International Journal of Innovative Computing, Information and Control
Volume **17**, Number **3**, June **2021**

ICIC International ©2021 ISSN 1349-4198
pp. **853–871**

TOWARD INFORMATION-CENTRIC NETWORKING RECEIVER-DRIVEN TRANSMISSION MECHANISM OVER WIRELESS LOCAL AREA NETWORK: IMPLEMENTATION AND OPTIMIZATION

Yifeng Liu[1,2], Xuewen Zeng[1,2], Rui Han[1,2] and Peng Sun[1,2]

[1]National Network New Media Engineering Research Center
Institute of Acoustics, Chinese Academy of Sciences
No. 21, North 4th Ring Road, Haidian District, Beijing 100190, P. R. China
{ liuyf; zengxw; hanr; wangx }@dsp.ac.cn

[2]School of Electronic, Electrical and Communication Engineering
University of Chinese Academy of Sciences
No. 19(A), Yuquan Road, Shijingshan District, Beijing 100049, P. R. China

Received December 2020; revised April 2021

Abstract. *Information-Centric Networking (ICN) is a novel network paradigm that is compatible with the pull-based content delivery approach. The pull-based and connectionless nature of ICN promotes the exploitation of receiver-driven transport protocols. In deployment, using Wireless Local Area Networks (WLANs) to provide users low-cost, easy-to-deploy and high-quality wireless accesses to ICN has practical significance. And from the view of transmission, the Quality-of-Experience (QoE) of mobile applications in ICN can be significantly enhanced through the exploitation of wireless networks. In this paper, we introduce a natively supported receiver-driven transmission mechanism for ICN and focus on the wireless access scenario. We have implemented it in the Linux kernel. First, we present the core functionalities of the protocol, including a receiver-based Bottleneck Bandwidth and Round-trip propagation time (BBR) algorithm. Then, we explore the major issues raised by our receiver-driven transmission in IEEE 802.11-based networks. Through analysis of the transport model, we propose corresponding schemes to balance communication overheads and control precision while achieving saturated throughput. Finally, we experimentally validate through extensive experiments on a physical testbed that the introduced design is robust, offers high performance and induces less overhead on Content Routers (CRs).*

Keywords: Information-centric networking, Receiver-driven transmission control, WLAN, Receiver-based BBR, Optimization

1. **Introduction.** Over the past decades, the pressure of massive content delivery has promoted the rethinking of Internet communication paradigm. Today's CDN and P2P system make efforts in this aspect. However, the rapid growth of traffic data still reveals shortcomings of their architecture on mobility support, content delivery efficiency, scalability, etc.

Information-Centric Networking (ICN) [1] is expected as an innovative network paradigm. And it is expected to have potential to enhance Fifth-Generation (5G) network on transmission latency and data distribution efficiency. ICN advocates the decoupling of the identifier and locator, which shifts the communication from host-centric to information-centric. ICN is characterized by ubiquitous in-network caching, mobility support, build-in multicast and inherent security. So far, many ICN architectures have been designed, such

DOI: 10.24507/ijicic.17.03.853

as Named Data Networking (NDN) [2], MobilityFirst (MF) [3], DONA [4], and Network of Information (NetInf) [5]. In contrast with those clean-state (e.g., NDN) that abandon the IP layer, ICN over IP [6,7] (e.g., MF) greatly reduces deployment costs due to its compatibility with current IP facilities. This incrementally deployed ICN has practical significance that the compatibility contributes to the smooth evolution of ICN. In this paper, our work is based on such ICN.

In ICN, data is treated as information objects. An information object is commonly known as a Named Data Object (NDO) – an addressable data chunk representing a piece of information. Each NDO has a globally unique identifier. NDO retrieval in ICN is natively pull-based driven by user requests. Through Name Resolution System (NRS), user gets the Network Address (NA) of NDO and sends requests (or interests) to this NA. And then, the Content Router (CR) returns the corresponding NDO to the user. Defining an overlay (e.g., HTTP) on the top of TCP can realize such request-response (or interests/data) mode. However, considering the heavy tasks of CR (high throughput forwarding, traffic monitoring, cache and storage [8,9]), its "sender-based" nature of transmission control puts massive pressure on CRs. By comparison, the native receiver-driven transmission offloads the state management and transmission control to all users and makes CRs stateless, which is more in agreement with the "connection-less" and "pull-based" nature of ICN. Moreover, this way greatly reduces the complexity of CRs and improves the robustness and performance of CRs.

As one of the most widely used wireless access technologies, Wireless Local Area Network (WLAN) [10] provides low-cost, easy-to-deploy and high-quality Internet access services. With the rapid increase of number of mobile users, users accessing to ICN system through the IEEE 802.11-based (Wi-Fi) networks is common. From the perspective of transmission, the wireless network, as the last-hop adjacent to the users, is usually the bottleneck of the whole transmission path. And the QoE (Quality-of-Experience) of mobile applications in ICN is critically subject to the transmission behavior on such medium. In fact, the receiver-driven way can bring many gains through exploiting information available only at the receiver end [11]. For instance, transmission can faster recover from loss state for that the receiver directly accessing to the receive buffer has a better knowledge of receiving situations without incurring the overhead of feedbacks (e.g., SACK in TCP). Moreover, since the user is adjacent to the wireless last-hop, it has the first-hand knowledge about wireless information, such as Received Signal Strength Indication (RSSI). This may be useful for optimizing transmission control, for example, when handoff occurs, the user can fast detect this non-congestion related outage and recover the transmission soon on the new path. On the other hand, due to Wi-Fi's characters of high degrees of aggregation (A-MPDU and A-MSDU in IEEE 802.11n/ac) and RTT variance, optimizing transmission on such medium has practical significance for deploying a high-available ICN system.

In this paper, we introduce a receiver-driven transmission mechanism with BBR and put it into the ICN context to provide pull-based, reliable, ordered delivery of streams of octets between users and CRs. Particularly, we focus on the performance in the wireless access scenario. We have implemented the mechanism in the Linux kernel (4.19.97). A demo with resource code is available on [12]. Through POSIX APIs, user-space programs can request their desired NDOs. The receiver-driven transmission does not require connection setups in agreement with "connection-less" nature of ICN and makes CRs stateless by offloading transmission control to all receivers.

The contribution of this work can be summarized as

- We show the receiver-driven transmission stack implemented in the kernel space and describe the key functionalities, especially a receiver-based BBR. To the best of our knowledge, we are the first to implement the BBR at the receiver end and evaluate its performance in practical uses.
- We focus on major issues raised by receiver-driven way in the context of Wi-Fi network. Through analysis of the transport model, we present requests aggregation and adjust the congestion window (cwnd) to achieve the saturated throughput.
- We conduct extensive experiments on a physical testbed to evaluate the performance of receiver-driven transmission under various link conditions, especially in Wi-Fi scenarios. The experimental results show that our receiver-driven transmission has high performance while consuming less resources on CRs.

The remainder of the paper is organized as follows. In Section 2, we review the related work about receiver-driven transmissions and the BBR algorithm. Section 3 presents the whole receiver-driven transmission system implemented in the kernel space. In Section 4, we explore major issues in Wi-Fi environment and propose corresponding schemes to optimize it. Then, we evaluate performance and analyze it in Section 5. Finally, we conclude this paper in Section 6.

2. **Related Work.**

Two key research fields related to our work: receiver-driven transmission and congestion control. This section presents a comprehensive study of the current literature on them.

2.1. **Receiver-driven transmission.**

There is some literature on receiver-driven transmission. NETBLT (Network Block Transfer) [13] may be one of the first transport protocols that delegate some transport functionalities to the receiver. It places retransmission timer at the receiver end to make error recovery more efficiently. In its later work, more control functionalities are delegated to the receiver for better performance. WTCP (Wireless Transmission Control Protocol) [14] delegates sending rate calculation to the receiver end. In TFRC (TCP-Friendly Rate Control) [15], receiver records the history of loss state and reports it to the sender. The latter then calculates the TCP-friendly sending rate according to feedbacks. TCP-Real [16] is a receiver-oriented TCP-compatible and -friendly protocol with aim to improve real-time capabilities of TCP. It tracks packet loss and determines the sending rates at the receiver. [17] and [18] delegate the receiver to control the bandwidth shares of TCP flows by adapting the receiver's advertised window and delay in returning ACK messages.

In contrast with above protocols that just place part of transmission control functionalities at the receiver, WebTP (Web Transport Protocol) [19] and Reception Control Protocol (RCP) [20] offload all control functions to the receiver. WebTP presents a receiver-oriented, request/response protocol for the Web. It is designed to be completely receiver-based in terms of transport initiation, flow control and congestion control. RCP is a receiver-centric transport protocol over wireless environments, especially in the context where mobile hosts equip with multiple interfaces. Authors argue that RCP allows for better congestion control, loss recovery and power management mechanisms compared to sender-centric ways. From the perspective of transport, [11] summarizes the key advantages and vulnerabilities of receiver-driven TCP. It concerns that transmission controlled by receivers introduces an incentive for misbehavior that receivers can illegally manipulate the congestion control to obtain higher throughput or lower latency. The paper argues that transmission should strike a balance between enforcement mechanisms and complete

trust of endpoints. However, despite this, receiver-driven way brings some gains in performance and functionality, such as faster recovery from loss state and first-hand knowledge about wireless link, which could help with transmission control on wireless paths. On the other hand, receiver-driven transmission can reduce the complexity of servers because it distributes the state management and control across the large number of clients.

In ICN domain, Carofiglio et al. propose the Interest Control Protocol (ICP) as a receiver-driven transport protocol for Content-Centric Network (CCN) [21]. It adopts a window-based interest flow control mechanism and depends on delay measurements and timer expirations to ensure reliability. Afterwards, authors propose the Remote Adaptive Active Queue Management (RAAQM) [22] applied in CCN. It focuses on multipath transmission and realizes separate RTT monitoring on each route. RAAQM adjusts congestion window of each flow to achieve efficient and fair resource utilization. NetInf TP [23] develops a receiver-driven transport protocol which provides reliable data transfer with TCP-like congestion control and a new type of retransmission. Simulation results show that NetInf TP slightly outperforms TCP New-Reno.

2.2. **Congestion control and BBR.** Congestion control is key to transmission. To the best of our knowledge, all of above receiver-driven protocols adopt a loss-based congestion control which adopts the Additive Increase Multiplicative Decrease (AIMD) model. Loss-based congestion control (e.g., CUBIC [24]) leads to "bufferbloat" issue [25] when bottleneck buffers are large and leads to low throughput when bottleneck buffers are small. In addition, window-based rate control, which just uses congestion window (cwnd) to bound the amount of data inflight (data sent but not yet acknowledged), incurs bursty traffic, ACK compression and multiplexing problems [26].

To deal with these, Google proposes a new congestion control algorithm, named BBR [27]. With the pacing which uses the window to determine how much to send out but uses rates instead of acknowledgements to determine when to send, BBR bounds the amount of data inflight near the estimated Bandwidth-Delay Product (BDP) and keeps rates equal to the estimated bottleneck bandwidth. To achieve this, BBR dynamically detects the maximum bandwidth and the minimum Round-Trip Time (RTT) by ProbeBW and ProbeRTT state. The ProbeBW is the cycle, wherein the pacing_gain periodically changes in $\{1.25, 0.75, 1, 1, 1, 1, 1, 1\}$. The phase of 1.25 means packets more quickly entering into the network for bandwidth detection. On the other hand, BBR periodically turns to ProbeRTT state to detect the minimum RTT. In this way, BBR can generate a smaller intermediate queue which introduces lower delays while achieving a full bandwidth utilization. Now, BBR is available on many Linux's distributions and many protocols support it, e.g., QUIC [28]. And as an emerging algorithm, BBR is still under standardization. [26] and the series of IETF documents [29-33] present the state machine model (i.e., Startup, Drain, ProbeBW and ProbeRTT state) and details of it. In the later work [34-36], many improvements are proposed, namely BBR v2.0 which has a state machine similar to v1 at high level but many details are changed, such as adjustment of control parameters (pacing_gain, cwnd_gain), and fairness.

BBR over wireless environments has aroused extensive attention. In [37], the authors evaluate the performance of BBR in the cellular networks. And in [36], the authors trace the packets over a Wi-Fi path where the TCP sender is on Ethernet and the receiver is on a Wi-Fi network. Due to Wi-Fi's aggregation and RTT variance, it introduces an aggregation estimator, which allows the sender to put extra data inflight, to achieve saturated throughput over Wi-Fi. [38,39] argue that TCP BBR is inefficient in exploring the Wi-Fi bandwidth due to the contradiction between TCP BBR's queue control and Wi-Fi's aggregation function. It reveals the impact of TCP TSQ and BBR on Wi-Fi uplink

and downlink throughput. The author increases the value of 1.25 for the pacing_gain in the cycle $\{1.25, 0.75, 1, 1, 1, 1, 1, 1\}$ to allow a longer queue for higher throughput, at the cost of slight increased delay.

All in all, from the view of protocol, most of receiver-driven mechanisms (e.g., WTCP, TFRC, TCP-Real) mentioned above all make incremental improvements based on TCP. However, TCP's connection-oriented mode and *data-ack* style of message exchange are in contradiction with pull-based, connection-less nature of ICN. From the view of transmission control, RCP, RAAQM and NetInf TP use *requst-data* exchange for data transfer, but they adopt AIMD or AIMD-like congestion control which leads to a high propagation delay. BBR is a good choice. However, receiver-based BBR still lacks on the literature. And BBR optimization over wireless network is based on TCP. We overcome the limitation of these schemes, place the BBR congestion control in receiver-driven transmission and improve its performance over WLANs.

3. System Design and Implementation.

The whole receiver-driven transmission system is described in this section. We first give the design of protocol. Then, we describe the protocol stack architecture implemented in the kernel space. And finally, we detail the key functionalities for receiver-driven transmission, including timeout request retransmission, fast request retransmission and receiver-based congestion control (BBR).

3.1. Protocol description.

To incrementally deploy the ICN, we place an EID layer on the IP layer to explicitly announce the EID of the content requested by the receiver in both request and data directions. Here, the EID is a globally unique identifier of the content. For executing ICN's cache policy, the EID layer is visible to the intermediate CRs. A receiver-driven transport layer is upon the EID layer. In receiver-driven way, data acknowledge the requests sent by the receiver instead of receiver acknowledging received data by ACK messages. Therefore, to provide pull-based, reliable, ordered delivery of streams of octets, the layer needs two essential fields, namely offset and length. The 2-tuple <offset, length> controlled by the receiver represents which data should be sent and how much data can be sent. Here, the two fields are both 32-bit. Besides, we set another 1-bit field app_limit which is to explicitly notify status of sender's application layer. For those end-to-end streaming media transmission, congestion control at the receiver end needs such a field to distinguish between different reasons of bandwidth decreasing – congestion or limitation of sender's application. However, in ICN, this scenario is not common due to ICN's chunk-based storage. On the other hand, for higher protocol efficiency, we allow the receiver to carry, if necessary, more control information in the option field of receiver-driven transport protocol. As usual, the item of option field is based on Tag-Length-Value (TLV) structure. The option filed enables the receiver to request the discrete byte fragments using one request. This is useful for recovery from the loss state in an effective fashion.

3.2. Protocol stack architecture.

The whole protocol stack has been implemented in the Linux kernel space, which is shown in Figure 1. It includes not only a receiver-based congestion control but all essential functionalities for transmission such as reliability guarantees, and reordering. The receiver-driven mechanism is integrated into a kernel module working on top of the receiver's IP layer. And we provide a set of POSIX APIs (e.g., socket, connect, sendto, recvfrom, and close) for user-space programs to obtain data from ICN. Specially, the semantics and functions of these APIs are reconstructed. For example, the "connect" has not the semantics of TCP's triple handshake. Instead, it represents that user asks for some content. Thus, in addition to the NA (IP address) of the CR, we allow the user-space program to pass the EID of desired content, which is

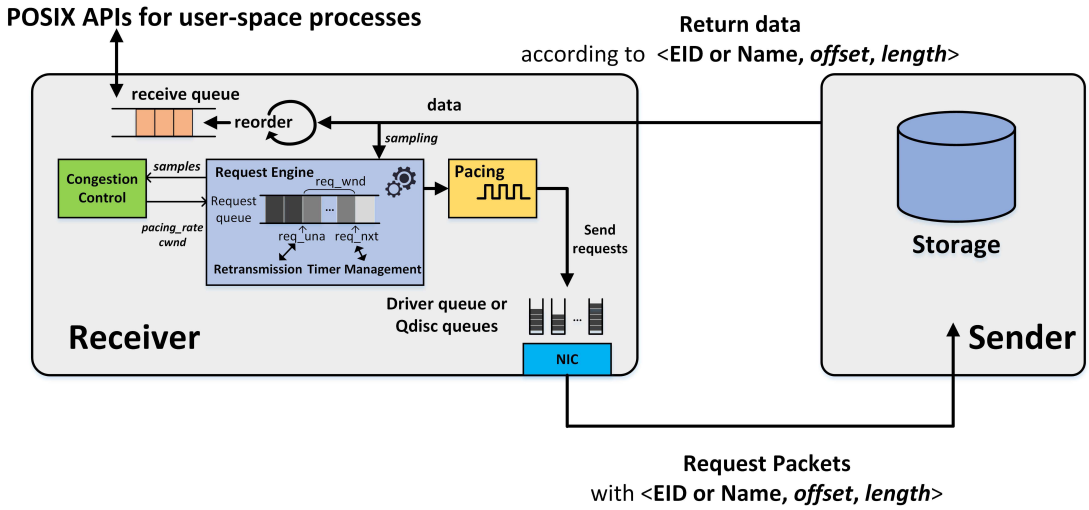


FIGURE 1. Protocol stack architecture in the kernel space

used to fill the EID layer of requests. The functionality of sender is simple that it just sends byte fragments according to the information – EID, offset and length – carried in the requests.

At the high level, our implementation has a framework similar to Linux TCP. We also utilize a two-layer lock and a backlog queue to synchronize information while minimizing contention for resource.

As shown in Figure 1, request engine and congestion control drive the pull-based transmission. It is still based on the sliding window mechanism, which has proven to be effective in dealing with transmission complexity. Those between *req_una* and *req_nxt* represent the amount of data inflight (data that has been requested but not yet received). In order to sampling, each item in the sliding window records the information of the request when the request sent, such as current timestamp, the number of delivered packets, send time of the most recently sent request, and timestamp of the most recently received packet. For each data packet arriving at the receiver, we can not only estimate the Round-Trip Time (RTT) and Retransmission Time-Out (RTO), but estimate the delivery rate which is important for BBR. An entire and robust method of delivery rate estimation is elaborated in [40]. The sampling results (delivery rate and RTT) will be passed to congestion control (BBR) module which will return two parameters (pacing_rate and cwnd) to control data requesting in turn.

We utilize an out-of-order (ofo) queue to reorder those out-of-order packets. The ofo queue is a red-black tree, which can search, insert or delete one packet within $O(\log n)$. Packets in the ofo queue are queued by byte offset in ascending order. When the offset of the incoming packet is not equal to req_una, we enqueue the packet into the ofo queue. And when the offset of the first packet in ofo queue equals *req_una*, we add consecutive packets starting with the head of ofo queue into the receiver queue (Figure 1) for accessing by program. Like TCP, reliability is guaranteed by timeout request retransmission and fast request retransmission.

3.3. **Timeout request retransmission.** Timeout request retransmission is triggered by timer expiration. And timeout setting relies on RTT and RTO estimate. We adopt a similar smooth RTT/RTO estimate method referring to RFC 6298 [41]. Further, we make some changes in timeout timer management for receiver-driven transmission. The rules of timer management are as follows:

- Every time a request is sent, if the timer is not running, the receiver starts it so that it expires after RTO. And when all requested data have arrived at receiver (*req_wnd* equals zero), turn off the timer.
- Restart the timer with current estimated value of RTO only if a data packet arrives in order (the offset field of the packet just equals *req_una*).
- When the timer expires, we re-request those segments that have been requested but are thought to be lost.

Unlike TCP just retransmitting the earliest segment that has not been acknowledgement, receiver re-requests all segments that are thought to be lost when the timer expires. This is because that receiver has direct access to the receiver queue and request queue to get more information about packet loss. In addition to the earliest requested segment that is not received, we will re-request all of those segments that have been requested but do not arrive at receiver (i.e., between *req_una* and *req_nxt*) within the estimated RTO.

3.4. **Fast request retransmission.** Fast request retransmission can detect packet loss earlier than timeout request retransmission so that transmission can recover from loss state faster. Fast request retransmission depends on statistics of the number of arriving out-of-order segments. In TCP, generally, once sender receives 3 duplicate ACKs or Selective ACKs (SACK), it thinks the packets are lost. The value of 3 is to tolerate for slight out-of-order phenomenon in data direction. Out-of-order in the feedback direction does not matter because of ACK's semantics of cumulative acknowledgement. However, this may be different in receiver-driven transmission for that out-of-order in both directions (request direction and data direction) can affect data delivery. Therefore, we increase this value to 6. Once the receiver observes 6 out-of-order packets, it immediately retransmits a request again to ask for that packet not yet received.

3.5. **Receiver-based congestion control.** We transplant the entire code of Linux (4.14.97) TCP BBR to the receiver end. In addition, a pacing function is used by receiver-based BBR to pace the requests smoothly. BBR is a model-based congestion control which follows a state machine including four states: Startup, Drain, ProbeBW and ProbeRTT. Detecting the bottleneck bandwidth (BtlBw) and $\mathrm{RTT}_{\mathrm{min}}$, BBR bounds the amount of data inflight near the estimated BDP and keeps the send rate equal to the estimated BtlBw. This brings a smaller intermediate queue (i.e., lower delays) while achieving a high throughput.

Figure 2 shows an overview of the receiver-based BBR. Receiver-based BBR still utilizes two control parameters to control the transmission, namely pacing_gain and cwnd_gain.

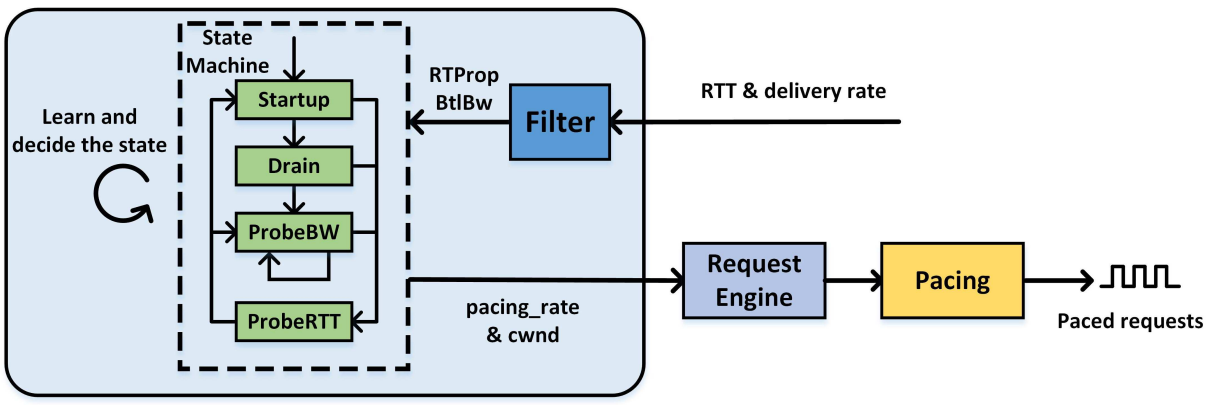


FIGURE 2. BBR and pacing at the receiver end

Learning from link conditions, receiver-based BBR determines the values of control parameters based on the state machine and finally returns pacing_rate and cwnd which control the data requesting in turn. The pacing_rate directly controls the request rate to indirectly control the sending rate of sender to be equal to BtlBw. Cwnd is the upper limit of the amount of data inflight. The outputs – pacing_rate and cwnd – are still yielded by following formulas, respectively:

$$\text{pacing\_rate} = \text{pacing\_gain} \times \text{BtlBw} \tag{1}$$

$$\text{cwnd} = \text{cwnd\_gain} \times \text{BtlBw} \times \text{RTT}_{\min} \tag{2}$$

Though pacing can be realized by Fair Queue of Qdisc, we implement an internal pacing like Linux TCP does for accuracy. The pacing function sets one high resolution timer (hrtimer) per socket and one tasklet per CPU which is a micro thread in the Linux kernel. Suppose that the receiver sends a request asking for M bytes data. It will start the hrtimer so that it expires after $\frac{\text{M}}{\text{pacing\_rate}}$. And before sending the next request, the receiver will check whether the hrtimer is enabled. If true, the request will not be sent. And when the hrtimer expires, it will delegate the task of sending request to the tasklet of current CPU to trying to send the next request at the appropriate time. In this way, the pacing function paces the requests instead of pouring them down.

4. **Optimization over Wireless Network.** In this section, we first give the transport model of receiver-based BRR and analyze the characteristic of wireless media. Based on these, we propose the request aggregation to improve transmission efficiency and bandwidth detection capability. Then, we adjust the cwnd to solve the cwnd exhaustion issue.

4.1. **Transport model.** Similar to MSS in TCP, we set up a minimum transmission unit that each data frame returned by CRs carries a payload of UNIT_MSS bytes. And each request must ask for a multiple of UNIT_MSS (i.e., the *length* field of request is $n \times$ UNIT_MSS, and the n is an integer).

For receiver-based BBR, the key is to estimate the bottleneck bandwidth $\widehat{\text{BtlBw}}$ and $\widehat{\text{RTT}_{\min}}$ by sampling. For an NDO which has $m \times$ UNIT_MSS ($m \in \text{N+}$) bytes, it has m minimum transmission units. For each unit u_i ($i \in [1, m]$), we suppose that the receiver requests it (i.e., sends a request) at T_i^{req} and receives it at T_i^{rcv}. $T_i^{delivered}$ is the timestamp of the most recently received data when sending a request for u_i.

Define

$$p(t) = \begin{cases} 1, & t = T_i^{rcv} \\ 0, & \text{others} \end{cases}, \quad i \in [1, m] \tag{3}$$

Then u_i generates a sample when it arrives at the receiver:

$$deliveryRate_i = \frac{\Delta delivered}{\Delta t} = \frac{\sum_{T_i^{req}}^{T_i^{rcv}} p(t)}{T_i^{rcv} - T_i^{delivered}} \tag{4}$$

$$rtt_i = T_i^{rcv} - T_i^{req} \tag{5}$$

Afterwards, we get

$$\widehat{\text{BtlBw}} = \text{WindowMax}(deliveryRate_i) \tag{6}$$

$$\widehat{\text{RTT}_{\min}} = \text{WindowMin}(rtt_i) \tag{7}$$

where the WindowMin gets the minimum value in the window of 10 seconds and the WindowMax gets the maximum value in window of 10 $\widehat{\text{RTT}_{\min}}$. Finally, according to the state machine, receiver-based BBR determines pacing_gain and cwnd_gain to control requesting data.

IEEE 802.11-based network is characterized by high degrees of aggregation (A-MPDU and A-MSDU in IEEE 802.11 n/ac) and RTT variance. [39] argues that BDP is not the optimal point to the Wi-Fi networks. In Figure 3(a), the authors think the point B is the best. It achieves a higher throughput, at the cost of a little extra delays. This is because a longer queue can fully utilize the aggregation functionality of Wi-Fi. In fact, in the phase of "bandwidth limited", the Round-Trip Time (RTT) of BBR over Wi-Fi is fluctuating. Despite this, BBR can always detect the $\mathrm{RTT}_{\min}$ due to its ProbeRTT state and the value of 0.75 for the pacing_gain in ProbeBW state. And this is shown in Figure 3(b). In Formula (7), it can be seen that BBR always uses the $\mathrm{RTT}_{\min}$ instead of the real-time fluctuating RTTs. Therefore, most of the time, each phase in the ProbeBW $\{1.25, 0.75, 1, 1, 1, 1, 1, 1\}$ lasts for the $\mathrm{RTT}_{\min}$ (i.e., $\widehat{\mathrm{RTT}}_{\min} = \mathrm{RTT}_{\min}$).

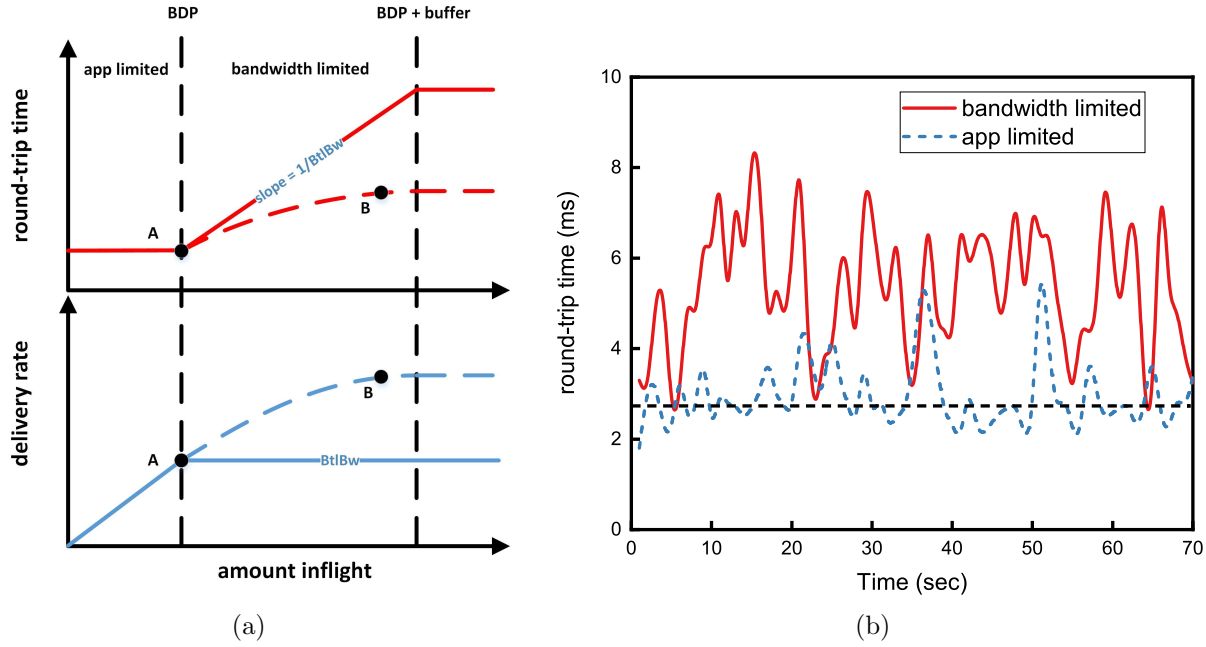


FIGURE 3. Characteristics of BBR over Wi-Fi: (a) Delivery rate and round-trip time vs. the amount of data inflight; (b) experimental data: round-trip time of BBR over Wi-Fi (IEEE 802.11ac)

4.2. **Request aggregation.** For efficiency, we introduce a request aggregation mechanism to determine the amount of requested data of one request. And we further adjust the degree of request aggregation to fit the wireless environment.

Request aggregation means that one request asks for more than 1× UNIT_MSS bytes. The motivations for request aggregation are as follows:

- In receiver-driven transmission, packets sending is limited by both sender and receiver's protocol stacks' sending performance. For many end devices, requesting data without aggregation cannot achieve high throughput anyway.
- One request just asking for UNIT_MSS bytes will result in high communication overheads of control frames.
- Since the requests pacing is based on hrtimer, too higher request frequency will accumulate more hrtimer timer errors and take more CPU resources. This will affect the accuracy of pacing and finally degrade the bandwidth.

In Figure 4, we experimentally show how request aggregation affects the actual throughput. With no limits of bottleneck bandwidth and congestion window, we use different request rates (200Mbps, 400Mbps, 600Mbps and 800Mbps) to ask for data. Here, UNIT_MSS is set to be 1400 bytes. "Aggregation counts n" ($n \in \{1, 2, 8, 20, 40, 80\}$) means that each request asks for $n \times$ UNIT_MSS bytes. It is obvious that requesting data with low aggregation count (e.g., 1, 2) has poor scalability, especially in high-bandwidth transmissions. It causes more "throughput loss" that the throughput cannot match the request rate anyway. Moreover, without request aggregation, the degree of "throughput loss" depends on the hardware performance of the receiver. The lower the performance, the higher the degree of bandwidth loss.

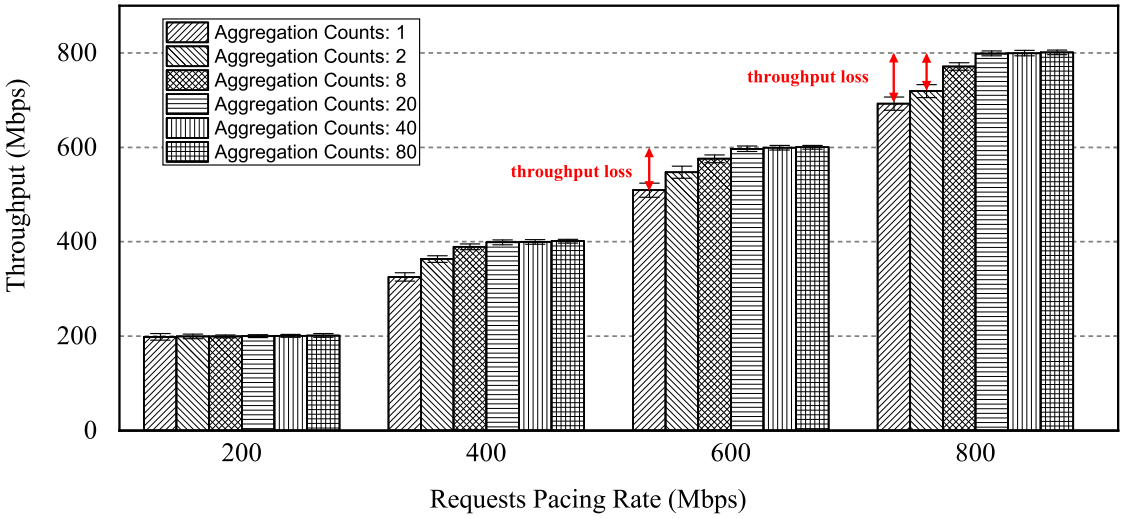


FIGURE 4. Throughput under different request rates and aggregation counts with no limits of bottleneck bandwidth and congestion window

However, high degree of request aggregation means that traffic seems like a burst in small timescale. And it is harmful to the fine-gained control over the transmission. In Wi-Fi networks, this may introduce packet loss, especially when multiple flows sharing a bottleneck. In fact, the request aggregation should balance communication overheads and control precision. We firstly set an aggregation count limit L which equals 64. It means that the amount of data requested by one request (including *length* field and option field) is no more than L $\times$ UNIT_MSS bytes. Then, we use the following equation to determine the aggregation count and the amount of data requested by one request is aggregation counts $\times$ UNIT_MSS bytes:

$$\text{Aggregation count} = \max\left(\min\left(\frac{\text{request rate}}{\text{UNIT\_MSS} * 1000}, \text{L}\right), 1\right) \tag{8}$$

where the request rate is in bytes/s and we prefer the amount of data requested per millisecond (ms) for that the order of magnitude of transmission is usually 10^{-3}(s). In other words, we tend to request some data every millisecond rather than requesting a large amount at a time. It is obvious that when request rate is low, we choose a low aggregation count which is enough to achieve saturated throughput. And when request rate is high, we use a high aggregation count to avoid "throughput loss". Finally, we utilize L ($= 64$) to limit the aggregation count in order to avoid serious bursty traffic.

Due to ICN's ubiquitous caching, users tend to access NDOs nearby. Therefore, with improvement of Wi-Fi efficiency, the transmission will be characterized by a very low delay (e.g., 2ms) in the state of "app limited". Given that the throughput of Wi-Fi is

not as high as wired networks, aggregation count will be small (e.g., 60Mbps throughput corresponds to the about 5 aggregation counts). And since each phase in the ProbeBW $\{1.25, 0.75, 1, 1, 1, 1, 1, 1\}$ lasts for the $\mathrm{RTT}_{\min}$ (ms), receiver-based BBR only requests a few more units (about $0.25 \times$ aggregation count $\times \mathrm{RTT}_{\min}$) in the gain phase (pacing_gain equals 1.25) to detect the more bandwidth. In Formula (4), Δt can be approximately equal to the real-time RTT which is fluctuating. Therefore, in such case, the samplings ($deliveryRate_i = \frac{\Delta delivered}{\Delta t}$) are hard to detect the real maximum bandwidth because the growth ratio of $\Delta delivered$ is likely to be offset by the fluctuation of real-time RTTs. In other words, the ability of (receiver-based) BBR to detect maximum bandwidth is not enough to make the transmission converge to the point B from A in Figure 3(a).

To solve this, we suggest enhancing the bandwidth detection capability in the context of Wi-Fi when BBR is in ProbeBW state and the following condition is met

$$\frac{\mathrm{avg}(rtt_i)}{\widehat{\mathrm{RTT}_{\min}}} > 1.25 \times \frac{\widehat{\mathrm{RTT}_{\min}}}{1} \tag{9}$$

where the meaning of rtt_i, $\widehat{\mathrm{RTT}_{\min}}$ has been explained above (in Section 4.1) and they are both in milliseconds. Besides the minimum RTT ($\widehat{\mathrm{RTT}_{\min}}$), we also record the average RTT in the window of 10 estimated RTTs. Based on this, we make a rough estimate that in the gain phase of ProbeBW, when the growth ratio of $\Delta delivered$ brought by pacing_gain of 1.25 is not obviously greater than the influence of RTT fluctuation, we think it is time to try to detect more bandwidth. And it is obvious that when the $\widehat{\mathrm{RTT}_{\min}}$ is smaller and real-time RTT is more fluctuating, Inequality (9) is more likely to be true.

To try to detect more bandwidth, we increase the request aggregation count. It is simple but effective. We increase the aggregate count by 3 folds and do not adjust the control parameters of receiver-based BBR so that it keeps a similar behavior at the macro level. For Wi-Fi, saturated throughput means full utilization of its aggregation functionality (e.g., A-MPDU or A-MSDU). And it can be defined that each time the access point's Wi-Fi driver competes and gets an opportunity to transmit data to the user, its buffer queue always has enough data whose size is greater than the maximum aggregation size. However, BBR's queue control may damage the throughput under the "noisy" of Wi-Fi's behaviors. In Figure 5, we give the waveform of packets arriving at access point's Wi-Fi driver queue in the small timescale. If the input rate of the queue is constant at the macro level, the large aggregation count can generate a longer queue in a short time. And this is more conducive to fully use the Wi-Fi's aggregation functionality, because when the access point obtains the transmit opportunities later, there is more likely to be enough data in the queue. Besides, increasing the aggregation count further decreases the communication overheads of control frames (requests), which is very meaningful in the context of Wi-Fi. On the other hand, even the duration of gain phase of ProbeBW is very short, a large aggregation count will not lead to a too weak ability of bandwidth detection.

4.3. **Cwnd adjustment.** Receiver-based BBR sometimes cannot achieve the maximum throughput over Wi-Fi. The main reason is cwnd exhaustion caused by aggregation and RTT variance. Normally, BBR set cwnd to be $2 \times$ BDP. In the wired network which is relatively stable, the amount of inflight data is usually near the BDP. There are enough quotas (cwnd-infligh) for BBR to request data faster to detect higher bandwidth during the gain phase where pacing_gain is 1.25. However, in the context of Wi-Fi, the amount of data inflight will achieve a high level which is far greater than one BDP. This is because data requested before does not arrive in time while the receiver continues to request new data. Therefore, there are likely no enough quotas in cwnd for receiver-based BBR to

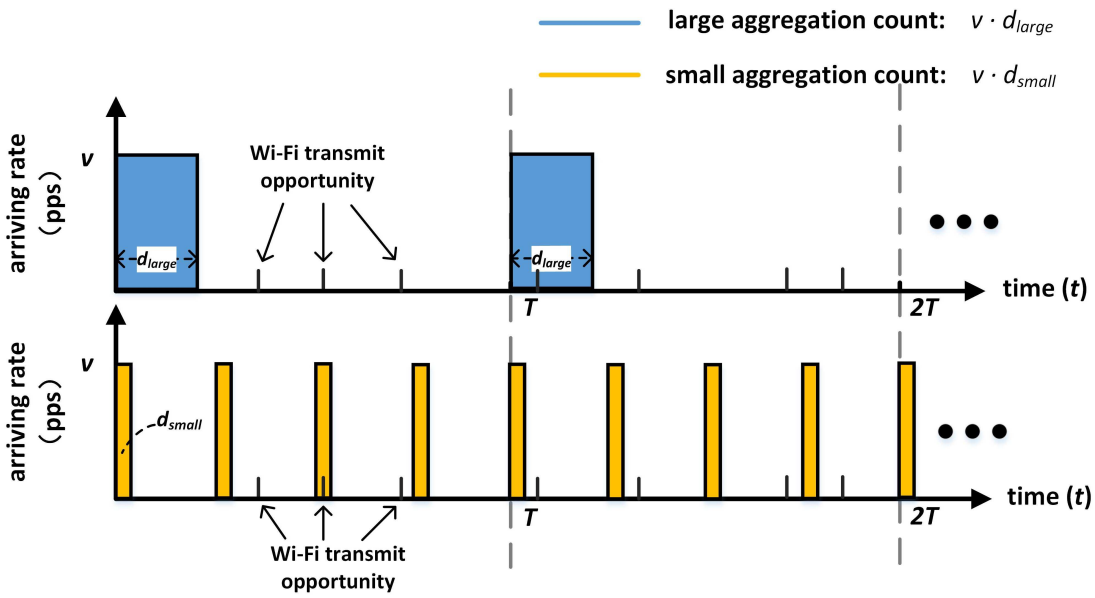


FIGURE 5. In small timescale, the waveform of packets arriving at access point's Wi-Fi driver queue is a series of pulses.

detect the maximum bandwidth of Wi-Fi. And when $\mathrm{RTT}_{\min}$ is smaller, RTT variation is greater than $\mathrm{RTT}_{\min}$, the cwnd exhaustion is more likely. It even can suppress data requesting in the whole cycle $\{1.25, 0.75, 1, 1, 1, 1, 1, 1\}$, which leads to link underutilization. Thus, the key is to have enough cwnd. [35] and [36] propose an aggregation estimator to estimate the degree of Wi-Fi aggregation. And it adds a value to the original cwnd ($2 \times$ BDP) to avoid cwnd exhaustion under RTT variance. By contrast, we adopt a direct and effective method for receiver-based BBR, namely ignoring cwnd's restriction on requesting data during the whole ProbeBW state. This adjustment does not degrade the advantage of BBR in aspect of delay, because BBR bounds the amount of inflight data through BDP instead of cwnd.

5. **Evaluation and Analysis.** In this section, we conduct extensive experiments to evaluate the performance of our receiver-driven transmission. We test the performance of receiver-driven transmission under various conditions, especially in the wireless network. The experimental results and analysis show the high performance and robustness of our receiver-driven transmission.

5.1. **Experimental setting.** In Figure 6, we build up a physical testbed, which involves three nodes – one client (user), one forwarding device and one server. Our receiver-driven transmission mechanism works in the client's kernel space. And a receive program runs on the client's application layer to receive data through POSIX APIs. Through Wi-Fi or Ethernet, the client connects to the forwarding device and continuously requests data from the server.

For comparison's sake, we mainly test TCP with BBR and TCP with CUBIC in our testbed. And to simulate various link conditions (e.g., delay, bandwidth, jitters), we utilize *tc* to set buffer queues, which have different QoS rules, to control traffic forwarding. The *tc* acts on both eth1 and eth2 instead of eth0 or eth3 because TSQ may limit TCP performance. During the transmission, we observe the size of buffer queue in data direction and the throughput of client.

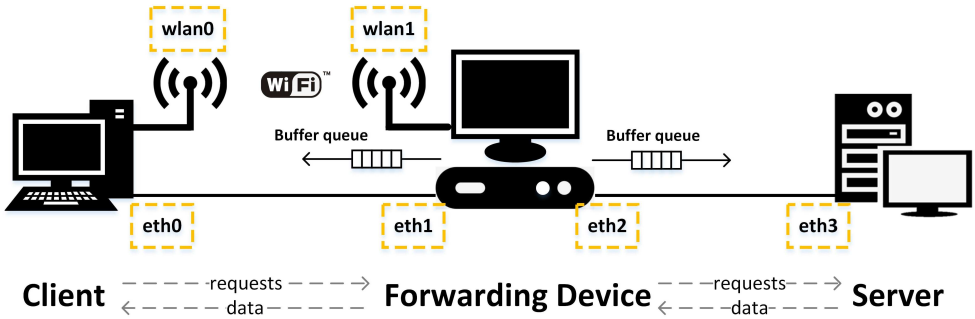
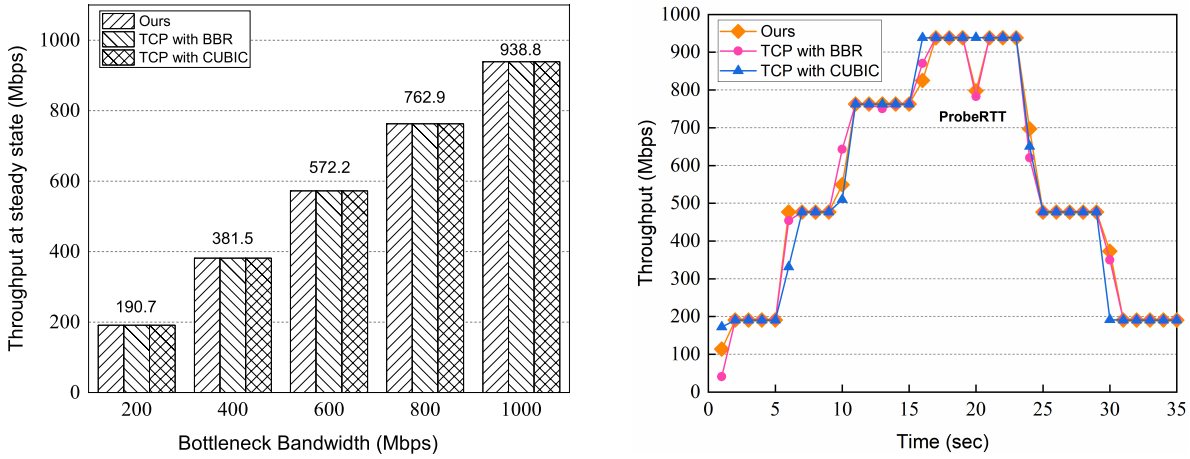


FIGURE 6. Physical testbed layout



(a) (b)

FIGURE 7. Bandwidth utilization under different bottleneck bandwidth: (a) Static tests; (b) dynamic tests in 35 seconds: RTT = 5ms, bottleneck bandwidth = 200 → 500 → 800 → 1000 → 500 → 200 (Mbps)

5.2. **Basic evaluation.** Bandwidth utilization is key to transmission. Under various link conditions, we test the throughput of our receiver-driven transmission at steady state. In Figure 7(a), it can be seen that receiver-driven transmission achieves a high bandwidth utilization same to TCP. Figure 7(b) shows the bandwidth utilization in 35 seconds under dynamic bottleneck bandwidth. It is obvious that our receiver-driven transmission has a good adaptability that it can fast converge to the high throughput when bottleneck bandwidth changes.

Chunk-based storage in ICN means that clients can access different pieces of one resource from multiple nodes simultaneously. To distinguish different flows and print them separately, we utilize *docker* to test. On the client, each container has its own virtual NIC and runs the receive program to request data from the server. In Figure 8, we test 4 receiver-driven flows sharing a 100Mbps-20ms bottleneck, 200Mbps-20ms bottleneck and 300Mbps-20ms bottleneck, respectively. It can be seen that 4 flows will converge to a fair share soon. The downward facing triangular structures are ProbeRTT states which can accelerate final convergence. Moreover, when a flow ends, other flows will soon take up the available bandwidth converging to a new fair share. And this does not cause the bandwidth loss – the total throughput ("Total" in Figure 8) keeps constant during the transmission.

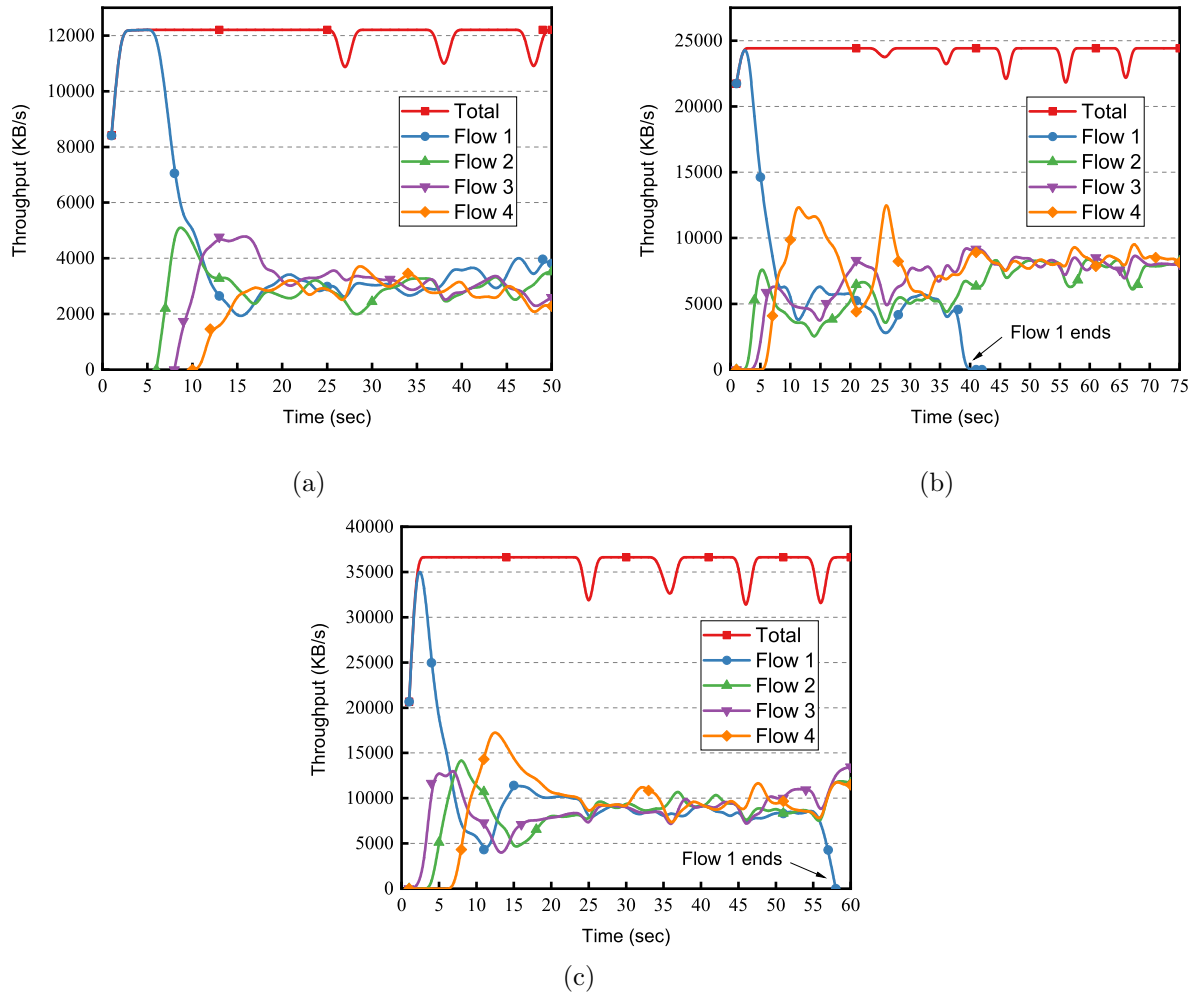


FIGURE 8. 4 receiver-driven flows sharing a bottleneck: (a) 100Mbps-20ms; (b) 200Mbps-20ms; (c) 300Mbps-20ms

5.3. **Evaluation over Wi-Fi.** We first give the proof of our analysis of problem in Section 4.2 and show the effectiveness of optimization. In Figure 9(a), the transmission link includes about a 2ms-RTT wireless link (IEEE 802.11ac) and about 1ms-RTT wired link. And in Figure 9(b), we set up a 5ms-RTT wired link. Along with the Wi-Fi hop, the transmission link has about 7ms RTT. We compare the throughput and RTT of receiver-driven transmission before and after optimization. It can be seen that compared with 2ms link, non-optimized transmission on 7ms link ("Throughput: non-optimized" in Figure 9(b)) can make better use of Wi-Fi's aggregation functionality to achieve saturated throughput. This is consistent with our previous analysis. The reason is that on 7ms link, receiver-based BBR has a longer gain phase (pacing_gain = 1.25) so that it can request more data, in which it is more likely to detect more bandwidth in the wireless network. By comparison, optimized receiver-driven transmission ("Thoughput: optimized" in Figure 9) does not have this problem. It can adapt to different Wi-Fi links achieving saturated throughput. Meanwhile, it introduces very few extra delays ("RTT: optimized" in Figure 9).

Figure 10 gives the experiments of throughput, delay and multiple flows in both IEEE 802.11ac and IEEE 802.11n networks. For comparison's sake, in addition to standard TCP, we reproduce the BBRp algorithm [39] and applied it to the receiver-driven transmission. We test two values of the parameter bbrp_pace of BBRp, namely bbrp_pace

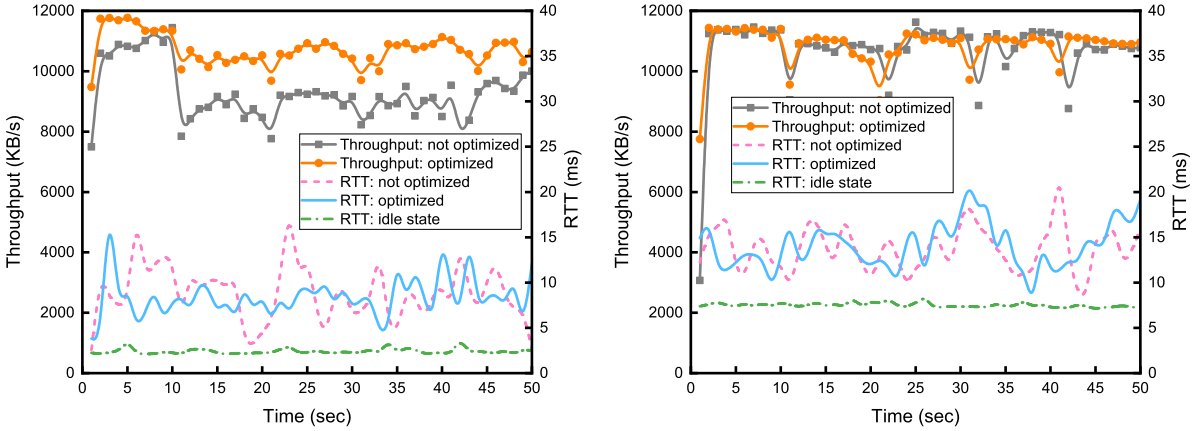


FIGURE 9. Throughput and RTT of receiver-driven transmission before and after optimization over IEEE 802.11ac. We take this to prove the analysis and optimization. (a) Only IEEE 802.11ac. (b) An IEEE 802.11ac hop along with a 5ms-RTT wired link.

= 6 and bbrp_pace = 8. The two values correspond to the pacing_gain of 1.5 and the pacing_gain of 2, respectively.

In Figures 10(a) and 10(d), it can be seen that our receiver-driven transmission achieves the full Wi-Fi bandwidth even slightly outperforming TCP with CUBIC in the IEEE 802.11ac scenarios. By comparison, TCP with the standard BBR and BBRp have the issue of cwnd exhaustion which limits data requesting and detecting maximum bandwidth. Therefore, their throughputs are not very high. In terms of RTT, our receiver-driven transmission just introduces slight extra delays, which is shown in Figures 10(b) and 10(e). By comparison, CUBIC brings a high delay in the lifetime of the transmission. In Figures 10(c) and 10(f), it can be seen that multiple receiver-driven flows can fast converge to a fair share soon.

5.4. **Resource consumption.** On a link with 500Mbps bottleneck bandwidth and 10ms RTT, we simulate the scenario where the CR simultaneously sends data to 100 users for 60 seconds. Table 1 shows that our receiver-driven transmission has a low CPU consumption, about 4.8%, while the TCP with BBR and TCP with CUBIC take 8.8% and 7.0%, respectively. "Memory usage" represents the memory consumed by transmission. In addition to the size of receiver queue, sender-based transmission mechanism must consume 3KB memory for each connection to record the state information. By comparison, in receiver-driven transmission, CRs are stateless and they just respond the content according to the incoming requests. On the other hand, to realize a complete transmission functionality on CR, receiver-driven transmission only needs about 500 extra lines of C code while sender-based one like TCP needs about 15000 lines which is mainly to deal with state

TABLE 1. Resource consumption on the server

	CPU usage (%)	**Memory usage per connection (bytes)**	**Code (lines)**	**Stateful**
Ours	4.8 ± 0.4	0	≈ 500	no
TCP with BBR	8.8 ± 0.6	≈ 3000	> 15000	yes
TCP with CUBIC	7.0 ± 0.5	≈ 3000	> 15000	yes

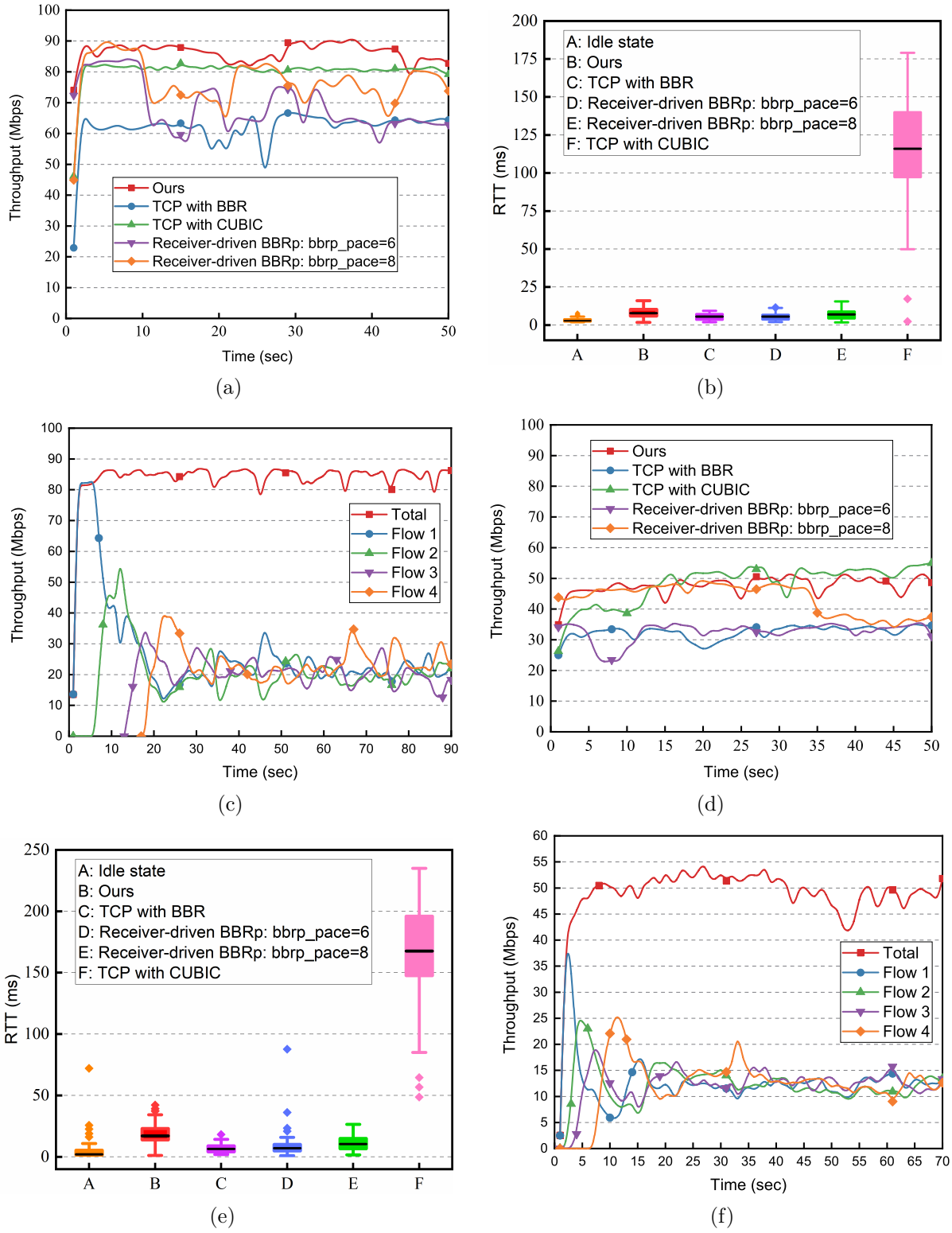


FIGURE 10. Throughput and delay evaluation over Wi-Fi: (a) Throughput comparison over IEEE 802.11ac; (b) RTT comparison over IEEE 802.11ac; (c) multiple flows in IEEE 802.11ac; (d) throughput comparison over IEEE 802.11n; (e) RTT comparison over IEEE 802.11n; (f) multiple flows in IEEE 802.11n

management. All in all, compared with sender-based transmission, receiver-driven transmission reduces the complexity and resource consumption of CR so that CR can focus on forwarding traffic, caching content and serving consumers.

6. **Conclusion.** In this paper, we introduce a receiver-driven transmission mechanism with BBR which is used in ICN over IP to provide pull-based, reliable, ordered delivery of streams of octets. We implement it in the Linux kernel space. Firstly, we show the receiver-driven transmission stack and describe the details of key functionalities, especially a receiver-based BBR algorithm with the pacing function. Then, we focus on transmission in the context of Wi-Fi and explore some issues. Through analysis of the transport model, we suggest adjusting degree of requests aggregation and increasing the congestion window (cwnd) to achieve saturated throughput. Extensive experiments are conducted on a physical testbed under various scenarios. Experimental results show that our receiver-driven transmission has a high bandwidth utilization, with low propagation delays and less consumption on servers.

Future work should focus on two aspects. Firstly, limiting burst traffic means a lot for transmission. The sender in our system just simply returns the requested data. This may lead to packet loss at network card queue when many requests arrive at sender at the same time. Thus, it is better to use a Token-Bucket based send queue to further limit serious burst traffic. Secondly, we will study the effect of receiver-driven transmission on wireless handoff, especially in the context where mobile hosts equip with multiple interfaces. We expect that our mechanism can fast recover the data transmission when the path switches.

Acknowledgment. This work is partially supported by Strategic Leadership Project of Chinese Academy of Sciences: SEANET Technology Standardization Research System Development (Project No. XDC02070100).

REFERENCES

[1] G. Xylomenos et al., A survey of information-centric networking research, *IEEE Commun. Surv. Tutorials*, vol.16, no.2, pp.1024-1049, doi: 10.1109/SURV.2013.070813.00063, 2014.

[2] L. Zhang et al., Named data networking, *SIGCOMM Comput. Commun. Rev.*, vol.44, no.3, pp.66-73, doi: 10.1145/2656877.2656887, 2014.

[3] A. Venkataramani, J. F. Kurose, D. Raychaudhuri, K. Nagaraja, M. Mao and S. Banerjee, MobilityFirst: A mobility-centric and trustworthy Internet architecture, *SIGCOMM Computer Communication Review*, vol.44, no.3, pp.74-80, doi: 10.1145/2656877.2656888, 2014.

[4] T. Koponen et al., A data-oriented (and beyond) network architecture, *SIGCOMM Computer Communication Review*, vol.37, no.4, p.181, doi: 10.1145/1282427.1282402, 2007.

[5] SAIL Project, SAIL deliverable B.3 (3.3), *Final NetInf Architecture*, http://www.sail-project.eu/deliverables/, 2013.

[6] D. Trossen, M. J. Reed, J. Riihijarvi, M. Georgiades, N. Fotiou and G. Xylomenos, IP over ICN – The better IP, *2015 European Conference on Networks and Communications (EuCNC)*, Paris, France, pp.413-417, 2015.

[7] M. Vahlenkamp, F. Schneider, D. Kutscher and J. Seedorf, Enabling ICN in IP networks using SDN, *2013 21st IEEE International Conference on Network Protocols (ICNP)*, Goettingen, Germany, pp.1-2, 2013.

[8] L. Ding, J. Wang and Q. Yang, Survey on architecture design of cache node in information center network, *Journal of Network New Media*, vol.8, no.3, pp.1-8, 2019.

[9] L. Ding, J. Wang, Y. Sheng and L. Wang, A split architecture approach to terabyte-scale caching in a protocol-oblivious forwarding switch, *IEEE Trans. Netw. Serv. Manage.*, vol.14, no.4, pp.1171-1184, doi: 10.1109/TNSM.2017.2761894, 2017.

[10] M. A. Sayeed and R. Shree, Optimizing unmanned aerial vehicle assisted data collection in cluster based wireless sensor network, *ICIC Express Letters*, vol.13, no.5, pp.367-374, 2019.

[11] A. Kuzmanovic and E. W. Knightly, Receiver-centric congestion control with a misbehaving receiver: Vulnerabilities and end-point solutions, *Computer Networks*, vol.51, no.10, pp.2717-2737, doi: 10.1016/j.comnet.2006.11.021, 2007.
[12] Y. Liu, *Receiver-Driven Transmission*, https://github.com/liu-yi-feng/receiver-driven-transmission, 2020.
[13] D. D. Clark, M. L. Lambert and L. Zhang, NETBLT: A high throughput transport protocol, *ACM SIGCOMM Computer Communication Review*, pp.353-359, 1988.
[14] P. Sinha, T. Nandagopal, N. Venkitaraman, R. Sivakumar and V. Bharghavan, WTCP: A reliable transport protocol for wireless wide-area networks, *Wireless Networks*, vol.8, nos.2/3, pp.301-316, doi: 10.1023/A:1013702428498, 2002.
[15] S. Floyd, M. Handley, J. Padhye and J. Widmer, Equation-based congestion control for unicast applications, *SIGCOMM Computer Communication Review*, vol.30, no.4, pp.43-56, doi: 10.1145/347057.347397, 2000.
[16] V. Tsaoussidis and C. Zhang, TCP-Real: Receiver-oriented congestion control, *Computer Networks*, vol.40, no.4, pp.477-497, doi: 10.1016/S1389-1286(02)00291-8, 2002.
[17] P. Mehra, A. Zakhor and C. de Vleeschouwer, Receiver-driven bandwidth sharing for TCP, *IEEE INFOCOM 2003. 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, San Francisco, CA, USA, pp.1145-1155, 2003.
[18] N. T. Spring, M. Chesire, M. Berryman, V. Sahasranaman, T. Anderson and B. Bershad, Receiver based management of low bandwidth access links, *Proc. of IEEE INFOCOM 2000. Conference on Computer Communications. 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, Tel Aviv, Israel, pp.245-254, 2000.
[19] R. Gupta, M. Chen, S. McCanne and J. Walrand, A receiver-driven transport protocol for the web, *Telecommunication Systems*, vol.21, nos.2/4, pp.213-230, doi: 10.1023/A:1020994414496, 2002.
[20] H.-Y. Hsieh, K.-H. Kim, Y. Zhu and R. Sivakumar, A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces, *Proc. of the 9th Annual International Conference on Mobile Computing and Networking – MobiCom'03*, San Diego, CA, USA, p.1, 2003.
[21] G. Carofiglio, M. Gallo and L. Muscariello, ICP: Design and evaluation of an Interest control protocol for content-centric networking, *Proc. of the IEEE INFOCOM 2012. Conference on Computer Communications Workshops*, Orlando, FL, USA, pp.304-309, 2012.
[22] G. Carofiglio, M. Gallo, L. Muscariello and M. Papali, Multipath congestion control in content-centric networks, *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Turin, pp.363-368, 2013.
[23] R. A. Potys, N. M. Ali, I. Marsh and F. Osmani, NetInf TP: A receiver-driven protocol for ICN data transport, *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, Portland, OR, USA, pp.267-272, 2015.
[24] S. Ha, I. Rhee and L. Xu, CUBIC: A new TCP-friendly high-speed TCP variant, *SIGOPS Oper. Syst. Rev.*, vol.42, no.5, pp.64-74, doi: 10.1145/1400097.1400105, 2008.
[25] J. Gettys and K. Nichols, Bufferbloat: Dark buffers in the Internet, *Queue*, vol.9, no.11, p.40, doi: 10.1145/2063166.2071893, 2011.
[26] A. Aggarwal, S. Savage and T. Anderson, Understanding the performance of TCP pacing, *Proc. of IEEE INFOCOM 2000. Conference on Computer Communications. 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, Tel Aviv, Israel, pp.1157-1165, 2000.
[27] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh and van Jacobson, BBR: Congestion-based congestion control, *Commun. ACM*, vol.60, no.2, pp.58-66, doi: 10.1145/3009824, 2017.
[28] Google, *bbr*, https://github.com/google/bbr, 2016.
[29] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, I. Swett, J. Iyengar, V. Vasiliev and V. Jacobson, *BBR Congestion Control: IETF 100 Update: BBR in Shallow Buffers*, https://datatracker.ietf.org/meeting/100/materials/slides-100-iccrg-a-quick-bbr-update-bbr-in-shallow-buffers, 2017.
[30] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, I. Swett, J. Iyengar, V. Vasiliev and V. Jacobson, *BBR Congestion Control: IETF 99 Update*, https://www.ietf.org/proceedings/99/slides/slides-99-iccrg-iccrg-presentation-2-00.pdf, 2017.
[31] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh and V. Jacobson, *BBR Congestion Control: An Update*, https://www.ietf.org/proceedings/98/slides/slides-98-iccrg-an-update-on-bbr-congestion-control-00.pdf, 2017.
[32] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh and V. Jacobson, *BBR Congestion Control*, https://www.ietf.org/proceedings/97/slides/slides-97-iccrg-bbr-congestion-control-02.pdf, 2016.

[33] N. Cardwell, Y. Cheng, S. H. Yeganeh and V. Jacobson, *BBR Congestion Control: Draft-cardwell-iccrg-bbr-congestion-control-00*, https://tools.ietf.org/html/draft-cardwell-iccrg-bbr-congestion-control-00, 2017.

[34] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, I. Swett, J. Iyengar, V. Vasiliev and V. Jacobson, *BBR Congestion Control Work at Google: IETF 102 Update*, https://datatracker.ietf.org/meeting/102/materials/slides-102-iccrg-an-update-on-bbr-work-at-google-00, 2018.

[35] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, I. Swett, J. Iyengar, V. Vasiliev and V. Jacobson, *BBR v2: A Model-Based Congestion Control*, https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00, 2019.

[36] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, I. Swett, J. Iyengar, V. Vasiliev and V. Jacobson, *BBR Congestion Control Work at Google: IETF 101 Update*, https://www.ietf.org/proceedings/101/slides/slides-101-iccrg-an-update-on-bbr-work-at-google-00, 2018.

[37] E. Atxutegi, F. Liberal, H. K. Haile, K.-J. Grinnemo, A. Brunstrom and A. Arvidsson, On the use of TCP BBR in cellular networks, *IEEE Commun. Mag.*, vol.56, no.3, pp.172-179, doi: 10.1109/MCOM.2018.1700725, 2018.

[38] C. A. Grazia, N. Patriciello, M. Klapez and M. Casoni, BBR+: Improving TCP BBR performance over WLAN, *ICC 2020 – 2020 IEEE International Conference on Communications (ICC)*, Dublin, Ireland, pp.1-6, 2020.

[39] C. A. Grazia, M. Klapez and M. Casoni, BBRp: Improving TCP BBR performance over WLAN, *IEEE Access*, vol.8, pp.43344-43354, doi: 10.1109/ACCESS.2020.2977834, 2020.

[40] Y. Cheng, N. Cardwell, S. H. Yeganeh and V. Jacobson, *Delivery Rate Estimation: Draft-cheng-iccrg-delivery-rate-estimation-00*, https://tools.ietf.org/html/draft-cheng-iccrg-delivery-rate-estimation-00, 2017.

[41] V. Paxson, M. Allman and J. Chu, *Computing TCP's Retransmission Timer*, https://datatracker.ietf.org/doc/rfc6298/?include_text=1, 2011.