the network or due to *Data* arriving from multiple sources.

Therefore, the work presented here identifies an algorithm called ***Pre-Recovery*** that addresses the issue associated with multi-source arrivals.

Recovery Algorithm - TCP uses the *Fast Recovery* and *Fast Retransmit* algorithms to resend data which are considered to be lost. Since CCN does not have the concept of duplicate ACKs, the work presented here identifies a CCN based ***Fast Recovery*** algorithm and ***Selective Fast Retransmit*** algorithm to handle the resending of *Interest* packets.

Resource Sharing Fairness - Due to the reasons explained in Section 2 on *Fairness Realisation*, the work presented here identifies a ***Hop-by-hop Fairness Control*** algorithm to provide fairness in sharing bandwidth.

The following sections describe the TCP flavour adaptations and the new algorithms identified for these adaptations to operate successfully.

3.1 Flow and Congestion Control Adaptation

The TCP flavours considered in the work presented here utilise the same algorithm during the *slow start* phase after establishing a connection or when an RTO occurs, i.e., it starts with $cwnd = 1$ and performs a $cwnd = cwnd + 1$ for every non-duplicate ACK received until slow start threshold ($ssthresh$) is reached. In the case of CCN, the same operation is performed but considering the *Data* receipt as the ACK of a successful delivery of data for the corresponding *Interest* sent previously.

The differences of these flavours occur in the *congestion avoidance* phase of operation. They are as follows,

NewReno [3] - The increase of $cwnd$ is performed using $cwnd = cwnd + 1/cwnd$ on each non-duplicate ACK arrival.

Compound [4] - Compound uses a congestion window that is computed based on combining the loss based window ($cwnd$) similar to New Reno and the delay based window ($dwnd$) that is updated at the end of every RTT. If no early congestion is detected, $dwnd$ is increased and if early congestion is detected, the $dwnd$ is decreased.

Cubic [5] - Cubic uses a congestion window based on real time unlike the RTT used in *NewReno* and updates the current window based on $W(t) = C \cdot (t - K)^3 \cdot + W_{max}$ where t is the elapsed time since the last window reduction time, $K = \sqrt[3]{\frac{W_{max}\beta}{C\cdot}}$, W_{max} is the window size before the last reduction and $\beta = 0.2$.

We adopt the same differences in our adaptations (including the variables) considering *Data* receipt as the ACK, RTT computation from the *Interest-Data* cycle and 3 out-of-order *Data* packets as the rigger for the recovery process (described in Section 3.2). We term these adapted flavours as *CCN-NewReno*, *CCN-Compound* and *CCN-Cubic*.

3.2 *Pre-Recovery* Algorithm

TCP/IP always assumes a point-to-point communication basis where data arrives from one source. In CCN, on the other hand, requests for content may be served from multiple sources due to caches. Therefore, out-of-order *Data* receipts may be due to a (a) loss or a delay of *Interest* or *Data*, or (b) due to arrivals from multiple sources that could also be multi-path transmissions.

An algorithm is identified in this work (called the *Pre-Recovery* algorithm) to detect whether out-of-order *Data* receipts are due to (a) or (b). A summery of the operation of this algorithm is presented in the Table 1.

Table 1. Operation Summary of the *Pre-Recovery* Algorithm

Pre-Recovery	**Operation**
Entry	3 out-of-order *Data* receipts (3 is the *FRT* value used in TCP)
During	Congestion Window updated based on FC-CC flavour used
Exit	A loss detection and *Fast Recovery* entry or, total number of out-of-order *Data* $< FRT$, when $FRT = 3$ or, the occurrence of RTO

The fundamental idea behind this algorithm is to prevent the unnecessary changes that is made to the congestion window when a false packet loss is detected (disregarding the multi-source arrival issue). Figure 1 shows how multi-source arrivals or *Data* loss detections are made, respectively.

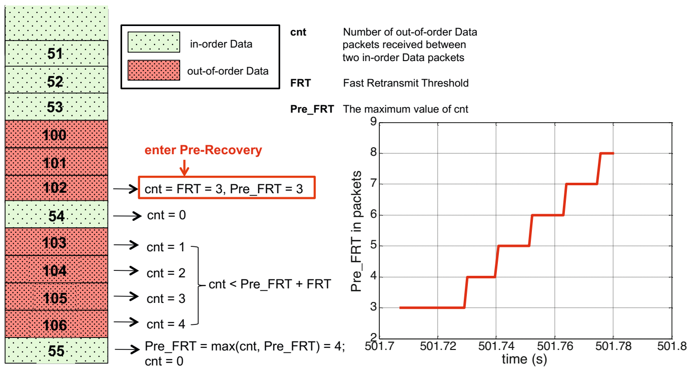
This algorithm utilises 3 variables in its process (Figure 1(a)). The FRT (similar as in TCP), maintains the amount of out-of-order *Data* packets that are considered for the algorithm to commence operation. Pre_FRT initially starts with the same FRT and is continuously checked to see if the contiguous receipts of out-of-order *Data* packets exceed the $FRT + Pre\_FRT$. As soon as an in-order *Data* packet arrives, the Pre_FRT is set to the $max(cnt, Pre\_FRT)$. A receipt of an in-order *Data* packet at this instance indicates a multi-source arrival and hence this algorithm will prevent the FC-CC moving into *Fast Recovery and Selective Fast Retransmit.* Thereby, *Pre-Recovery* prevents the ping-pong effect that the congestion window may display due to multi-source arrivals.

If the number of contiguous out-of-order *Data* receipts continue to grow beyond $FRT + Pre\_FRT$, FC-CC is moved into *Fast Recovery and Selective Fast Retransmit* (Figure 1(b)).

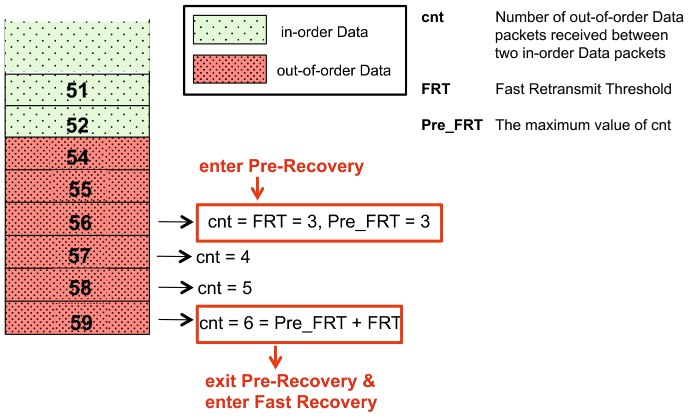
3.3 *Selective Fast Retransmit* Algorithm

Once the *Pre-Recovery* (Section 3.2) algorithm determines that a packet loss has occurred ($cnt = FRT + Pre\_FRT$), FC-CC moves into *Fast Recovery.* The *Fast Recovery* algorithm operates in a similar manner to TCP but with adaptations to operate in a CCN context.

On entry into *Fast Recovery*, the *cwnd* is reduced to $ssthresh + n$. In TCP, n is the 3 duplicate ACKs received. But in CCN, since the *Pre-Recovery* process would have received a number of out-of-order *Data* packets, n refers to these out-of-order packets. During the *Fast Recovery*, a receipt of an out-of-order *Data* packet results in the *cwnd* being increased in the same manner as TCP would


51
52
53
100
101
102
54
103
104
105
106
55
in-order Data
out-of-order Data
cnt Number of out-of-order Data packets received between two in-order Data packets
FRT Fast Retransmit Threshold
Pre_FRT The maximum value of cnt
enter Pre-Recovery
cnt = FRT = 3, Pre_FRT = 3
cnt = 0
cnt = 1
cnt = 2
cnt = 3
cnt = 4
cnt < Pre_FRT + FRT
Pre_FRT = max(cnt, Pre_FRT) = 4;
cnt = 0
Pre_FRT in packets
time (s)

(a)


51
52
54
55
56
57
58
59
in-order Data
out-of-order Data
cnt Number of out-of-order Data packets received between two in-order Data packets
FRT Fast Retransmit Threshold
Pre_FRT The maximum value of cnt
enter Pre-Recovery
cnt = FRT = 3, Pre_FRT = 3
cnt = 4
cnt = 5
cnt = 6 = Pre_FRT + FRT
exit Pre-Recovery &
enter Fast Recovery

(b)

Fig. 1. Detection of Multi-source Arrivals and Lost *Data* in Pre-Recovery

perform in the case of a receipt of a duplicate ACK. Further, *cwnd* is decreased during this period when an in-order *Data* packet is received. This is done due to the consideration that *Interest* packets sent before the *Fast Recovery* for *Data* are not yet received. TCP has a similar behaviour (i.e., decrease of *cwnd*) when a partial ACK (i.e., an ACK that does not acknowledge all previous data) is received.

The recovery during *Fast Recovery* is performed by requesting for the missing data. TCP uses the *Fast Retransmit* algorithm. For CCN, we identify the *Selective Fast Retransmit* algorithm (also called "hole filling" in CCN in [1]) that requests for the missing *Data* packets. In TCP, even though there could be a number of missing data packets, the ACKs will only be sent for the last received in-order data until the next in-order data is received. CCN, on the other hand is in a better position as CCN is able to specifically request for the missing *Data* packets by resending the corresponding *Interest* packets. This process is identified as*Selective Fast Retransmit* in our work.

The exit from *Fast Recovery* occurs when the missing *Data* packets are received for the *Interest* packets sent before the *Fast Recovery* commenced. If an RTO occurs during this period, that too will result in the exit of *Fast Recovery.* Once *Fast Recovery* is exited, the *cwnd* is set to the *ssthresh.*

3.4 *Hop-by-Hop Fairness Control* Algorithm

Every router in CCN forwards the *Interest* packets received, in the direction of the content (when cannot be served from own cache). Therefore, at any given time, there are many different content flows (*Interest* and *Data*) traversing a CCN router. Since the router has limited resources, an aggressive content flow may make it impossible for other flows to receive a fair share of the resources of the router. This becomes acute in situations where CCN applications use FC-CC to retrieve content as the congestion window may increase rapidly.

Therefore, to overcome the unfair utilisation of resources by aggressive content flows, we use a resource allocation algorithm in CCN routers based on max-min fairness [11]. The aim of fair sharing with max-min fairness in our work is to assign a fair share of the use of a face for outgoing *Interest* and *Data* packets.

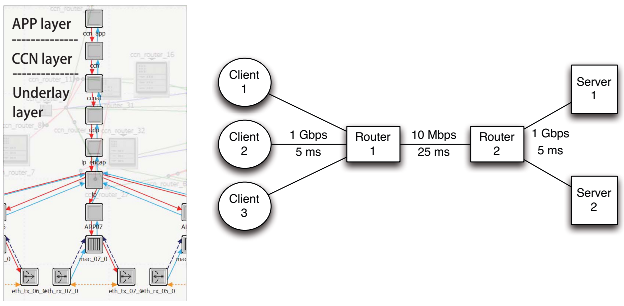
The max-min fairness assigns a fair share of the available bandwidth of a face (i.e., the resource) equally to all the content flows that use that face. The process of assigning the maximum bandwidth is done in an iterative basis. The first iteration equally divides the available bandwidth to all the active flows and the subsequent iterations reassigns the surplus allocations to the deficit allocations. The allocations and the operation is performed in the following manner.

- ∗ Assuming that a content flow has an incoming data rate of X and the fair share is F then the outgoing rate X' should be,
  - $X' = X$ if $X < F$;
  - $X' = F$ if $X \geq F$
- ∗ The data rate X is computed by $X = chunksize/\Delta t$ where *chunksize* is the size of the payload in the *Data* packet and Δt is the time interval between 2 sequentially arriving packets (*Interest* or *Data*)
- ∗ Fairness is applied to both *Interest* and *Data* flows associated with each content flow
- ∗ Since, only the *Data* packets carry a payload, information of the payload size (i.e., *chunksize*) is used to also determine the fair sharing for the corresponding *Interest* flows
- ∗ Flows that have *Interest* or *Data* arrivals above the assigned fair share (aggressive flows) are delayed based on the allocated bandwidth
- ∗ A flow is identified using the content name, sequence number and a nonce that is generated and assigned to the *Interests* a flow by the originator of those *Interests*
- ∗ The fairness assignment is considered for,
  - *Interest* packets that are forwarded after consulting the Forwarding Information Base (FIB)

- *Data* packets that are forwarded after consuming the corresponding Pending Interest Table (PIT) entry
- *Data* packets that are originated from the router itself due to the availability in the Content Store (CS)

4 Performance Results and Analysis

The validation of the adapted TCP flavours and the new algorithms is done using a packet level CCN simulator built in OPNET. The node model consists of 3 protocol layers (Figure 2(a)). The *Application Layer* implements the 3 FC-CC enabled applications (*CCN-NewReno*, *CCN-Compound* and *CCN-Cubic*) that perform *Interest* generation and *Data* consumption functionality. These applications also implement the *Pre-Recovery* and the *Selective Fast Retransmit* algorithms. The *CCN Layer* implements the forwarding mechanisms and the related management functionalities of CCN including the *Hop-by-hop Fairness Control* algorithm. In this simulator, CCN is made to operate over TCP/IP. Therefore, the lowest layer, which we term as the *Underlay Layer* consist of the 4 sub-layers; *Adaptation Layer*, *Transport Layer*, *Network Layer* and the the underlying *Link Layer*. The simulator uses UDP and Ethernet, and the *Adaptation Layer* handles the conversions between UDP and CCN.



(a) (b)

Fig. 2. CCN Node Architecture and Considered Network Topology in the Simulator

There are a number of other capabilities that are built into the simulator. Some of the relevant capabilities are explained below.

CS Policies - There are different *Cache Replacement* policies that can be used by a CS. Among them, Least Recently Used (LRU) is most commonly used in the context of CCN [12,13] and therefore, we have implemented the LRU *Cache Replacement* policy.

PIT Expiration Policy - The PIT registers the *Interests* with a timeout [1]. Since the *Interest* expiration in applications (with FC-CC) may clash with the PIT expirations (e.g., *Interest* packets re-sent by application may not be forwarded by *CCN Layer* due to unexpired PIT entry), we use a 2-level (*soft-hard*) timeout mechanism for PIT. An *Interest* packet received before *soft* timeout results in that *Interest* being only registered in PIT while any *Interest* after, results in it being forwarded as well. A *hard* timeout is considered as in [1].

Forwarding Strategies - [1] proposes 2 forwarding strategies. The *standard* strategy broadcasts received *Interest* packets to all the faces while the *best-face* strategy selects a face based on previous experiences and uses this face to send *Interest* packets out.

The evaluation considers a number of FC-CC enabled application versions that are configured with differing combinations (Table 2) of the algorithms discussed in Section 3 (settings). Therefore, in the following sections, when a reference is made to a performance graph such as *CCN-NewReno-Smart*, it indicates that the graph shows the performance of the *CCN-NewReno* implementation together with the *Fast Recovery*, *Selective Fast Retransmit* and *Pre-Recovery* enabled.

Table 2. Algorithms and Features enabled in Different Settings

Version	Simple1	Simple2	Simple3	Smart
Slow Start	√	√	√	√
Congestion Avoidance	√	√	√	√
Fast Recovery		√	√	√
Selective Fast Retransmit			√	√
Pre-Recovery				√

To evaluate these different FC-CC versions, a network topology is identified that consist of multiple CCN clients, CCN servers and CCN routers (Fig. 2(b)). In each of the following performance evaluations, a scenario is identified using parts of this topology to evaluate a particular version (Table 2) with a particular flavour (e.g., *CCN-NewReno-Simple1*).

4.1 *Fast Recovery* Algorithm

The "Client 1" in Fig. 2(b) requests a Content with a size of 20 MB, which resides on the "Server 1". The "Router 2" drops only one packet at random intervals at different rates. Therefore, there is no continuous packet drops in this setup and the effect of *Selective Fast Retransmit* is not highlighted. And also, out-of-order packets do not occur due to CCN *Data* packets coming from multiple sources since both routers are configured not to cache. CCN *Data* come only from the "Server 1" and thus the effect of *Pre-Recovery* is also not highlighted. These settings are used only to evaluate the effect of *Fast Recovery.* The 3 variants of *CCN-NewReno*, *CCN-Cubic* and *CCN-Compound* without *Fast Recovery*, i.e., "Simple1" and with *Fast Recovery*, i.e., "Smart" is analysed in this section.

Fig. 3 shows *cwnd* variations of *CCN-NewReno*, *CCN-Cubic* and *CCN-Compound*, respectively. Without *Fast Recovery* (i.e., "Simple1"), the receiver ("Client 1") enters *Slow Start* when detecting a packet loss through a retransmission timeout. When using *Fast Recovery* (i.e., "Smart"), a receiver reacts to a packet loss first with *Pre-Recovery* and then enters *Fast Recovery.* As explained in Section 3.3, it continues to increase *cwnd* until the *Pre-Recovery* threshold ($Pre\_FRT + FRT$) is reached and then enters *Fast Recovery*, in which *cwnd*

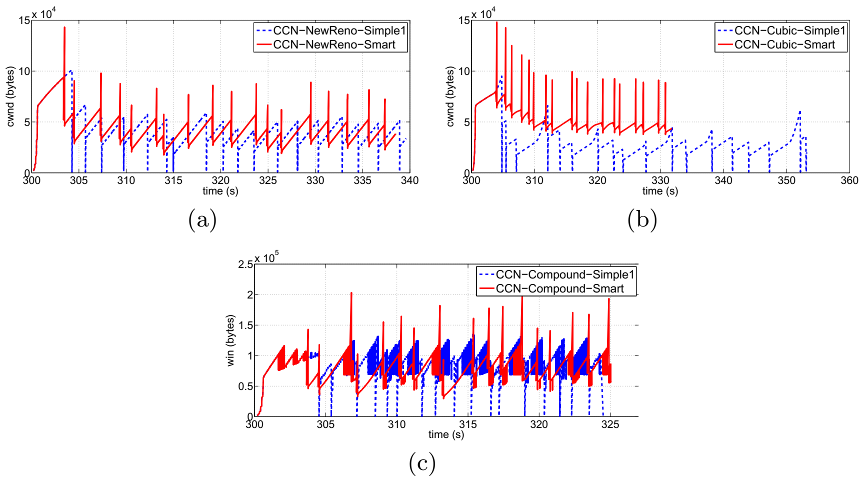


Fig. 3. *cwnd* variations : CCN-NewReno, CCN-Cubic and CCN-Compound with "Simple1" and "Smart" Settings

increases exponentially for receipts of each out-of-order *Data* packet (refers to the spikes between two consecutive *Congestion Avoidance* phases in Fig. 3 with "Smart" version) until it receives an in-order *Data* packet.

The difference between the three adapted variations are how *cwnd* updates during the *Congestion Avoidance* phase. The *CCN-NewReno cwnd* increases linearly during the *Congestion Avoidance* (Fig. 3(a)). *CCN-Cubic* increases in a linear manner immediately after entering the *Congestion Avoidance*, because *CCN-Cubic* emulates regular TCP *cwnd* when it is in the TCP-friendly region [5] (Fig. 3(b)). *Fast Recovery* reduces the download time (Fig. 4) for both *CCN-NewReno* and *CCN-Cubic*. *CCN-Cubic* shows a higher gain with *Fast Recovery* with the increase of the packet loss rates, compared to *CCN-NewReno*.

In contrast to the performance of *CCN-NewReno* and *CCN-Cubic*, *CCN-Compound* shows better performance without *Fast Recovery* (Fig. 3(c). This is due to the consideration of the delay-based window (*dwnd*) in addition to the *win* as determined by *CCN-NewReno*. The *dwnd* of *CCN-Compound* with "Simple1" grows more aggressively than *CCN-Compound* with "Smart". *dwnd* is only effective during the *Congestion Avoidance*. When a packet loss is detected, *dwnd* is set to 0 and *CCN-Compound* goes to *Slow Start* in "Simple1". In contrast, "Smart" version enters *Fast Recovery* which increases the sending of packets resulting in a higher RTT and the decrease of *dwnd*. Therefore, the overall *win* in *CCN-Compound* grows slower with the "Smart" version and this effect is more with the increase of the packet loss rate.

Fig. 4, which shows a comparison of download times of "Simple1" (without *Fast Recovery*) and "Smart" (with *Fast Recovery*) versions when using *CCN-NewReno*, *CCN-Compound* and *CCN-Cubic* under loss rates of 0.1%, 0.2% and 0.5%, confirms the better performance of *CCN-Compound*.

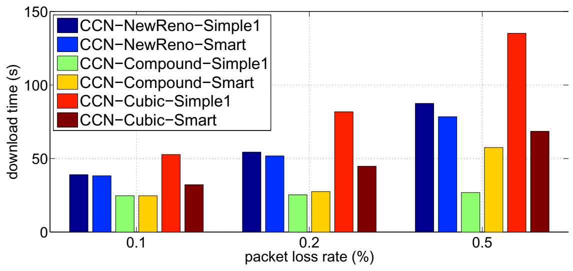


Fig. 4. Download time of *CCN-NewReno*, *CCN-Compound* and *CCN-Cubic* w.r.t. different packet loss rates without *Fast Recovery* ("Simple1") and with ("Smart") *Fast Recovery*

4.2 *Selective Fast Retransmit* Algorithm

The evaluation of *Selective Fast Retransmit* is done using similar versions as in 4.1, but letting "Router 2" in Fig. 2(b) drop multiple *Data* packets continuously at a specified time in order to highlight the effect of *Selective Fast Retransmit*. Here, the comparison is done with "Simple2" and "Smart" versions. In "Simple2", TCP-like *Fast Retransmit* is enabled, while "Smart" is enabled with *Fast Recovery*, *Selective Fast Retransmit* and *Pre-Recovery*. Note that, *Pre-Recovery* is not triggered in this scenario since there are no out-of-order *Data* packets due to the use of a single server in this scenario.

We have compared the recovery time in Fig. 5, which shows the average time that a CCN application stays in *Fast Recovery* when packet losses occur. In case of one packet loss, both "Simple2" and "Smart" shows exactly the same time in *Fast Recovery* for all 3 variants (*CCN-NewReno*, *CCN-Compound* and *CCN-Cubic*). But, recovery time increases drastically with the increase of multiple packet drops for "Simple2", while "Smart" stays almost the same as the single packet drop case.

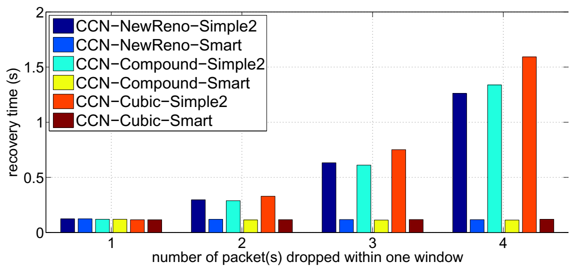
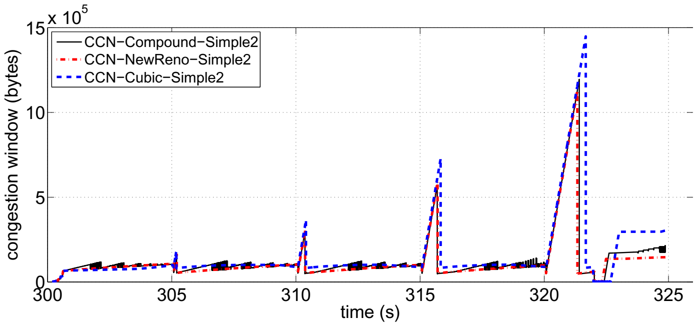
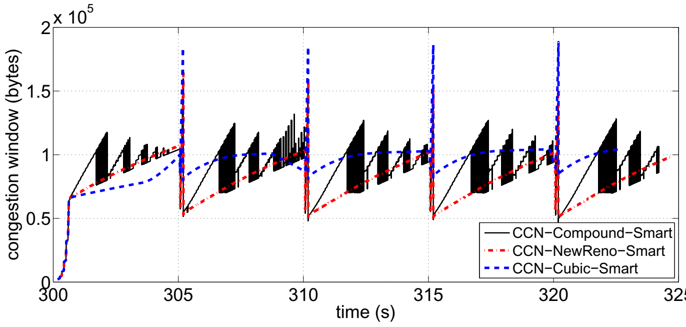


Fig. 5. Recovery time of CCN-NewReno, CCN-Compound and CCN-Cubic w.r.t. number of continuous packet losses

Fig. 6 compares how *cwnd* varies when dropping packets, 1 packet at 305 s, 2 packets at 310 s, 3 packets at 315 s and 4 packets at 320 s. Fig. 6(a) shows that *cwnd* of *Fast Recovery* increases rapidly when the number of packets lost increases continuously, when the *Selective Fast Retransmit* is not used ("Simple2"). In contrast, when *Selective Fast Retransmit* is used ("Smart"), Fig. 6(b) shows that the variations of *cwnd* is not dependant on the number of packet losses. As explained in Section 3.3, this is due to *Selective Fast Retransmit* of CCN continuously sending *Interest* packets of the missing *Data* packets. This is not the case with TCP-like *Fast Retransmit* that waits for in-order data to arrive for the next ACK to be sent.



(a)



(b)

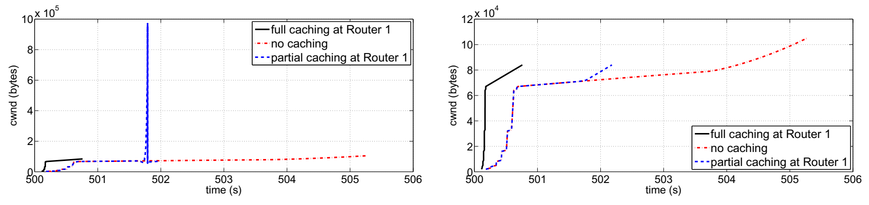
Fig. 6. *cwnd* variations without ("Simple2") and with ("Smart") *Selective Fast Retransmit*

4.3 *Pre-Recovery* Algorithm

The setup shown in Fig. 2(b) is used to evaluate the performance of *Pre-Recovery* by distributing *Data* packets among multiple sources. At the beginning, "Client 1" starts downloading contents while "Router 1" is set to cache segments using LRU caching strategy. "Client 2" is made to start downloading the same content a little later and when "Client 2" starts downloading, "Router 1" has full or part of *Data* packets depending on pre-configured cache sizes. All the results shown here are taken at "Client 2", which gets out-of-order packets due to *Data* packets coming from the "Server 1" as well as from the "Router 1". The results are taken for 3 cases, viz., where all *Data* packets are cached at "Router 1" (full caching), where all *Data* packets are at "Server 1" (no caching) and where the latter part of *Data* packets are cached at "Router 1" (partial caching).

The comparison is done with "Simple3" version, in which *Pre-Recovery* is disabled and "Smart" with *Pre-Recovery* enabled. In this scenario, the effects of packet drops are not emulated.

With TCP-like *Fast Recovery*, even without packet drops, "Simple3" detects a false packet loss through out-of-order *Data* receipts (Fig. 7(a)) and enters *Fast Recovery* immediately, when using partial caching. Fig. 7(b) shows that the *Fast Recovery* is not triggered immediately when *Pre-Recovery* is enabled. When *Data* packets arrive from a closer source (i.e., "Router 1"), *Pre-Recovery* makes sure that *Fast Recovery* is not triggered until it reaches the *Pre-Recovery* threshold (Section 3.2). Therefore, *cwnd* follows the same upward climb as in full caching at "Router 1" when *Data* packets are received from a closer source. At the beginning, *cwnd* variation of partial caching follows a similar trend as in no caching (due to the initial *Data* packets coming from the "Server 1") until it detects the receipt of out-of-order *Data* packets.



(a) (b)

Fig. 7. *cwnd* of CCN-cubic without ("Simple3") and with ("Smart") Pre-Recovery

4.4 *Hop-by-Hop Flow Fairness* Algorithm

The performance of *Hop-by-hop Flow Fairness* is evaluated using a scenario based on the topology in Fig. 2(b). In this scenario, both "Client 1" and "Client 2" download the same content from "Server 1", but with "Client 1" starting initially and "Client 2" a little later. LRU based caching is enabled in the "Router 1" and therefore, the download done by "Client 2" initially obtains the content from the cache at "Router 1". The results are compared with and without using the *Hop-by-hop Flow Fairness* algorithm, when using *CCN-NewReno* with different levels of Background Traffic Loads (BTL) in the network. The BTL is created as a percentage of the link capacity.

Table 3 shows the performance comparison of without *Hop-by-hop Flow Fairness* (Case 1) and with *Hop-by-hop Flow Fairness* (Case 2), for the different

Table 3. Hop-by-hop Fairness Performance Comparison

	Client 1				**Client 2**			
	No BTL		80% BTL		No BTL		80% BTL	
	Case 1	Case 2	Case 1	Case 2	Case 1	Case 2	Case 1	Case 2
Download time, sec	24.19	24.19	230.13	180.5	23.43	23.43	208.44	209.44
Throughput, Kbps	826.8	826.8	86.9	110.8	853.4	853.4	95.9	95.5
Packet losses	0	0	26	0	0	0	0	0

clients under no BTL and with an 80% of BTL. Due to the cached content in "Router 1", the FC-CC application in "Client 2" becomes aggressive as the content can be retrieved faster, initially. During this initial period, since the RTT is smaller, the *cwnd* grows rapidly, thereby becoming aggressive. But, after a while, when the cache is exhausted, "Client 2" also starts fetching the content from the "Server 1". In this way, "Client 1" is disadvantaged. But when *Hop-by-hop Flow Fairness* operates at the "Router 1", the aggressive flow is slowed down by limiting the bandwidth it is able to use. Thereby, both content flows are given a fair share of the bandwidth. This is evident from the better performance (lower download time, better throughput and no packet losses) experienced by "Client 1" for 80% BTL. One other observation made was that the benefits of *Hop-by-hop Flow Fairness* becomes more evident when the amount of BTL increases which is the usually expected when networks are congested (higher BTL).

5 Conclusion

The work presented here discussed about the adoption of FC-CC in CCN based networks. Our contribution focussed on adapting the most widely used TCP flavours of *CCN-NewReno*, *CCN-Cubic* and *CCN-Compound*. We have discussed a number of aspects that need to be considered when adapting TCP due to the architectural differences in CCN compared to TCP/IP, the adaptation of these 3 flavours and the additional algorithms (*Pre-Recovery*, *Selective Fast Retransmit* and *Hop-by-hop Flow Fairness*) identified to address some of the issues relevant to CCN. The *Pre-Recovery* algorithm was identified to avoid the false detection of packet losses in CCN when content arrives from multiple sources. The *Selective Fast Retransmit* algorithm was identified to perform selective resending of *Interest* packets during the loss recovery period. The *Hop-by-hop Flow Fairness* algorithm that considers both *Interest* and *Data* flows was identified to provide fair sharing of bandwidth for competing content flows. The flavour adaptations

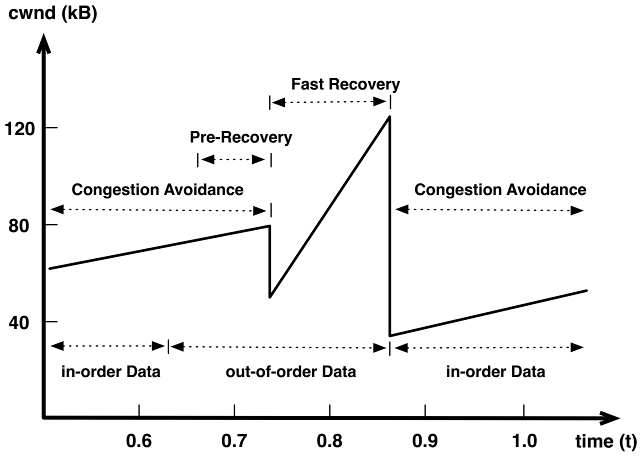


Fig. 8. Operation of the Different FC-CC Algorithms

and the identified algorithms were implemented in an OPNET based simulator and the performance was evaluated. Fig. 8 shows the operation of the different algorithms discussed in our work.

The analysis of the results obtained show that *Fast Recovery* makes *CCN-NewReno* and *CCN-Cubic* more efficient in downloading content under stable loss rates. However, *CCN-Compound* has a larger throughput when losses are only detected from a retransmission timeout. The *Selective Fast Retransmit* algorithm in CCN has a better recovery efficiency than TCP-like *Fast Retransmit*. The *Pre-Recovery* shows the advantages of detecting whether out-of-order *Data* packets are as a result of multiple sources or packet losses. *Hop-by-hop Flow Fairness* enhances the performance of less aggressive flows in terms of goodput and download efficiency by providing a fair share of the use of faces.

We intend to further improve the performance of our work by evaluating these scenarios in large scale network topologies.

References

1. Jacobson, V., Smetters, D.K., Thornton, J.D., Plass, M.F., Briggs, N.H., Braynard, R.L.: Networking named content. In: Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies, pp. 1–12. ACM (2009)
2. Zhang, L., Estrin, D., Burke, J., Jacobson, V., Thornton, J.D., Smetters, D.K., Zhang, B., Tsudik, G., Massey, D., Papadopoulos, C., et al.: Named data networking (NDN) project, NDN-0001, Xerox Palo Alto Research Center-PARC (2010)
3. Floyd, S.: The NewReno Modification to TCPs Fast Recovery Algorithm, RFC3782 (April 2004)
4. Tan, K., Song, J., Zhang, Q., Sridharan, M.: A Compound TCP Approach for High-speed and Long Distance Networks. In: IEEE Infocom, Barcelona, Spain (April 2006)
5. Ha, S., Rhee, I., Xu, L.: CUBIC: A New TCP-Friendly High-Speed TCP Variant. ACM SIGOPS Operating System Review (2008)
6. Allman, M., Paxson, V., Stevens, W.: TCP Congestion Control, RFC 2581 (April 1999)
7. Oueslati, S., Roberts, J., Sbihi, N.: Flow-aware traffic control for a content-centric network. In: 2012 Proceedings IEEE INFOCOM. IEEE (2012)
8. Rozhnova, N., Fdida, S.: An effective hop-by-hop Interest shaping mechanism for CCN communications. In: Computer Communications Workshops (INFOCOM WKSHPS). IEEE (2012)
9. Carofiglio, G., Gallo, M., Muscariello, L.: Joint hop-by-hop and receiver-driven interest control protocol for content-centric networks. In: Proceedings of the Second Edition of the ICN Workshop on Information-centric Networking. ACM (2012)
10. Palo Alto Research Centre, Draft CCN Protocol Specification, `http://www.ccnx.org`, (accessed on May 2013)
11. Keshav, S.: An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network. Addison-Wesley Professional (1997)
12. Psaras, I., Clegg, R.G., Landa, R., Chai, W.K., Pavlou, G.: Modelling and Evaluation of CCN-Caching Trees. In: Domingo-Pascual, J., Manzoni, P., Palazzo, S., Pont, A., Scoglio, C. (eds.) NETWORKING 2011, Part I. LNCS, vol. 6640, pp. 78–91. Springer, Heidelberg (2011)
13. Muscariello, L., Carofiglio, G., Gallo, M.: Bandwidth and Storage Sharing Performance in Information Centric Networking. In: ACM SIGCOMM Workshop on Information-centric Networking, pp. 26–31 (2011)