Escape Cache Traps by Rate Feedback for NDN Real-time Video Streaming

Zhaohua Zhu*†, Yongrui Chen‡, Linggang Li§, Zhijun Li§, Weizhe Zhang†, Yu Zhang†
* Harbin Institute of Technology, Shenzhen, China § Harbin Institute of Technology, Harbin, China
† Pengcheng Laboratory, Shenzhen, China
‡ University of Chinese Academy of Sciences, Beijing, China
zhaohua_zhu@163.com, chen_yong_rui@163.com, lizhijun_os@hit.edu.cn

***Abstract*—In-network caching is one of the most important characteristic of Named Data Networking (NDN). However, while replacing producers in responding to interest requests, caching data packets also shields consumers from perceiving the bottleneck bandwidth of the transmission path between the producer and the consumer. Therefore, when the content source switches from the cache node to the producer due to data exhaustion, the consumer can not adjust the requesting rate accordingly, and may lead to the serious bufferbloat or packet loss - we call it as Cache Trap. We found that Cache Trap occurs commonly in streaming services and the state-of-art NDN congestion control schemes cannot achieve efficient and stable quality of service when it happens.**

To escape Cache Trap, this paper proposes an explicit rate feedback congestion control algorithm, named as RFCC. RFCC leverages NDN routers' ability of encapsulating customized information in data packets to send link state information to consumers. Specifically, when responding to interest packets, RFCC nodes estimate data throughput received from the producer and insert this information into the returned data packets. The consumer perceives the change of content source according to the hopcount tag in data packet, and then adjusts the sending rate of interest packet based on the explicit rate information. We have implemented RFCC in both real-world NDN live video streaming and NDNsim simulation platforms, and compared it with the state-of-arts congestion control algorithms in a variety of scenarios. The experimental results show that when Cache Trap occurs, RFCC maintains a stable QoE in live video streaming, reduces 50% delay jitters compared with DPCCP and achieves 2.4× throughput compared with PCON.

***Index Terms*—named data networking, congestion control, live streaming, NDN**

I. Introduction

Nowadays, the streaming businesses, mainly short videos and live streaming, have been growing by leaps and bounds. According to statistics [1], the Video Streaming (SVoD) market for worldwide is expected to reach $108.50 billion in 2024. The fast-growing market scale also puts forward higher requirements for the Quality of Experience (QoE) of various video applications. Driven by this, Named Data Networking (NDN), as a new network structure, has been increasingly applied to video services in the research community [2] [3] [4] [5] with its decentralized architecture and support for in-network storage. In addition, the latest work has deployed NDN into the kernel of the operating system [6], and it is foreseen that NDN will play an increasingly important role in future network architectures due to its advantages in accelerating the data transfer for streaming services.

NDN is a content-driven network [7]. In NDN, when a user (consumer) is going to acquire a content, it does not need to establish a connection to a specific host. Instead, it sends a message (interest packet) identified by the name of the content to the network. The holder of the content (producer) responds to the arriving interest request by encapsulating the content in packets and then returning them to the consumer along the opposite path. During the transmission, these packets will be temporarily cached in all NDN nodes it passes through, so that if another consumer wants to request the same content, the caching node will respond to the interest packet immediately, thus speeding up the response to the consumer and relieving the pressure on video servers.

Although efficient, there is still a defect in the state-of-art NDN congestion control algorithms. Especially, when the cached data on the forward node has been sent out, the interest packets will be forwarded to the producer to acquire contents from it. However, the consumer cannot perceive the current bottleneck bandwidth of the transmission path from the producer to the consumer. As a result, the consumer will keep the previous interest sending rate, which is much higher than the bottleneck bandwidth, and are unable to adjust it to an appropriate rate. Too many interest packets will inevitably lead to too fast data transmissions, and lead to bufferbloat or even packet loss on the bottleneck node. This will in return make the congestion control cut down the requesting rate sharply, leading to the under-utilization of links and unstable of transmission performances (i.e., delay, throughput). We call this as Cache Trap.

To be more specific, Cache Trap occurs in the following two generic situations in NDN real-time video streaming: i) In many-to-one scenarios, due to the interest aggregation property of NDN [7], there is commonly one consumer retrieving data from the producer through the transmission path, and other consumers will obtain data from the cache node. Since the cache node can provide faster response rate for interest packets than the producer, consumers who retrieve content from the cache node will inevitably exhaust the data in the cache at some time and switch to request content from the producer.

Yongrui Chen‡ and Weizhe Zhang† are both corresponding authors.

However, their requesting rate still remains high and will lead to too fast data transmissions. ii) In one-to-one scenarios, sometimes the consumer may not receive a complete video segment in one synchronization period of media playlist file [8], e.g., due to link losses, buffer overflow, etc,. In this case, the consumer will request the same segment again in the next synchronization period from the cache node. This will lead to the same situation of data exhaust and source switch as in i). What's worse, this process may repeat again and again, and pose a risk of frequent QoE degradation.

To escape Cache Trap, this paper proposes RFCC, a congestion control algorithm based on explicit rate feedback for NDN. First, RFCC utilizes the path identification mechanism to set path-specific sending rate of interests in NDN. Then, in order to perceive accurate bandwidth of the complete transmission path from producer to consumer, RFCC enables the NDN cache node that stores the cache content to explicitly inform the consumer of the calculated caching rate by embedding this information into the data packet header. In addition, in cache node, RFCC designs a bandwidth update strategy based on Coefficient of Variation (CV) to avoid updating inaccurate bottleneck bandwidths when the data flow is unstable. Finally, RFCC consumers perceive the changes of the content sources according to the hopcount field in NDN header, and then adjust sending rate of interest packets based on the explicit feedback values when the source change occurs.

The contributions of this work are summarized as follows:

- We revealed the problem of Cache Trap in NDN live streaming applications, and analyzed its reason in detail. To cope with it, we present RFCC, a NDN congestion control algorithm based on explicit rate feedback. To the best of our knowledge, this is the first congestion control strategy which is designed for Cache Trap.
- To enable RFCC escape Cache Trap, we address several challenges, including i) how to accurately capture the caching rate of different data flows in NDN cache node with rapidly changing traffic; ii) how the consumer perceives changes of content source during transmission; iii) how to accurately adjust the interest sending rate of consumers when source switching occurs.
- We implemented RFCC in both real-world NDN platform and NDNsim simulation platform, and compared it with existing congestion control algorithms in various scenarios. The experimental results show that RFCC maintains a stable QoE in live video streaming, reduces 50% delay jitters compared with DPCCP and achieves 2.4× throughput compared with PCON.

II. Background

A. NDN Congestion Control

NDN congestion control is based on the delay or throughput of the received data packets at the consumer, to adjust the sending rate of interest packets accordingly to reduce congestion or improve link utilisation. For early NDN congestion control algorithms [9] [10], the consumer can not distinguish the source of contents, and treat the received packets as a whole stream. However, there are different response sources in a NDN transmission process, this uniformly setting of sending rate will not fully utilize the bandwidth. For example, if one of the transmission paths is congested, while the others are still under-utilized. the consumer will enter a dilemma of whether or not to reduce the requesting rate. Recent studies have proposed path specific mechanisms to distinguish the packet sources [11] [12] by recording the transmission paths and controlling the send rate on each path separately.

B. Path Identification Mechanism

The path identification mechanism defines the transmission path in the NDN network, that is, the transmission path starts with the consumer and ends with the producer [11] [12]. When a consumer is first connected to the network, it broadcasts a special interest packet to detect potential producers in the network, and the NDN node forwards the packet returned by the producer by pressing its own node information into a stack structured path label, so that the consumer can extract the complete path from this label when it receives the packet, and specify this path in all subsequent interest packets. Based on this label, the consumer can maintain the congestion information (e.g., RTT) for each path independently. Only when the congestion information under the same path changes, the consumer adjusts the interest packet's send rate for this path.

III. Problem Analysis

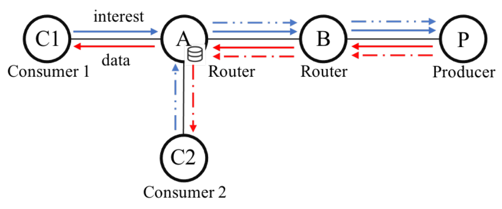


Fig. 1. A simple scenario to demonstrate the Cache Trap problem.

As shown in Fig. 1, 2 consumers, i.e., C1 and C2 are connected to the producer P through routers A and B, consumer C1 is continuously sending interest requests for video stream, and these interest packets are passed through path C1-A-B-P to the producer P. The data packets returned from P are transmitted along the opposite path to C1 and cached in A and B. After a period of time, Consumer C2 starts requesting the same video stream. Since the video content has been cached in node A, the initial interest packet sent by consumer C2 is responded by router A, and then two possible cases follow. i) If the bottleneck of path P-C2 lies in the link A-C2, i.e., the bottleneck is the downstream of the cache, the data receiving rate of router A will be greater than the interest requesting rate from consumer C2, and the interests from C2 will be always responded by router A until the video ends. In this case, Cache Trap will not happen. ii) If the transmission bottleneck of path P-C2 lies between P-A, i.e., the bottleneck is the upstream of the cache, e.g., the link B-A, the interest requesting rate from consumer C2 will be greater than the data receiving rate of router A. In this case, the cached content at node A will be exhausted at some time, and the content source of C2 will be taken over by producer P. However, when the content source changes, consumer C2 still maintains the previous high interest sending rate, resulting in bufferbloat or packet loss at the bottleneck node. We note that in the this case, the caching rate/data receiving rate of router A will be equal to the bottleneck bandwidth on path C2-P since the bottleneck exists in the path P-A. Although the consumers are unable to perceive the bottleneck bandwidth of the entire transmission path, the cache node can calculate it from the data receiving rate and send it to the consumers. This is the key insight of our design.

IV. Design of System

A. System Overview

The system of RFCC is shown in Fig. 2. Consumers in RFCC firstly fill bottleneck on each path with a fixed gain coefficient and increase sending rate of interest packets until they perceive congestion from changes in transmission delay. Then, consumers will enter congestion regulation phase, which adjust interest sending rate with local statistical information, to adapt to the congestion level. During the transmission, consumers identify content source according to the value of hopcount tag in data packets. Once the content source changes, consumers will update interest sending rate immediately according to the value of the rate tag in data packets. RFCC routers forward interest packets based to the path identification mechanism and estimate data caching rate of each flow when the request content not cached locally. Once the request content is cached in cache node, the data caching rate on the specific path will be encapsulated in data packet as a rate tag and returned to the consumer.

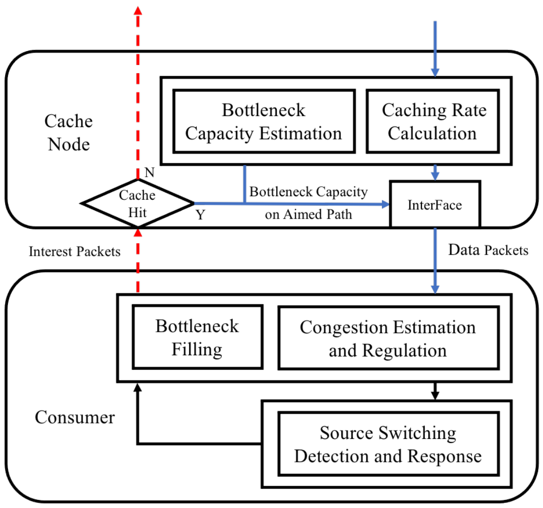

Cache Node
Bottleneck Capacity Estimation
Caching Rate Calculation
N
Cache Hit
Y
Bottleneck Capacity on Aimed Path
InterFace
Interest Packets
Data Packets
Bottleneck Filling
Congestion Estimation and Regulation
Consumer
Source Switching Detection and Response

Fig. 2. Design overview.

B. Cache Node

Caching Rate Calculation: In RFCC, a cache node who responds the interest packets of a given flow calculates its data caching rate according to a sliding window. Specifically, for the cache node on transmission path p, denote its hop to the consumer as h, the caching rate on this node is calculated as:

$$c_{p,h} = \alpha * \frac{N}{(t_{tail} - t_{head})} + (1-\alpha) * c_{p,h,last} \tag{1}$$

Where $c_{p,h}$ is the caching rate, N is the size of the sliding window, t_{tail} and t_{head} are the timestamp when the first and last packet of these N packets are received, respectively, $\alpha \in$ [0,1] is use to control the decay of historical values, which is typically set as 0.75. We note that the value of N directly affects the accuracy of the estimated throughput. If it is too short, the error in the calculated throughput may be large. And if the window size is too long, the calculated result cannot effectively reflect the fluctuation of throughput. We have conducted extensive experiments to decide the optimal N. In our implementation, the value of N is set as 48.

Bottleneck Capacity Estimation: We note that in HLS (Http Live Stream protocol) [8] based video applications, consumers do not continuously request video streams, but request video segments. When all the interest packets which request the current video segment have been sent out, the consumer does not immediately send the interest packets to request next segment, but pauses until receiving the complete video segment. This will cause a jitter in the interest flow sent by the consumer. Since the data packet is returned as a response to the interest packet, the data flow sent by the content source will also experience a jitter. In this case, the caching rate calculated by the cache node cannot reflect the actual bottleneck capacity, and the cache node cannot obtain an accurate pause time to correct the estimated caching rate.

Therefore, in RFCC, we designed a strategy to update bottleneck capacity $Bc_{p,h}$. Specifically, after calculating the caching rate, the node first determines whether the interest flow and the returned data flow on path p are stable. If both of them are stable, the node updates $Bc_{p,h}$ with Eq.(1). Or else, the node does not update it. We note that the pause in requesting process produces a outlier in the arrival time interval of the interest flow or the data flow. In order to detect stability of the packet flow, we adopt a dispersion indicator in statistics, coefficient of variation [13], which is defined as the ratio of the standard deviation to the mean. The cache node keeps track of adjacent arrival interval of recent N interest packets and data packets, and stores them in vector A and D respectively. When receiving an interest packet or a data packet, the coefficient of variation of interest flow $Cv_{p,h}(A)$ and data flow $Cv_{p,h}(D)$ are calculated as:

$$Cv_{p,h}(A) = \frac{std(a_i)}{mean(a_i)} \tag{2}$$

$$Cv_{p,h}(D) = \frac{std(d_i)}{mean(d_i)} \tag{3}$$

Here, a_i and d_i denotes the time interval of adjacent interest packets and data packets respectively, std and $mean$ operator denotes standard deviation and mean value. When the coefficient of variation of a sequence is less than 1, it indicates that the standard deviation of the sequence is smaller than its mean, and the sequence is considered low-variance. Thus in RFCC, when the CV value of A and D are both less than 1, we consider the interest flow and data flow are stable and update the bottleneck capacity with calculated caching rate. In summary, when receiving a data packet, the bottleneck capacity on cache node is updated as follows:

$$Bc_{p,h} = \begin{cases} c_{p,h} & if\ Cv_{p,h}(A) < 1\ \&\ Cv_{p,h}(D) < 1 \\ Bc_{p,h_{last}} & otherwise \end{cases} \tag{4}$$

Interest Packets Forwarding/Responding: When receiving an interest packet, the aimed transmission path, which is encapsulated in the interest packet head according to path-specific mechanism, is firstly extracted by the cache node. And then, the cache node considers the following two scenarios. If it does not have the request content of the interest packet, the cache node will send the interest packet to the specified path. Or else, the cache node will return the data packet with the requested content, and explicitly inform the consumer of the bottleneck capacity on the aimed transmission path by embedding this information into the returned data packet according to NDN router's ability of encapsulating customized information in the head of NDN packets [14].

C. Consumer

Bottleneck Filling: Traditional congestion control algorithms fill up the link quickly by continuously increasing the sending rate until congestion is detected. However, due to the lag in the return of congestion information, by the time congestion is detected at the sender, this continuously increasing sending rate has grown to an excessive value, leading to bufferbloat problems. To mitigate this issue, we design a 'detect-and-wait' mechanism. Specifically, RFCC divides each probing round into two stages and keeps a tolerable sending rate at the beginning of each probing round. In the first stage, the sending rate of this round is increased by a fixed gain factor according to the tolerable value and maintains one RTT to probe if the bottleneck can tolerate the increased sending rate. In the second stage, the sending rate is recovered to the tolerable value and also holds for one RTT to wait for the congestion response from the previous stage. Meanwhile, the minimum RTT value in the second stage will be recorded as baseRTT. In summary, the sending rate sr_p on path p in each probe round can be expressed as follows:

$$sr_p = \begin{cases} sr_p^* * (1+\beta) & stageI \\ sr_p^* & stageII \end{cases} \tag{5}$$

Where sr_p^* is the tolerable sending rate of the path which has been probed in the last round, and β is the gain factor used to increase the interest sending rate. In this paper, we set β to 0.25, which is consistent with the probe bandwidth growth rate in the classical congestion control algorithm BBR [15]. When consumer perceiving congestion from local statistics, it indicates that the probe rate of the previous stage has filled up the link, so RFCC immediately exits the bottleneck filling state and enters to the congestion regulation phase. Otherwise, the tolerable sending rate which will be adopted in the next round is increased to the current stage I interest sending rate to continue filling the bottleneck until the link is full: [1]

$$sr^*_{p,next\_round} = sr_p^* * (1+\beta) \tag{6}$$

Congestion Estimation and Regulation: The congestion state in RFCC is estimated according to the number of backlogged packets [12]. Note that consumers in NDN cannot specify the response source of interest packets, when a consumer sends interest packets to a path, there may be multiple returned data flows along this path. For the data flow on transmission path p, we denote its hop to consumer as h, and denote the set of returned data flows as $H(p)$. When receiving a data packet, RFCC consumer calculates the number of backlogged packets $\delta_{p,h}$ of flow $f_{p,h}$ according to RTT and baseRTT, that is:

$$\delta_{p,h} = (\hat{Rtt}_{p,h} - baseRtt_{p,h}) * r_{p,h} \tag{7}$$

Here, $\hat{Rtt}_{p,h}$ is the averaged RTT, calculated by Exponentially Weighted Moving Average (EWMA) of RTTs of recent 8 packets and $baseRtt_{p,h}$ is the baseRTT of hop h which is measured before stage II. $r_{p,h}$ denotes the data receiving rate from content source with hop h. For each flow, $r_{p,h}$ is calculated the same way as $c_{p,h}$ (except is calculated by the consumer), as shown in (8).

$$r_{p,h} = \alpha * \frac{N}{(t_{tail} - t_{head})} + (1-\alpha) * r_{p,h,last} \tag{8}$$

Then, the backlogged packet on transmission path Δ_p is calculated as the summation of the number of backlogged packets $\delta_{p,h}$ on each flow as shown in (9).

$$\Delta_p = \sum_{h \in H(p)} \delta_{p,h} \tag{9}$$

During the transmission process, when the congestion value of transmission path Δ_p is larger than a backlog threshold Δ_1, RFCC consumer moves from bottleneck filling state to congestion regulation state. In this state, the consumer uses an Adaptive Additive Increase Additive Decrease algorithm (A-AIAD) [12] to adjust its interest sending rate on path p to make the backlogged packets Δ_p converge to the setpoint Δ_2, which is shown in (10).

$$sr_p = \begin{cases} r_p + \rho & \Delta_p < \Delta_2 \\ r_p - \rho & \Delta_p > \Delta_2 \end{cases} \tag{10}$$

Here, r_p is the data receive rate on path p, which is calculated as the summation of the data receive rate on each

[1]We set the tolerable send rate to 10 interest packets per second when a new transmission has just started.

flow $r_{p,h}$, and ρ is the adaptive factor adjusted according to the expected convergence time and the actual convergence speed. These two params are calculated as follows:

$$r_p = \sum_{h \in H(p)} r_{p,h} \tag{11}$$

$$\rho = \begin{cases} 1 & if\ (\Delta_p - \Delta_2) * (\Delta_2 - \Delta_{p,last}) \geq 0 \\ \rho_{last} + 1 & if\ |\Delta_p - |\Delta_{p,last\_K}||/K < 1 \\ \rho_{last} - 1 & if\ |\Delta_p - |\Delta_{p,last\_K}||/K > 1 \end{cases} \tag{12}$$

Here,$\Delta_{p,last}$ is the last value of Δ_p, $\Delta_{p,last\_K}$ is the historical value before K baseRTT cycle, and $|\Delta_p - \Delta_{p,last\_K}|/K$ is the average convergence speed of K non-overlapping baseRTT periods. This means if the adjacent congestion value pass through the target rate Δ_2, is reset to 1, if the changing rate of congestion value Δ_p in K baseRTT periods is smaller than 1, ρ is increased, otherwise ρ is reduced [12]. In congestion regulation state, the sending rate is updated every RTT period, and the adaptive factor is updated K non-overlapping baseRTT periods.

Source Switching Detection and Response: For each transmission path p, to track the content source of the data flow and to perceive changes on content source, the RFCC consumer keeps track of hopcount tag values for recent received ψ data packets, and puts them in a vector U. When receiving a data packet, the consumer uses the *mode* operator to take the value that appears most often in vector U as the content source cs_p of path p, that is:

$$cs_p = mode(U) \tag{13}$$

After calculating cs_p, if the last content source $cs_{p,last}$ is not equal to cs_p, the consumer steps into the source switching response state. When entering this state, the consumer first clears the values recorded in U, and then, we consider two different situations. First, if cs_p is smaller than $cs_{p,last}$, which indicates that most of the recent data comes from a closer content source. For example, if another consumer is requesting content at a faster rate, which makes the current consumer have to download data packet from a cache node. In this case, as discussed in Sec. III, the bottleneck of the new data flow is no smaller than that of the old one, therefore the RFCC consumer directly assigns the current data flow's congestion value to the path's congestion value, as shown in (14).

$$\Delta_p = \delta_{p,h} \tag{14}$$

Then the RFCC consumer maintains the current interest sending rate and return to the state before entering the source switching state. Second, if cs_p is greater than $cs_{p,last}$, it indicates that most of the recent data comes from a farther source, which means that the cached content at a closer source is likely to have exhausted. In this case, to avoid Cache Trap issues, it is necessary to update the interest sending rate according to the explicitly feedback bottleneck capacity. Therefore, we set sr_p as $Bc_{p,h}$ and keeps it for a RTT round, as shown in (15).

$$sr_p = Bc_{p,h} \tag{15}$$

After that, the RFCC consumer exits the response state and then determines the next enter state based on current Δ_p, if no congestion occurred, it enters the bottleneck filling state, otherwise it enters the congestion regulation state.

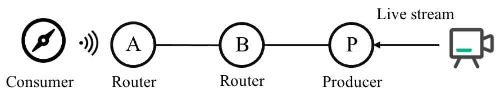
The value ψ ensures that consumer does not react to temporary changes in content source and we have conducted extensive experiments to decide the optimal ψ. We found from our experimental results that the setting of this parameter directly affects the transmission performance of the network. In our implementation, we set ψ as 32, which can perceive changes in the content source in a timely manner and ensures the stability of the transmission.

V. Evaluation

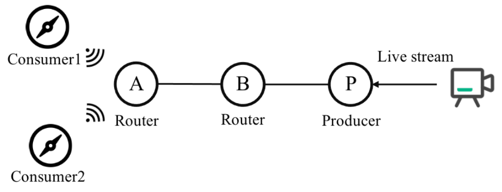
A. Experimental Setup

We implemented RFCC in both NDN live-streaming platform based on HLS protocol [8] and NDNsim simulation platform [16]. As shown in Fig. 3, we deployed RFCC in an open-source video engine shaka-player [17] with version 3.3.9 to make consumers watch live videos through NDN networks, and deployed a NDN producer at a laptop computer to publish video content. There are two forward nodes, A and B, which runs NDN forward daemon (NFD) [18] to support NDN packet forwarding. The video producer provides video content for 3 different resolutions - 240p, 480p and 720p. The packet size of each NDN packet is set to 8000 bytes, which is based on the common settings in NDN live streaming applications [19]. As for NDNsim platform, we simulate the data request pattern of consumers in HLS based live streaming applications, where consumers request continuous data segments instead of continuous data streams from the network. The packet size is also set to 8000 bytes, and the size of each data segment is set to 1500 packets. Due to the limitation of our video server, experiments before Sec. V-E are performed in the real platform, and experiments after Sec. V-E are performed in the NDNsim simulator.

We compare RFCC with PCON [10] and DPCCP [12] in different scenarios. Among them, PCON is a classical congestion control algorithm in NDN, which uses the active queue management algorithm CoDel [20] to mark congestion at the NDN node. It supports three modes of congestion regulation, namely AIMD [21], BIC [22], and Cubic [23] modes. DPCCP is a recently proposed congestion control algorithm based on path identification, which performs individual congestion control on each path and estimates the congestion status of the link based on the change of RTT. We perform 10 tests for each congestion control algorithm in each scenario separately, and then analyze the performances in terms of the video playback resolution, transmission delay and data receive rate when the content source switches. The gain factor β is set to 0.25 [15], K is set to 3 and the regulation threshold Δ_1, Δ_2 of RFCC are set to 2 and 20 by default [12].

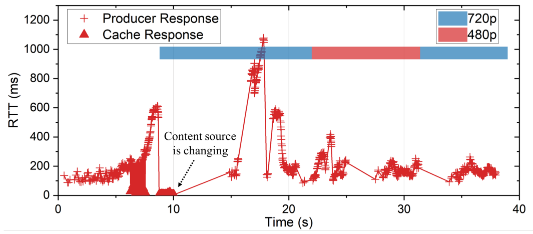


(a) One-to-one scenario.

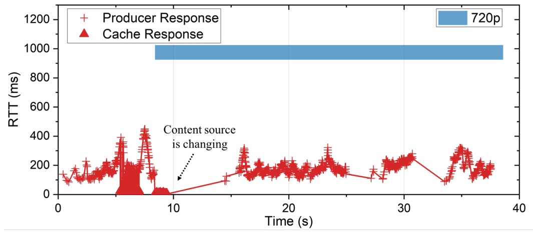


(b) Two-to-one scenario.

Fig. 3. Deployment of network topology in real platform.

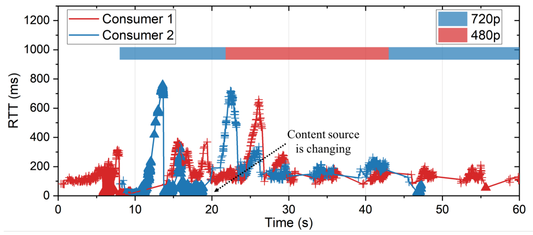


(a) The latency and video resolution of DPCCP.

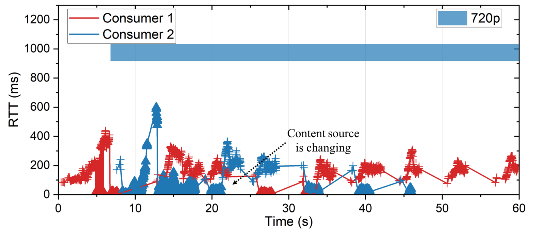


(b) The latency and video resolution of RFCC.

Fig. 4. The latency and video resolution of different congestion control algorithms in one-to-one scenario.



(a) The latency and video resolution of DPCCP.



(b) The latency and video resolution of RFCC.

Fig. 5. The latency and video resolution of different congestion control algorithms in two-to-one scenario.

B. Performances in One-to-one Scenario

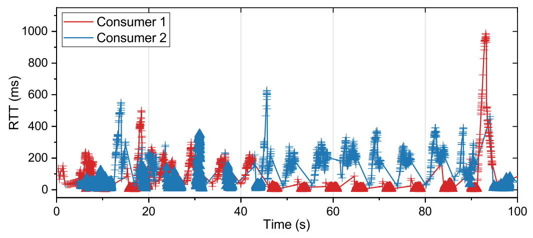
In this section, we analyze the performances of RFCC when a consumer starts to request video content. We note that due to the characteristics of the shaka-player, consumers using this video engine will repeatedly request the first video segment when accessing video content, thus the interest packets for this part will be responded by the cache node since the prior requests has already pulled the video content to the cache node. We use different markers to indicate different content sources for received data packets, which are, the triangle symbol denotes packets returned from cache nodes and the ’+’ symbol denotes packets returned from the producer. The transmission delay and bandwidth between node A and node B is set to 30ms and 3Mbps respectively according to Qdisc tool. The bandwidth of other links are only restricted by its NIC and are higher than the bandwidth between A and B.

As shown in Fig. 4, after the content source changes from node A to the producer, we can see the latency of DPCCP increases significantly, and then the resolution of video streams is decreased from 720p to 480p. This is because when the cached data in node A is exhausted, the bottleneck link of data flow is changed from A-consumer to A-B. However, DPCCP consumer still maintains an ”expired” interest sending rate to send requests to the network, which causes heavy backlog of data packets at new bottleneck and introduces a sudden rise in transmission latency. This excessive latency will decrease the data receive rate perceived by the video engine, since the sending rate of interests depends on RTT in DPCCP. This will in turn make the player actively reduce the resolution of video content. In contrast, since RFCC consumer maintains an appropriate interests sending rate in response to the change of content source, the latency in RFCC is stable and the resolution of video application will not change during the transmission.

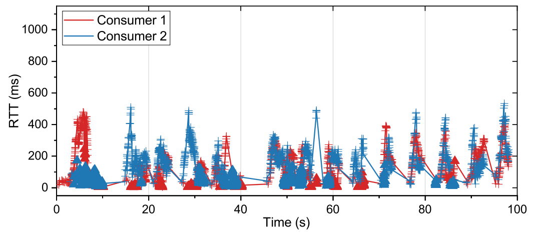
C. Performances with Dual Users

In this section, we compare the performances of RFCC and DPCCP when a new device accesses the live system. During the experiment, consumer 1 first connects to the system, and consumer 2 joins in after tens of seconds and then exits after running for 45 seconds, and consumer 1 exits at 60s. To avoid a decrease in video resolution as in Fig.4(a) when consumer 2 startup, we set the bandwidth between A-B to 4Mbps.

As shown in Fig. 5, it can be seen that in DPCCP, when the content source of consumer 2 switches from node A to the producer, the latency of returned data flow increases sharply. The reason is, when content source switches, the sending rate of interest packets does not change and becomes too high for the bottleneck. Therefore, the data packets accumulate at the bottleneck node and introduce excessive delay. This will in turn decrease the sending rate of interest packets of consumer 2 to a much lower level, and make the producer degrade the video resolution. At the same time, due to characteristic of interest aggregation [24], the interest packets sent by consumer



(a) The latency of DPCCP.



(b) The latency of RFCC.

Fig. 6. The latency of different congestion control algorithms when bottleneck bandwidth changes from 5Mbps to 3Mbps at a rate of 0.2Mbps every 10 seconds.

1 will be suspended at node A and wait until the data requested by consumer 2 arrives at node A. Since consumer 1 and 2 share the same data packets responded by the producer, their resolution will be the same and all will be degraded. In contrast, RFCC can maintain the stability of latency during transmission and ensure the QoE for all users.

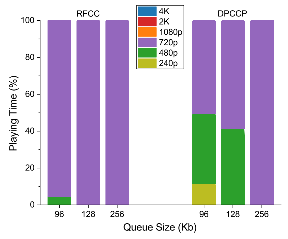
D. Performances on Time-varying Bottleneck Bandwidth

In this section, we investigate the round-trip delay when the bottleneck bandwidth is time varying. This experiment is performed in two-to-one scenario. During the experiment, consumer 1 first accesses to the system and consumer 2 joins in five seconds later, and both of them exit at 100s. When consumer 1 startup, we set the bandwidth of A-B to 5 Mbps and then decay it at a rate of 0.2 Mbps every 10 seconds.

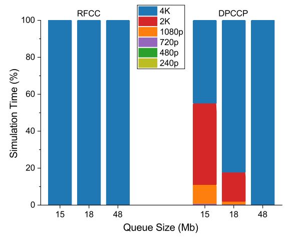
As shown in Fig. 6, it can be seen from the experimental results that as the bottleneck bandwidth gradually decreases, the latency of DPCCP becomes higher when source switching occurs while the latency of RFCC remains stable. This is because DPCCP consumers cannot perceive bandwidth changes in the bottleneck when fetching content from the caching node. Once the cache is exhausted, packets requested at the ”expired” interest sending rate will backlog at the transmission bottleneck, and this backlog will become more severe on links with low bottleneck bandwidths. In contrast, RFCC senses the change of the link bottleneck and adjusts interest sending rate with the correct value when source switching occurs, which avoids drastic jitter of the transmission delay.

E. Performances under Different Queue Buffer Sizes

In this section, we measure QoE on the end-user side in terms of the video resolution that the end-user experiences under different queue buffer sizes. Due to the performance limitations of our video server, the maximum video resolution that our real platform can support is only up to 720p. Considering that mainstream video applications can support commercial video resolutions up to 4K, we simulate 4K video playback in the NDNsim platform. We count the data throughput of the consumer and estimate playback resolution based on the bandwidth requirement of different resolutions [25]. For example, the minimum bandwidth requirement for 4K video is 16Mbps, so we consider a consumer is playing a 4K video when its throughput is higher than 16Mbps. We still perform experiments in two-to-one scenario, and set the bandwidth of A-B to 5Mbps and 20Mbps in the real and NDNsim platforms respectively. The queue size of node B is set to three different values for different bottleneck bandwidths. During the experiment, consumer 1 accesses the network first and consumer 2 accesses the network 5 seconds later, and the entire experiment lasted 200s.



(a) The video resolution on real platform when bottleneck bandwidth is 5Mbps.



(b) The video resolution on Ndnsim platform when bottleneck bandwidth is 20Mbps.

Fig. 7. Video resolution on the end-user side with different queue buffer sizes on different platforms.

As shown in Fig. 7, it can be seen that DPCCP consumers cannot maintain a stable video resolution with queue buffer sizes of 128Kb and 18Mb in bottleneck bandwidths of 5Mbps and 20Mbps, and only keep the optimal resolution 59% and 82% of the playing time, respectively. In addition, as the queue buffer size decreases, the video resolution of DPCCP consumers drops more severely. This is because DPCCP does not respond to source switching during transmission, and the bottleneck node needs large queue buffer to absorb congestion caused by over-transmitted interest packets. When the queue buffer is too small, the forwarding queue cannot store the stacked data packets, which leads to packet loss and reduces the QoE of video application. While RFCC can quickly adjust interest request rate when source switching occurs, so as to maintain stable video resolution when the queue buffer size is small.

F. Performances on Multi-path Transmission

Due to the limitation of hardware, from now on, we use NDNsim to do our tests. In this section, we analyze the performance of RFCC from the perspective of data throughput on each path in multi-path transmission mode and compare it with DPCCP and PCON. The topology of this experiment is shown in Fig. 8, where node E is the producer, nodes 1 and 2 are consumers, and nodes A to D are routers on the transmission path. Consumers are connected to the producer through two paths, ABDE and ACDE, which have bandwidths and transmission delays of 20Mbps, 40ms and 3Mbps, 70ms, respectively. We use different colored backgrounds to indicate different content sources for consumer 2, which are, the red back background denotes the producer and the blue background denotes the cache node. During the experiment, consumer 1 and 2 starts at 0s and 5s, respectively, and exits at 40s simultaneously.

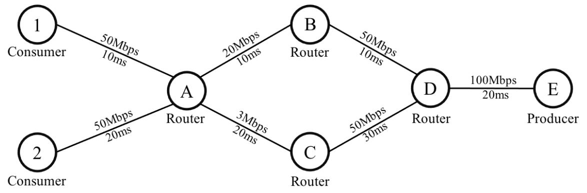


Fig. 8. Topology for multi-path transmission.

As shown in Fig. 9, the throughput of all algorithms periodically decreases to 0 on different transmission paths. This is because in HLS based video applications, the consumer who sends an interest request for a video segment temporarily stops requesting the next video segment until all the returned data is received. Before content source switching, the data requested by DPCCP and RFCC consumer through paths EDBA and EDCA can be transmitted almost simultaneously. However, in DPCCP, when source switching occurs, the transmission time of data segments on different paths appear to be significantly different because the congested path takes extra time to handle bufferbloats. The transmission time of data segment on these two paths in PCON is not affected by the switching of the content source, but its data throughput is the lowest of these three algorithm, since PCON is unable to independently control the throughput on each path. In contrast, RFCC not only guarantees the throughput of each available path, but also ensures the synchronization of data transmission on each path.

G. Performances on Different Bottleneck Bandwidth

In this section, we analyze the throughput and queuing delay for consumer 1 when source switching occurs. The experiments in this section are performed in the topology shown in Fig.8, and we set the transmission delay to 20ms for all links. The bottleneck of EDCA is C-A and the bandwidth is set to 5Mbps, and the bottleneck of EDBA is C-B and is set to three different values of 10Mbps, 15Mbps and 25Mbps. During the experiment, consumer 1 accesses the network from 0s and consumer 2 randomly accesses the network from 7-12s.

As shown in Fig. 10, RFCC has the highest throughput and its throughput averages 13.89Mbps, 18.32Mbps and 27.32Mbps at A-B bandwidths of 10Mbps, 15Mbps and 25Mbps, respectively, which are 1.08x, 1.14x and 1.16x higher than those of DPCCP. Meanwhile, the queuing delay of RFCC are 0.41x, 0.29x, 0.34x that of DPCCP at three bandwidths of A-B. This is because DPCCP cannot adjust the sending rate of interest packets accurately when the content source changes, resulting in a backlog of returned data at the bottlenecks of paths EDBA and EDCA. When the difference between the bottleneck bandwidth of these two paths is small, both paths take a long time to clear the congestion. When the difference between the bottleneck bandwidths is large, while the path EDBA can clear congestion quickly, it will remain blocked until the transmission of video segmented data on the other path is complete. The three modulation modes of PCON differ slightly in performance. Among them, PCON-CUBIC has the lowest transmission latency, but its throughput is only 41.3% of RFCC at 25Mbps bottleneck bandwidth of A-B. The reason is that PCON cannot perform congestion control on each path individually, when the link C-A is congested, PCON reduces the traffic of interest on all paths. In contrast, RFCC ensures high link utilization and stable transmission delay over a variety of links with different bandwidths.

H. Performances on Different Consumers

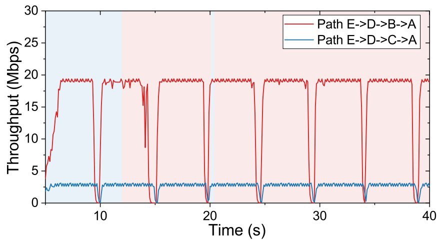
Since video servers often serve multiple users in live streaming systems. In this section, we analyze the RFCC transmission performance under different numbers of consumers. We still use the topology shown in Fig.8, setting the transmission delay to 20ms for all links, as well as setting the bandwidth between A-B and A-C to 15Mbps and 5Mbps respectively. The number of consumers connected to the network is set to three different values of 3, 5 and 7. During the experiment, the first consumer starts at 0s, and subsequent consumers start randomly 1-4s after the previous one, and the experiment lasts for 60s.

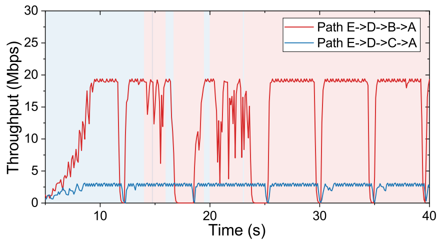
As shown in Fig. 11, the queuing delay in DPCCP increases sharply when the number of consumers becomes large, which are 1.58x, 2.17x, and 3.27x that of RFCC when the number of consumers is 3, 5 and 7, respectively. This is because DPCCP consumers fall into Cache Traps when source switching occurs, and a larger number of consumers will cause the content source to switch more frequently. The PCON algorithm for three modes maintains stable queuing delay with different consumers, but their throughput are much lower, which are 0.85x, 0.72x and 0.68x than that of RFCC respectively when 7 consumers access in. This is because PCON algorithm uses Codel to mark congestion and the consumer reduces the sending window aggressively when congestion occurs. As a result, it does not cause excessive queuing delays, but accordingly, these methods cannot achieve high link utilization. In contrast, RFCC guarantees stable transmission performance even in multi-consumer scenarios.

I. Performances under Different Topology

In this section, we use a complex diamond topology to compare performances of different algorithms. The topology of this experiment is shown in Fig. 12, where consumer 1 and consumer 2 access the network from node A and request data from producer F through four paths, which are ACEF, ADEF, ABCEF and ABDEF. We consider two different link settings, i.e., the same bandwidths and different latencies (Case I) and the same latencies and different bandwidths (Case II).

Fig. 13 shows the data receive rate of these algorithms when source switching happens. In Case I, the data download rate of


(a) Two RFCC data flows at router A.


(b) Two DPCCP data flows at router A.

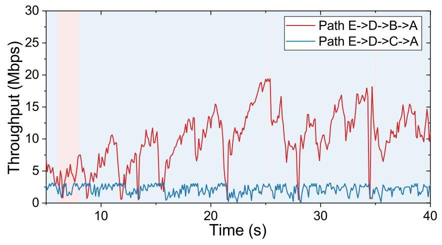
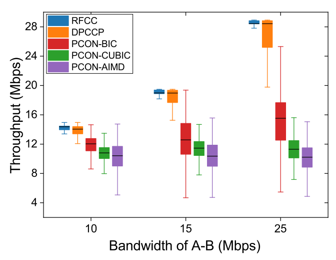

(c) Two PCON data flows at router A.

Fig. 9. Throughput of different congestion control algorithms when the content source of returned packets changes from cache node (blue background) to producer (red background).


(a) Throughput.

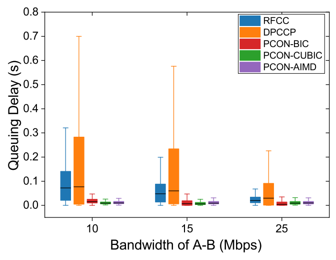
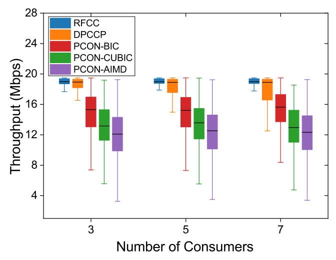

(b) Queuing delay.

Fig. 10. Throughput and queuing delay of different congestion control algorithms with different bottleneck bandwidth.


(a) Throughput.

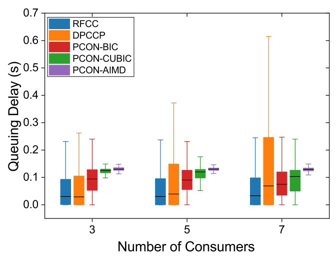

(b) Queuing delay.

Fig. 11. Throughput and queuing delay of different congestion control algorithms with different consumers.

RFCC is 30.99Mbps, that of DPCCP is 25.13Mbps, and that of three modes of PCON-BIC, PCON-CUBIC and PCON-AIMD are 23.43Mbps, 23.71Mbps and 21.15Mbps, respectively. For DPCCP, as the difference between different transmission delay is relatively large, when source switching occurs, the path with the larger delay will experience more serious congestion, which results in a degradation of transmission performance. For PCON, even though the difference in bottleneck bandwidth between transmission paths is small, its aggressive window reduction strategy results in an excessive drop in data throughput each time congestion occurs. In Case II, the data download rate of RFCC is 27.98Mbps, that of DPCCP is 21.58Mbps, that of PCON-BIC, PCON-CUBIC, and PCON-AIMD is 17.31Mbps, 12.42Mbps and 12.67Mbps, respectively. For PCON, since consumers cannot distinguish between data streams, the sending rate is uniformly adjusted according to congestion signals in concurrent streams, and high bandwidth links are often underutilised when low bandwidth paths exist in the network. For DPCCP, due to bandwidth differences, Cache Traps occurring on low-bandwidth links result in more severe congestion, and consumers need to wait for all content on the congested path to be transmitted before they start requesting the next data segment, which reduces link utilization for other paths. In contrast, RFCC adjusts the interest request rate of each path to the appropriate value in time when source switching occurs, which ensures the link utilization of each transmission path.

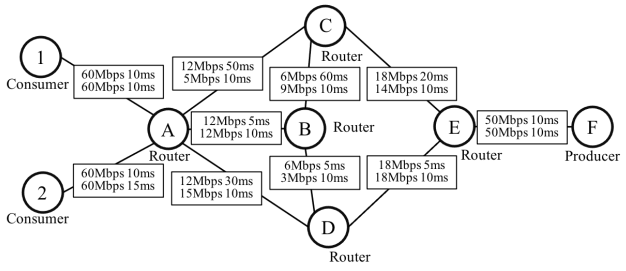


Fig. 12. Diamond-structured topology for multi-path transmission.

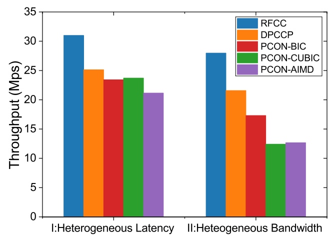
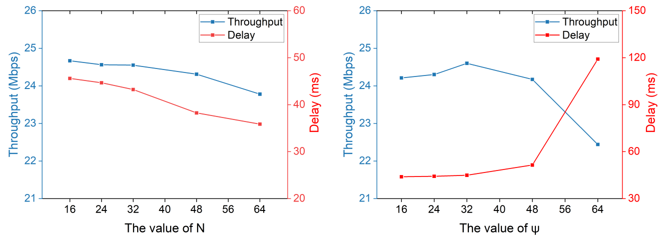


Fig. 13. Throughput of different congestion control algorithms under diamond-structured topology with different link settings.

J. Performances with Different Parameter Settings

In the design of RFCC, we define two parameters, namely, the sliding window size N used to estimate data receive rate, and the sliding window size ψ used to perceive content source switching. We set the bottleneck bandwidth of link A-B in Fig. 8 to 25Mbps and repeat the test ten times for all parameters, and take the average throughput and delay during receive current data segment at consumer 1 when source switching occurs.



(a) The impact of N on performance when ψ is set to 32.

(b) The impact of ψ on performance when N is set to 48.

Fig. 14. RFCC performance under different parameter settings.

In this experiment, we first set the value ψ to 32 and then test the performance of RFCC when N is set to 16, 24, 32, 48 and 64 respectively. The result shows in Fig. 14(a). As the value N increase, the throughput and queuing delay decrease. This is because when the value of N becomes large, the throughput estimated by the consumer and cache nodes cannot effectively the fluctuation of throughput, leading to low estimates in unstable data streams. Similarly, when we set N to 48 and ψ to 16, 24, 32, 48 and 64. As ψ increases, the queuing delay increase and the throughput firstly increase but decrease when the value exceed 32. This is because a larger value of ψ means that it takes more time for the consumer to detect the switching of content source, which leads to the consumer not being able to adjust requesting rate in time and thus falling into the Cache Trap. On the contrary, when the value of ψ is small, the consumer prematurely reduces the requesting rate, resulting in a reduction of link utilization. Referring to QCC [26], we set the parameter selection criteria as the value that maximizes the ratio of throughput to delay, and our experimental results show that the values of 48 for N and 32 for ψ meet this condition.

VI. Discussions and Future Works

Adaptive Parameter Tuning: Parameters in RFCC (i.e., α, β, ψ, K, N) are either in line with other algorithms (i.e., BBR [15], DPCCP [12]), or determined by experimental tests. Considering varying network conditions, we plan to enable these parameters to be adaptively tuned to optimal values, which may be achieved by analytical or heuristic ways. For example, the size of sliding windows can be set smaller when network flows are stable, and vice versa.

More Extensive Evaluations: Considering the dynamic nature of live streaming and the diversity of network streams, we plan to conduct more tests to validate the performance of RFCC under more realistic conditions. We will also do extension evaluations on other network metrics beyond QoE such as packet loss or energy efficiency.

VII. Related Works

NDN congestion control algorithms can be classified as Receiver-based and Hop-by-Hop adaptive forwarding schemes. Early researches in Receiver-based algorithm mainly focused on how to enable consumers to adjust the interest sending rate according to the congestion signals from different content sources [27] [28] [29] [30] [31]. However, these approaches can not be generally applied in multi-source multi-path scenarios. MIRCC [9] is a rate-based method that allows consumer to control the flow of each stream independently. Compared to RFCC, the core limitations of MIRCC include [13]: 1) each router needs to calculate the optimal rate for each flow, which is not scalable 2) it requires accessing underlying link status (queue), which can be infeasible if NDN is an overlay and 3) its RCP-based congestion control cannot support in-network cache.

The path-specific mechanism proposed [11] [12] [32] recent years provides a new solution to the multi-source problem. It allows the consumer to identify all available transmission paths and control the sending rate of interest packets on each transmission path independently, so that if one path is congested, the interest sending rates of other paths can still be increased. However, these methods perceive congestion based on end-side statistics and cannot adapt to dynamic changes of content sources in the transmission link, leading to packet loss or bufferbloat when Cache Trap happens.

Hop-by-Hop adaptive forwarding schemes [33] [34] [35] are mainly implemented at NDN routers, and prevent congestion by limiting the forwarding rate of interest packets or forwarding them to other available path. These approaches can perceive congestion quickly at router. However, routers cannot get information from the entire network, making it difficult to make correct forwarding decisions. Besides, these works also do not consider the possible congestion due to the change of content sources.

VIII. Conclusion

In this work, we carefully analyze and validate the reasons for QoE degradation in NDN live video streaming. When content source switches, consumers can not adjust the sending rate of interest packets in time since they can not perceive the bottleneck bandwidth of the transmission path. To solve this problem, we present RFCC, which aims at optimizing the sending rate of interests in NDN fows by explicitly feedback the bottleneck data rate in data packets. Our experimental results on the real-world and simulation platforms show that RFCC ensures the consumer's QoE in real-time video services, reduces 50% delay jitters compared with DPCCP and achieves 2.4× throughput compared with PCON.

Acknowledgment

This work was supported by National Key Research and Deployment Program of China (Grant No.2020YFB1806402), Major Key Project of PCL (Grant No.PCL2023A06) and Fundamental Research Funds for the Central Universities E3E41902 and E3ET1901X2. We sincerely thank our shepherd Toru Hasegawa and anonymous reviewers for their valuable comments and suggestions.

References

[1] Video streaming (svod) - worldwide. [Online]. Available: https://www.statista.com/outlook/dmo/digital-media/video-on-demand/video-streaming-svod/worldwide

[2] T. Liang, Y. Zhang, B. Zhang, W. Zhang, and Y. Zhang, "Low latency internet livestreaming in named data networking," in *Proceedings of the 9th ACM Conference on Information-Centric Networking*, 2022, pp. 177–179.

[3] C. Ghasemi, H. Yousefi, and B. Zhang, "Far cry: Will cdns hear ndn's call?" in *Proceedings of the 7th ACM Conference on Information-Centric Networking*, 2020, pp. 89–98.

[4] ——, "icdn: An ndn-based cdn," in *Proceedings of the 7th ACM Conference on Information-Centric Networking*, 2020, pp. 99–105.

[5] P. Gusev and J. Burke, "Ndn-rtc: Real-time videoconferencing over named data networking," in *Proceedings of the 2nd ACM Conference on Information-Centric Networking*, 2015, pp. 117–126.

[6] T. Li, T. Song, and Y. Yang, "{iStack}: A general and stateful name-based protocol stack for named data networking," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 267–280.

[7] D. Saxena, V. Raychoudhury, N. Suri, C. Becker, and J. Cao, "Named data networking: a survey," *Computer Science Review*, vol. 19, pp. 15–55, 2016.

[8] R. Pantos and W. May, "Http live streaming," Tech. Rep., 2017.

[9] M. Mahdian, S. Arianfar, J. Gibson, and D. Oran, "Mircc: Multipath-aware icn rate-based congestion control," in *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, 2016, pp. 1–10.

[10] K. Schneider, C. Yi, B. Zhang, and L. Zhang, "A practical congestion control scheme for named data networking," in *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, 2016, pp. 21–30.

[11] Y. Ye, B. Lee, R. Flynn, N. Murray, G. Fang, J. Cao, and Y. Qiao, "Ptp: Path-specified transport protocol for concurrent multipath transmission in named data networks," *Computer Networks*, vol. 144, pp. 280–296, 2018.

[12] Y. Ye, B. Lee, R. Flynn, J. Xu, G. Fang, and Y. Qiao, "Delay-based network utility maximization modelling for congestion control in named data networking," *IEEE/ACM Transactions on Networking*, vol. 29, no. 5, pp. 2184–2197, 2021.

[13] G. S. Koch and R. F. Link, "The coefficient of variation; a guide to the sampling of ore deposits," *Economic Geology*, vol. 66, no. 2, pp. 293–301, 1971.

[14] J. Shi and B. Zhang, "Ndnlp: A link protocol for ndn," *NDN, NDN Technical Report NDN-0006*, 2012.

[15] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[16] A. Afanasyev, I. Moiseenko, L. Zhang *et al.*, "ndnsim: Ndn simulator for ns-3," 2012.

[17] Shaka-player. [Online]. Available: https://github.com/shaka-project/shaka-player

[18] A. Afanasyev, J. Shi, B. Zhang, L. Zhang, I. Moiseenko, Y. Yu, W. Shang, Y. Huang, J. P. Abraham, S. DiBenedetto *et al.*, "Nfd developer's guide," *Dept. Comput. Sci., Univ. California, Los Angeles, Los Angeles, CA, USA, Tech. Rep. NDN-0021*, vol. 29, p. 31, 2014.

[19] P. Gusev, Z. Wang, J. Burke, L. Zhang, T. Yoneda, R. Ohnishi, and E. Muramoto, "Real-time streaming data delivery over named data networking," *IEICE Transactions on Communications*, vol. 99, no. 5, pp. 974–991, 2016.

[20] K. Nichols and V. Jacobson, "Controlling queue delay," *Communications of the ACM*, vol. 55, no. 7, pp. 42–50, 2012.

[21] Y. R. Yang and S. S. Lam, "General aimd congestion control," in *Proceedings 2000 International Conference on Network Protocols*. IEEE, 2000, pp. 187–198.

[22] K. P. Burnham and D. R. Anderson, "Multimodel inference: understanding aic and bic in model selection," *Sociological methods & research*, vol. 33, no. 2, pp. 261–304, 2004.

[23] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.

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

[25] What is a good upload speed for streaming. [Online]. Available: https://restream.io/blog/what-is-a-good-upload-speed-for-streaming/

[26] L. Li, Z. Li, Y. Chen, J. Zhang, and L. Li, "Upload your data faster: Driver-queue based congestion control for wireless networks," in *2022 IEEE 30th International Conference on Network Protocols (ICNP)*. IEEE, 2022, pp. 1–11.

[27] G. Carofiglio, M. Gallo, and L. Muscariello, "Icp: Design and evaluation of an interest control protocol for content-centric networking," in *2012 Proceedings IEEE INFOCOM Workshops*. IEEE, 2012, pp. 304–309.

[28] L. Saino, C. Cocora, and G. Pavlou, "Cctcp: A scalable receiver-driven congestion control protocol for content centric networking," in *2013 IEEE international conference on communications (ICC)*. IEEE, 2013, pp. 3775–3780.

[29] G. Carofiglio, M. Gallo, and L. Muscariello, "Optimal multipath congestion control and request forwarding in information-centric networks: Protocol design and experimentation," *Computer Networks*, vol. 110, pp. 104–117, 2016.

[30] F. Zhang, Y. Zhang, A. Reznik, H. Liu, C. Qian, and C. Xu, "A transport protocol for content-centric networking with explicit congestion control," in *2014 23rd international conference on computer communication and networks (ICCCN)*. IEEE, 2014, pp. 1–8.

[31] Z. Li, X. Shen, H. Xun, Y. Miao, W. Zhang, P. Luo, and K. Liu, "Coopcon: Cooperative hybrid congestion control scheme for named data networking," *IEEE Transactions on Network and Service Management*, 2023.

[32] F. Wu, W. Yang, M. Sun, J. Ren, and F. Lyu, "Multi-path selection and congestion control for ndn: An online learning approach," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1977–1989, 2020.

[33] Y. Wang, N. Rozhnova, A. Narayanan, D. Oran, and I. Rhee, "An improved hop-by-hop interest shaper for congestion control in named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 55–60, 2013.

[34] N. Rozhnova and S. Fdida, "An effective hop-by-hop interest shaping mechanism for ccn communications," in *2012 Proceedings IEEE INFOCOM Workshops*. IEEE, 2012, pp. 322–327.

[35] S. Song and L. Zhang, "Effective ndn congestion control based on queue size feedback," in *Proceedings of the 9th ACM Conference on Information-Centric Networking*, 2022, pp. 11–21.