SDATP: An SDN-Based Traffic-Adaptive and Service-Oriented Transmission Protocol

Jiayin Chen, Qiang Ye, *Member, IEEE*, Wei Quan, Si Yan, *Student Member, IEEE*, Phu Thinh Do, Peng Yang, *Member, IEEE*, Weihua Zhuang, *Fellow, IEEE*, Xuemin Shen, *Fellow, IEEE*, Xu Li, and Jaya Rao

***Abstract*—In this paper, a software-defined networking (SDN) based adaptive transmission protocol (SDATP) is proposed to support applications in fifth generation (5G) communication networks. For time-critical services, such as machine-type communication services for industrial automation, high reliability and low latency are required. To guarantee the strict service requirements, a slice-level customized protocol is developed with in-network intelligence, including in-path caching-based retransmission and in-network congestion control. To further reduce the delay for end-to-end (E2E) service delivery, we jointly optimize the placement of caching functions and packet caching probability, which reduces E2E delay by minimizing retransmission hops. Since the joint optimization problem is NP-hard, we transform the original problem to a simplified form and propose a low-complexity heuristic algorithm to solve the simplified problem. Numerical results are presented to validate the proposed probabilistic caching algorithm, including its adaptiveness to network dynamics and its effectiveness in reducing retransmission hops. Simulation results demonstrate the advantages of the proposed SDATP over the conventional transport layer protocol with respect to E2E packet delay.**

***Index Terms*—5G networks, SDN, adaptive transmission protocol, in-path packet caching/retransmission, probabilistic caching.**

I. Introduction

With the rapid development of the fifth generation (5G) communication networks, bandwidth-hungry applications are foreseen to be supported, which have stringent quality-of-service (QoS) requirements. With limited resources of network switches and tranmission links in current core networks, it is challenging to guarantee end-to-end (E2E) QoS requirements of diverse services [2], [3]. Therefore, an efficient transport-layer protocol is required for E2E QoS enhancement. Transmission control protocol (TCP) [4], [5] is widely used for reliable E2E packet transmissions in a packet switching network. In TCP, a two-way communication connection is established between a pair of end hosts for data transmission. The sender applies retransmission timeout (RTO) for lost packet recovery and mitigate network congestion with congestion window adjustment.

To guarantee E2E QoS requirements for diversified services, many existing studies propose enhanced transport-layer protocols in terms of E2E performance improvement, including avoidance of false fast retransmissions [6], [7], accurate RTO estimation [8], [9], and efficient congestion control [10], [11]. However, current transport-layer protocols (e.g., TCP and user datagram protocol (UDP)) only achieve best-effort E2E performance [12], due to the distributed and ossified network architecture. In TCP, a sending host records for every transmitted data packet a round-trip time (RTT) for packet loss detection and congestion control; a three duplicate acknowledgements (ACKs) mechanism is applied for fast retransmission of lost packets. However, the TCP congestion detection is performed only at a sending host based on feedback ACK information from its receiving end, without in-network congestion awareness, which leads to a long response delay for packet retransmission and congestion control.

Software-defined networking (SDN) is proposed to balance traffic load and reduce network congestion [13]–[15]. It decouples control and forwarding functions at network servers/switches. The routing decisions for each traffic flow[1] are made by a centralized SDN control module. With network-wide information, global optimal E2E routing paths can be established by the SDN control module, which realizes load balancing for network congestion alleviation. In addition, network function virtualization (NFV) emerges to provide flexible and cost-effective service through virtualization technology [16], [17]. In this way, different customized services can be supported over a physical network as embedded virtual networks.

Manuscript received September 9, 2019; revised November 19, 2019; accepted December 22, 2019. Date of publication December 31, 2019; date of current version June 9, 2020. This work was supported in part by the Research Grants from Huawei Technologies Canada and in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada. This paper was presented in part at IEEE ICC 2019 [1]. The associate editor coordinating the review of this article and approving it for publication was M. Chen. *(Corresponding author: Qiang Ye.)*

Jiayin Chen, Si Yan, Peng Yang, Weihua Zhuang, and Xuemin Shen are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: j648chen@uwaterloo.ca; s52yan@uwaterloo.ca; p38yang@uwaterloo.ca; wzhuang@uwaterloo.ca; sshen@uwaterloo.ca).

Qiang Ye is with the Department of Electrical and Computer Engineering and Technology, Minnesota State University, Mankato, MN 56001 USA (e-mail: qiang.ye@mnsu.edu).

Wei Quan is with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China (e-mail: dr.wei.quan@ieee.org).

Phu Thinh Do is with the Institute of Research and Development, Duy Tan University, Da Nang 550000, Vietnam (e-mail: dopthinh@gmail.com).

Xu Li and Jaya Rao are with the Ottawa Wireless Advanced System Competency Center, Huawei Technologies Canada Inc., Ottawa, ON K2K 3J1, Canada (e-mail: xu.lica@huawei.com; jaya.rao@huawei.com).

Digital Object Identifier 10.1109/TCCN.2019.2963149

[1] A traffic (service) flow refers to an aggregation of packets of same service type passing through a pair of edge switches in the core network.

The SDN architecture can reduce E2E delay by enabling packet caching and retransmission functions at the in-path network switches [18], [19]. The function placement realizes in-path packet loss recovery and retransmission, both of which can reduce the delay and overhead of packet loss recovery. Compared with a conventional TCP network, these in-network functionalities require computing, caching, and higher protocol-layer packet analyzing capabilities. Given a number of activated caching nodes, the location of these caching functionalities affects QoS provisioning. For a network routing path, if a caching functionality is activated near the receiving node, a high performance gain (e.g., reduced retransmission hops) is achieved for each individual lost packet which, on the other hand, reduces the caching effectiveness due to the small chances of being retransmitted by in-path caching nodes for loss recovery. The number of caching nodes also has an impact on the performance gain. With more caching nodes, the caching buffer overflow probability can be reduced, while the caching resource usage becomes less efficient. Thus, the caching performance should be optimized via trading off the caching effectiveness with resource utilization. Therefore, the caching node placement and packet caching probability should be jointly considered, accounting for their correlation in the caching scheme design, to improve caching efficiency, reduce retransmission overhead, and minimize the E2E packet delay for time-critical services (e.g., machine-type communication (MTC) for industrial automation).

In this paper, we present a comprehensive SDN-based adaptive transmission protocol (SDATP) supporting time-critical services that require high reliability and low-latency. Different from TCP operating on a packet switching network, the SDATP is based on an SDN/NFV enabled virtual network which is logically circuit switching. To satisfy the stringent requirements, we exploit both in-path packet caching and in-network congestion control. A joint optimization of caching function placement and probabilistic packet caching is studied to reduce the delay for retransmitted packets. This joint optimization problem is formulated as a mixed-integer non-linear programing (MINLP) problem to minimize the number of average packet retransmission hops. The dependence between packet caching probabilities and locations of caching node placement poses technical challenges in solving the problem. For a tractable solution, we simplify the problem by reducing the number of decision variables, which is achieved by considering the feature of in-path caching. Then, a low complexity heuristic algorithm is developed to solve the simplified problem. In comparison with several benchmark policies, the proposed probabilistic caching policy requires the minimum number of retransmission hops, thus achieves the fastest packet retransmission.

The remainder of this paper is organized as follows: The system model is described in Section III. In Section IV, we present the proposed customized protocol elements of SDATP to support time-critical services. Simulation results are given in Section VI to demonstrate the effectiveness of the proposed protocol. Finally, conclusions are drawn in Section VII.

II. Related Work

With the SDN architecture, in-network control intelligence can be utilized by transport-layer protocols to enhance congestion detection and congestion recovery. In-network statistics (e.g., resource utilization information) can be collected from OpenFlow switches; Accordingly, the SDN control module can make congestion detection and alleviation [20]–[22]. In [20], a software-defined TCP (SDTCP) is proposed where an SDN controller collects queue length information from each OpenFlow switch and calculates traffic shares for each service flow traversing the switch. If the traffic share of one flow exceeds a threshold, indicating certain level of congestion, the SDN controller will modify the flow table entries to migrate some of the flow traffic to alternative paths to alleviate the congestion. A scalable congestion control protocol is developed based on TCP in [21], where congestion control is performed at end hosts for a packet switching network.

The SDN architecture can reduce E2E delay by enabling in-path packet caching and retransmission functionalities. An SDN-based UDP framework is proposed in [23], where each network switch is equipped with retransmission functionality to perform in-path packet loss detection and recovery. Packet loss detection is based on the observed out-of-order packet reception, without distinguishing between packet retransmission and reordering. Caching and retransmission functions are activated at every switch, which increases resource usage at switches, and leads to significant signaling overhead between switches (e.g., for retransmission request and caching release). To reduce network cost and signaling overhead, caching functions are enabled only on selected in-network switches [24], [25]. All received packets are cached at each caching switch, which causes additional storage resource usage and processing delay by repetitive packet caching along the forwarding path. To better utilize storage resources and reduce processing delay, packets are cached with certain probability in [26], given caching node locations. To improve network performance, existing studies deal with the placement of caching functions and the packet caching probabilities as two separate problems in designing packet caching policies. However, based on E2E delay analysis, the impact of caching function placement and packet caching probabilities are coupled, which should be jointly studied to minimize E2E delay with limited caching resources.

III. System Model

In this section, we present the network model, protocol function modules, and traffic models for time-critical services. Important mathematical symbols are listed in Table I, where RRH stands for required retransmission hop.

A. Network Model

Consider multiple virtual networks embedded on a 5G physical infrastructure for diverse applications. Data traffic from a set of E2E connections of one type of service are aggregated

TABLE I
SUMMARY OF MATHEMATICAL SYMBOLS

Symbols	Definition
B_m	Caching buffer size for the m^{th} network switch
$\mathbb{C}(N) = \{C_n\}_{n=1}^{N}$	Index set of activated caching nodes
$F(C_n)$	Caching reliability of caching node C_n
F_N	Caching reliability with N caching nodes
$G\,(N)$	Gain achieved by N in-path caching nodes
$G_r\,(N)\,(G_o\,(N))$	$G\,(N)$ in caching buffer non-overflow (overflow) case
$G_{o,l}\,(N)\,(G_{o,u}\,(N))$	Lower (upper) bound of $G_o\,(N)$
$H(C_n)$	Eliminated RRHs for packets cached at caching node C_n by SDATP
L_m	The m^{th} link
M	Number of links
N	Number of activated caching nodes
$P_c(C_n)$	Caching probability of a packet passing through caching node C_n
$\mathbb{P}(N) = \{P_t(C_n)\}_{n=1}^{N}$	Set of the probabilities of a packet cached at caching node C_n
$Q_s(m)$	Probability of a packet successfully transmitting to the m^{th} network switch
q_m	Packet loss probability over the m^{th} hop
$R(m)$	Upper bound of Y, limited by the m^{th} network switch
R_N	Upper bound of Y for path with N caching nodes
S_m	The m^{th} network switch in routing path
T	Time interval of caching buffer releasing
U_n	Number of lost packets over the n^{th} caching-caching segment during T
W_m	Caching weight for the m^{th} network switch
Y	Packet arrival rate of traffic flow (packet per T)
λ	Average packet arrival rate of traffic flow (packet per T)

at virtual switches[2] (i.e., vSwitches) at the edge of the core network as one traffic flow. Thus, the edge vSwitch should be equipped with the capabilities of traffic aggregation and higher-layer network functionalities (e.g., header processing). Each E2E path consists of three domains: (i) between the source node and ingress edge vSwitch, (ii) between two edge vSwitches, (iii) between egress edge vSwitch and the receiving node. To guarantee protocol backward compatibility on end hosts, the connections between end hosts and edge vSwitch are established based on the state-of-the-art TCP protocol. We propose a customized protocol for enhanced data transmission between a pair of ingress and egress edge vSwitches. The protocol operates at slice level. We define one network *slice* as a combination of customized protocol elements, allocated resources, and the designed virtual network topology to support one aggregated service flow between a pair of edge vSwitches. Slices are differentiated and logically isolated to guarantee different levels of service QoS requirements. Each slice has a unique *slice ID* which is a combination of source and destination edge switch IP addresses and a pair of port numbers for E2E service delivery.

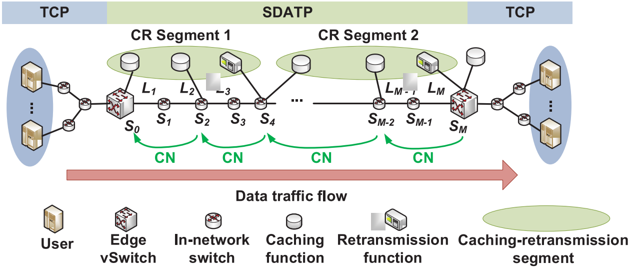


Fig. 1. Illustration of the embedded network topology under consideration.

We focus on one unicast service as an example. The aggregated traffic flow traverses through a linear network topology, as shown in Fig. 1. Let M denote the number of transmission links between edge vSwitches, and S_0 and S_M denote the ingress and egress edge vSwitches. The traversed path consists of $(M - 1)$ in-network vSwitches, denoted by $\{S_1, S_2, \ldots, S_{M-1}\}$, and M transmission links between S_0 and S_M, denoted by $\{L_1, L_2, \ldots, L_M\}$. In addition to all the vSwitches managed by an SDN controller, there exist conventional switches, with only packet forwarding capability (not shown in Fig. 1 for clarity). In the following, a switch refers to a vSwitch. Since some services require caching functions, both the two edge switches and the $(M - 1)$ in-network switches are equipped with caching buffers, the sizes of which are denoted as $\{B_0, B_1, \ldots, B_M\}$. For SDATP, caching functionalities are activated at $N\ (\leq M - 1)$ in-network switches, and the index for one caching node is the index of the corresponding network switch. The set of indexes for the caching nodes is denoted by $\mathbb{C}(N) = \{C_1, C_2, \ldots, C_N\}$. For example, we have $C_1 = 2$ and $C_2 = 4$ shown in Fig. 1. Since both edge switches (S_0 and S_M) are always activated as caching nodes to cache all received packets, the caching policy is designed only for in-network switches ($S_m\,(m = 1, 2, \ldots, M - 1)$). Packet loss at the m^{th} $(m = 1, 2, \ldots, M)$ transmission hop includes packet loss due to transmission errors of link L_m and congestion at switch S_{m-1}. If congestion happens at S_M, it will be handled by TCP operating between S_M and the receiving node.

[2] A virtual switch refers to a softwarized network switch with virtualized resources to program higher-layer network functions, apart from the functionalities of traffic forwarding or traffic aggregation.

B. Network Functionalities

Consider that the service requires high reliability and low latency where lost packets need to be retransmitted. A set of on-demand functionalities can be activated, including packet loss detection and packet retransmission realized by activating caching and retransmission functionalities at in-network switches.

1) Caching Function: A caching node is a network switch with activated caching function, equipped with pairs of data transmission and caching buffers. When a node receives a data packet, it caches the packet with a certain probability, which is referred to as *probabilistic caching*. The caching buffer is used for storing cached packets, while the data transmission buffer is used for queueing packets to be forwarded. Each network slice has a unique pair of data transmission and caching buffers.

Caching Release Function – Due to capacity constraint at each caching buffer, caching buffer release is necessary to avoid overflow, which is realized by transmitting a caching notification (CN) packet upstream from one caching node to its previous one for releasing cached packets, as shown in Fig. 1. Time is partitioned into intervals with constant duration T (in the unit of second), and a caching node sends a CN packet to its previous caching node in every T time interval.

Each CN packet conveys the following information: (1) Packet type, indicating that a packet is a caching notification packet; (2) Caching release window, showing the sequence numbers of successfully cached packets at a caching node during T. When a caching node receives a CN packet from its subsequent caching node, it releases the amount of cached packets according to the received caching release window; (3) Caching state information, which is the available buffer size of a caching node, indicating current caching node's packet caching capability. Such information is also used for in-network congestion control, to be discussed in Section IV-D.

2) Retransmission Function: A network switch with activated retransmission function is referred to as a retransmission node, with in-path packet loss detection and retransmission triggering functionalities. When a retransmission node detects packet loss, it triggers retransmission by sending a retransmission request to upstream caching nodes consecutively. If a requested data packet is recovered at a caching node, the packet will be retransmitted from that caching node. We refer to *one caching-retransmission (CR) segment* as a network segment, including the network switches/transmission links between two consecutive retransmission nodes, as shown in Fig. 1. The caching and retransmission functions within one CR segment cooperate to recover the packet loss happened in this segment, so multiple segments can operate parallelly to reduce the retransmision delay.

In the virtual network establishment, the SDN controller makes decisions on where to activate caching or retransmission functionality. To minimize the maximum packet retransmission delay,[3] both the ingress and egress edge switches are equipped with the caching functionality, while the egress edge switch is also enabled retransmission functionality. Specifically, based on packet loss probability and available resources over each transmission hop, the policy of activating caching and retransmission functionalities is discussed in Section V. If a packet is lost between two CR segments, the packet may be released by the caching nodes in previous CR segment, while it has not been received by the caching nodes in the next CR segment. Therefore, the packet loss between two CR segments requires retransmission from the source node, which leads to a longer delay than retransmission from in-path caching nodes. To ensure seamlessness in packet caching, the retransmission node of one CR segment is also the first caching node of its subsequent CR segment.

C. Traffic Model

Traffic arrivals for services requiring high reliability and low latency at the ingress edge vSwitch are modeled as a Poisson process, with the consideration of temporal independence of access requests from different users [27], [28]. The packet arrival rate of the aggregated flow ranges from tens to a few hundreds packets per second [27]. In each time interval of T, the number of packets arriving at edge switch S_0, denoted by Y, follows a Poisson distribution, with mean number of packet arrivals denoted by λ over T. It is proved in [29] that bounds of the median of Y satisfy

$$\lambda - \ln(2) \leq median(Y) < \lambda + \frac{1}{3}. \tag{1}$$

With λ much greater than 1, we have $median(Y) \approx \lambda$, i.e., $Pr\{Y \leq \lambda\} \approx 0.5$.

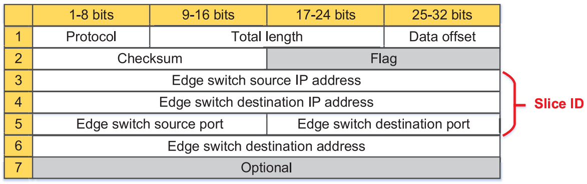
IV. Customized Protocol for Time-Critical Services

We propose a customized protocol to meet the high reliability and low latency requirements for time-critical services. The protocol function elements include connection establishment, data transmission, caching-based in-path packet retransmission, and congestion control. The E2E packet delay is to be minimized by activating these elements on demand. We also develop an optimization framework to determine the probabilistic caching policy, including the optimal number, placement, and packet caching probabilities of enabled caching nodes.

A. Connection Establishment

In conventional TCP for a packet switching network, due to a distributed network paradigm, a *three-way* handshake is required for connection establishment to ensure the reachability of an E2E path before data transmission. However, based

[3] Packet retransmission delay is the time duration from the instant that a retransmission request packet is sent (after detecting packet loss) to the instant that a retransmitted packet is received by the request node.



	1-8 bits	9-16 bits	17-24 bits	25-32 bits
1	Protocol	Total length		Data offset
2	Checksum		Flag	
3	Edge switch source IP address			
4	Edge switch destination IP address			
5	Edge switch source port		Edge switch destination port	
6	Edge switch destination address			
7	Optional			

Fig. 2. The SDATP packet header format.

on the SDN architecture, the three-way handshake process can be simplified.

(1) With the global information of network status, the SDN controller can check the E2E path availability. This is more efficient than the distributed signaling exchange;

(2) To prepare for data transmissions, the SDN controller assigns the initial sequence number and acknowledgement number to the end hosts;

(3) During the connection establishment, a customized E2E routing path can be established by the SDN controller according to certain service requirement.

Therefore, an SDN-based connection establishment mechanism is designed to reduce both the time consumption and signaling overhead for the connection establishment. A two-way handshake is applied to establish the connection for the forward and reverse directions respectively. The detailed procedure is presented in our previous work [1].

B. Data Transmission

1) SDATP Packet Format: To achieve efficient slice-level data transmission, a set of new packet formats are needed. Comparing with the conventional TCP/IP, the header format can be simplified in SDATP, due to the SDN based in-network transmission control. For example, the Acknowledgment field in the conventional TCP header is used to acknowledge each received packet. This field is removed from the SDATP packet header, thanks to its receiver-triggered packet loss detection. The SDATP packet header includes 24-byte required fields and 20-byte optional fields, as shown in Fig. 2, where slice ID consists of source and destination edge switch IP addresses and a pair of port numbers for E2E service delivery.

For packet forwarding, the required matching fields (i.e., slice ID) can be extracted by the OpenFlow-switches, which are used to match the cached flow entries for forwarding. Since some customized functionalities are introduced in SDATP, the header format is designed to support these new functionalities, such as in-path caching and retransmission, and caching-based congestion control. Therefore, in SDATP packet header, the Flag and Optional fields are used to differentiate important types of packets, including data packet, retransmission request (RR) packet, retransmission data (RD) packet, and CN packet, to support the enhanced protocol functionalities.

2) Header Conversion/Reversion: For compatibility with end hosts, the conventional TCP is applied between an end host and an edge switch. With the SDATP protocol for packet transmission between edge switches, packet header conversion and reversion for E2E communication are required, such as via the tunneling technology [30]. When a TCP packet is sent from the source node to the ingress edge switch, it is converted to an SDATP data packet, by adding a new SDATP header over the TCP header. When the SDATP packet arrives at the egress edge switch, it is reverted to a TCP packet by removing the SDATP header.

C. Packet Retransmission

In this subsection, we present the details of our proposed packet retransmsission scheme, including in-path receiver-based packet loss detection and caching-based packet retransmission.

1) Receiver-Based Packet Loss Detection: With activated retransmission functions, the in-network switches enable in-path receiver-based packet loss detection for fast packet loss recovery. If no packet loss happens, packets are expected to be received in sequence because of the linear topology. However, packet loss leads to enlarged time interval for consecutive packet reception or a series of disordered packet reception. Accordingly, the retransmission node measures the time intervals for consecutive packet receptions and extracts the number of received disordered packets. Two thresholds, named *expected interarrival time* and *interarrival counter threshold*, are maintained at the retransmission node to detect packet loss.

Since packet loss can happen for RR and RD packets, a retransmission node should be capable to detect the retransmitted packet loss and resend the RR packet. The detection is realized based on the measurement of each sampled packet retransmission delay, where the sample mean is estimated and used as the retransmission timeout threshold. The detailed information of how the loss detection thresholds are obtained is given in Appendix A.

After packet loss is detected, the retransmission node sends an RR packet upstream to its preceding caching node(s) in its CR segment. The sequence number information used for identifying loss packet(s) is included in the RR packet. This information is maintained by the retransmission node through establishing and updating an *expected packet list*, including which packets are not received and how the packets are detected to be lost. In addition, a *content window list* is established and maintained to record the received packets, which is used to update the expected packet list. More details of how to update these two lists are given in Appendix B.

2) RD Packet Retransmissions by Caching Nodes: The procedure of how caching and retransmission functions are operated within one CR segment is shown in Fig. 3. For packet loss recovery, both the first and last nodes in one CR segment are equipped with caching and retransmission functions. After an RR packet is received by a caching node, a range of sequence numbers and the retransmission triggering condition for the requested packets can be obtained. Based on the sequence number information, the caching node searches from its data caching buffer. If the requested packets are found, the RD packets are sent out downstream. Each RD packet also includes the timestamp from its received RR packet, which can be used for calculating packet retransmission delay. If current caching node cannot find the requested packets, the RR packet is forwarded upstream to preceding caching nodes until the requested packets are found.

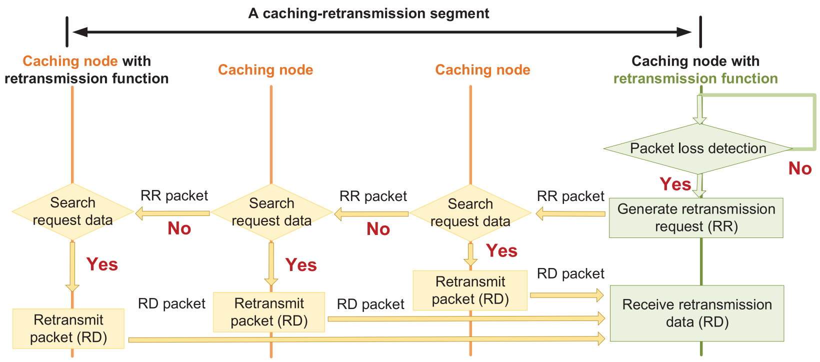


Fig. 3. An illustration of functionalities in a CR segment.

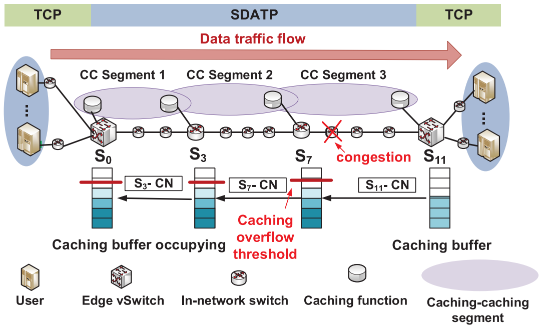


Fig. 4. Segment-level congestion control with $M = 11$.

D. Congestion Control

For each slice, communications between an end host and an edge switch follow TCP, while the proposed SDATP is used for the communications between two edge switches. Multiple slices share communication links and switches. With busy traffic flows, congestion happens when resource utilization is high, which leads to packet loss. Thus, we use packet loss as the indicator for congestion detection in the core network.

1) Caching-to-Caching Congestion Control: For the link congestion problem, we introduce caching-to-caching congestion control between two edge switches to mitigate congestion as shown in Fig. 4, where only caching nodes are specified for illustration purpose.

We define *one caching-caching (CC) network segment* which includes two consecutive caching nodes and the network switches/transmission links between them. Congestion detection for the n^{th} $(n = 1, \ldots, N-1)$ CC segment is based on CN packets sent from downstream caching node $S_{C_{n+1}}$ to upstream caching node S_{C_n}. From the caching release window field of a CN packet, the upstream caching node S_{C_n} obtains the number of successfully received packets at caching node $S_{C_{n+1}}$ during last *T*. Since S_{C_n} determines how many packets are transmitted during last *T*, the number of lost packets over the n^{th} CC segment, denoted by U_n, is calculated. Note that U_n is updated every *T*, and a large value of U_n indicates high congestion over the n^{th} CC segment. Therefore, based on the information in a CN packet, link congestion can be detected.

An example of caching-to-caching congestion control is given in Fig. 4, where 4 caching nodes (i.e., S_0, S_3, S_7, and S_{11}) are activated, and congestion happens in the third CC segment (i.e., between S_7 and S_{11}). Caching node S_7 detects the congestion from receiving a CN packet that indicates a high value of U_3, and starts to lower its sending rate to alleviate the congestion condition. Therefore, the local response time to congestion is *T*. The remaining caching buffer space of S_7 is decreased because of its lowered sending rate and the reduced number of released cached packets. Based on CN packets sent by node S_7, node S_3 obtains the remaining caching space of S_7 and compared with a threshold. If the remaining space is lower than the threshold, node S_3 estimates that caching buffer overflow likely happens at S_7, and slows down its sending rate to lower the risk. The congestion information is spread out upstream until it reaches the ingress edge switch (S_0). After S_0 estimates the potential caching buffer overflow at S_3, it reduces the sending rate, and updates the rate of replying ACKs according to its remaining caching buffer size.

When congestion happens, the nearby in-network caching node detects congestion by observing packet loss. Then, preceding caching nodes spread out the information upstream to the source node by estimating the risk of caching buffer overflow, which is guided by a threshold. Thus, the reaction time to the congestion depends on the caching overflow threshold. With a smaller caching overflow threshold, each preceding caching node can achieve a faster overflow estimation, which makes the source node exhibit a faster reaction. On the contrary, with a larger caching overflow threshold, the source node will send out more packets at the premise of not aggravating the congestion. By reducing the waiting time at the source node, a lower E2E transmission delay can be achieved. Thus, the caching overflow threshold is an important design parameter, due to its impact on both congestion reaction time of the source node and E2E transmission delay.

V. Optimized Probabilistic Caching Policy

To satisfy high-reliability and low-latency requirements of services, we introduce in-path caching-based retransmission in SDATP, which is achieved through activating caching and retransmission functions at in-network switches. Furthermore, to minimize the average number of packet retransmission hops, we propose an optimized packet caching policy, to determine caching node placement, packet caching probability, and the number of activated caching nodes.

With the embedded network topology supporting time-critical services shown in Fig. 1, a packet loss at the m^{th} link indicates that the packet has been successfully transmitted through the first ($m - 1$) links and lost at the m^{th} link (or transmission hop), the probability of which is given by

$$\begin{cases} p_1, & \text{if } m = 1 \\ Q_s(m-1) \times q_m, & \text{if } m = 2, 3, \ldots, M \end{cases} \tag{2}$$

where q_m denotes the packet loss probability over link $m\,(m = 1, 2, \ldots, M)$ between edge switches caused by network congestion and/or link errors, $Q_s(m-1) = \prod_{i=1}^{m-1}(1 - p_i)$ is the successful transmission probability of a packet over the first ($m - 1$) transmission hops before arriving at switch S_m. Thus, the probability of packet loss between S_m and edge switch S_M is $[Q_s(m) - Q_s(M)]$. Within a time interval of T, switch S_m successfully receives $Y \cdot Q_s(m)$ packets, among which $Y \cdot [Q_s(m) - Q_s(M)]$ packets are lost on average between S_m and S_M.

For packet retransmission, both the delay and transmission resource consumption are proportional to the required retransmission hops (RRHs). Thus, we describe the performance of in-path caching-based retransmission in terms of RRH number in one T. Specifically, the benefit achieved by exploiting in-path caching nodes is the eliminated RRHs compared with no in-path caching nodes.

To improve the caching efficiency, packets can only be cached once, indicating the already cached packets cannot be cached again. Thus, the probability of a packet cached at S_{C_n} after passing through a sequence of caching nodes $\{S_{C_i}, i = 1, 2, \ldots, n-1\}$ is given by

$$P_t(C_n) = \begin{cases} P_c(C_n), & \text{if } n = 1 \\ \prod_{i=1}^{n-1}[1 - P_c(C_i)] \cdot P_c(C_n), & \text{if } n = 2, 3, \ldots, N, \end{cases} \tag{3}$$

where $P_c(C_n)$ denotes the caching probability of a packet passing through caching node S_{C_n}.

With in-path caching, if a packet cached at S_{C_n} is lost and is requested for retransmission, it can be retransmitted by caching node S_{C_n}. Otherwise, it needs to be retransmitted from the ingress edge switch. Compared to the case without in-path caching, at least C_n transmission hops can be avoided for each packet retransmission. With caching buffer capacity B_{C_n} (i.e., maximum number of cached packets) allocated to S_{C_n}, the number of packets that can be cached at S_{C_n} is $\min[Y \cdot Q_s(C_n) \cdot P_t(C_n), B_{C_n}]$. Then, the reduction in the average number of RRHs for packets cached at S_{C_n} during T is given by

$$H(C_n) = \min[Y Q_s(C_n) P_t(C_n), B_{C_n}] \frac{Q_s(C_n) - Q_s(M)}{Q_s(C_n)} C_n. \tag{4}$$

A. Problem Formulation

The performance gain is represented by a ratio of the total number of eliminated RRHs for packets cached at all caching nodes to the total number of RRHs for all lost packets without in-path caching. That is,

$$\begin{aligned} &\frac{\sum_{n=1}^{N} H(C_n)}{Y \cdot [1 - Q_s(M)] \cdot M} \\ &\quad = \frac{\sum_{n=1}^{N} \min[Y \cdot Q_s(C_n) \cdot P_t(C_n), B_{C_n}] \frac{Q_s(C_n) - Q_s(M)}{Q_s(C_n)} \cdot C_n}{Y \cdot [1 - Q_s(M)] \cdot M}. \end{aligned} \tag{5}$$

Since Y is a random variable, we calculate the expectation of (5) as

$$\begin{aligned} G(N) &= \mathbb{E}\left\{ \frac{\sum_{n=1}^{N} H(C_n)}{Y \cdot [1 - Q_s(M)] \cdot M} \right\} \\ &= \frac{\sum_{n=1}^{N} \frac{Q_s(C_n) - Q_s(M)}{Q_s(C_n)} C_n \mathbb{E}\left\{ \min\left[Q_s(C_n) P_t(C_n), \frac{B_{C_n}}{Y} \right] \right\}}{[1 - Q_s(M)] \cdot M} \end{aligned} \tag{6}$$

where $\mathbb{E}\{\cdot\}$ is the expectation operation. In (6), we have

$$\begin{aligned} &\mathbb{E}\left\{ \min\left[Q_s(C_n) \cdot P_t(C_n), \frac{B_{C_n}}{Y} \right] \right\} \\ &\quad = F(C_n) \cdot Q_s(C_n) \cdot P_t(C_n) + B_{C_n} \cdot \sum_{k=R(C_n)+1}^{\infty} \frac{1}{k} \cdot \frac{\lambda^k}{k!} e^{-\lambda} \end{aligned} \tag{7}$$

where $F(C_n)$ denotes the caching buffer non-overflow probability (i.e., caching reliability) at switch S_{C_n}, given by

$$\begin{aligned} F(C_n) &= Pr\{Y Q_s(C_n) P_t(C_n) \le B_{C_n}\} \\ &= Pr\{Y \le R(C_n)\}. \end{aligned} \tag{8}$$

For Y is limited by the caching buffer capacity of S_{C_n}, the traffic rate upper bound, $R(C_n)$ is given by

$$R(C_n) = \left\lfloor \frac{B_{C_n}}{Q_s(C_n) \cdot P_t(C_n)} \right\rfloor \tag{9}$$

where $\lfloor \cdot \rfloor$ is the floor function.

It is indicated in (5) that the performance gain depends on the in-path caching policy. Therefore, to minimize the average packet retransmission delay (i.e., to maximize $G(N)$), we consider to optimize the number of enabled caching nodes, N, caching function placement indicated by $\mathbb{C}(N) = \{C_1, C_2, \ldots, C_N\}$, and the set of packet caching probabilities $\mathbb{P}(N) = \{P_t(C_1), P_t(C_2), \ldots, P_t(C_N)\}$. The optimization problem for probabilistic caching is formulated to maximize $G(N)$ as

$$(\mathbf{P1}): \max_{N, \mathbb{C}(N), \mathbb{P}(N)} G(N)$$

$$\text{s.t.} \begin{cases} 1 \le N \le M - 1, N \in \mathbf{Z}^+ & \text{(10a)} \\ 1 \le C_1 < C_2 < \cdots < C_N \le M - 1 & \text{(10b)} \\ C_n \in \mathbf{Z}^+, n = 1, 2, \ldots, N & \text{(10c)} \\ 0 < P_c(C_n) \le 1, n = 1, 2, \ldots, N. & \text{(10d)} \end{cases}$$

This optimization problem is an MINLP problem and is difficult to solve due to high computational complexity. In the following, based on design principles for $\mathbb{P}(N)$, we derive $\mathbb{P}(N)$ in terms of N and $\mathbb{C}(N)$, upon which ($\mathbf{P1}$) can be simplified with a reduced number of decision variables.

There are two principles for determining $\mathbb{P}(N)$: 1) All transmitted packets should be cached in path between the edge switches (i.e., $P_c(C_N) = 1$ and $\sum_{n=1}^{N} P_t(C_n) = 1$), which guarantees that all lost packets can be retransmitted from in-path caching nodes; 2) The caching buffer overflow probabilities at all caching-enabled switches should be equal, in order to balance the caching resource utilization at different in-path caching nodes.

To satisfy principle 2), $R(C_n)$ (or $F(C_n)$) is the same for all $n \in \{1, 2, \ldots, N\}$. Thus, we use R_N and F_N to represent $R(C_n)$ and $F(C_n)$ for simplicity. Based on (9), $P_t(C_n)$ is given by

$$P_t(C_n) = \frac{B_{C_n}}{Q_s(C_n) \cdot R_N} \tag{11}$$

and the summation of $P_t(C_n)$ over all caching nodes is

$$\begin{aligned}\sum_{n=1}^{N} P_t(C_n) &= \sum_{n=1}^{N} \frac{B_{C_n}}{Q_s(C_n) \cdot R_N}\\ &= \frac{1}{R_N} \cdot \sum_{n=1}^{N} \frac{B_{C_n}}{Q_s(C_n)} = 1.\end{aligned} \tag{12}$$

Therefore, R_N is expressed as

$$R_N = \sum_{n=1}^{N} \frac{B_{C_n}}{Q_s(C_n)} \tag{13}$$

and F_N is calculated based on (13) as

$$F_N = Pr\{Y \le R_N\} = \sum_{k=0}^{R_N} \frac{\lambda^k}{k!} e^{-\lambda}. \tag{14}$$

Based on (11) and (13), $\mathbb{P}(N)$ is expressed as a function of *N* and $\mathbb{C}(N)$, and (**P1**) is simplified in terms of decision variables.

To further reduce the problem complexity, we simplify the objective function of (**P1**) through algebraic manipulation. In (**P1**), *G*(*N*) is a summation of $G_r(N)$ and $G_o(N)$, which represents a combination of performance gains in caching buffer non-overflow and overflow cases, respectively. Thus, we obtain $G_r(N)$ and $G_o(N)$ as

$$G_r(N) = \frac{\sum_{n=1}^{N} [Q_s(C_n) - Q_s(M)] \cdot C_n \cdot \frac{B_{C_n}}{Q_s(C_n)}}{[1 - Q_s(M)] \cdot M \cdot \sum_{n=1}^{N} \frac{B_{C_n}}{Q_s(C_n)}} \cdot F_N \tag{15}$$

and

$$G_o(N) = \frac{\sum_{n=1}^{N} [Q_s(C_n) - Q_s(M)] \cdot C_n \cdot \frac{B_{C_n}}{Q_s(C_n)} \cdot A}{[1 - Q_s(M)] \cdot M} \tag{16}$$

where $A = \sum_{k=R_N+1}^{\infty} \frac{1}{k} \cdot \frac{\lambda^k}{k!} e^{-\lambda}$. Since the calculation of infinite series in A incurs high computational complexity, we calculate its lower and upper bounds, denoted by A_l and A_u. We first determine A_l as

$$\begin{aligned}A_l &= \sum_{k=R_N+1}^{\infty} \frac{1}{k+1} \cdot \frac{\lambda^k}{k!} e^{-\lambda} = \frac{1}{\lambda} \sum_{k=R_N+2}^{\infty} \frac{\lambda^k}{k!} e^{-\lambda}\\ &= \frac{1}{\lambda}(1 - F_N - Pr\{Y = R_N + 1\}).\end{aligned} \tag{17}$$

Then, we have

$$\begin{aligned}A - A_l &= \sum_{k=R_N+1}^{\infty} \left(\frac{1}{k} - \frac{1}{k+1}\right) \cdot \frac{\lambda^k}{k!} e^{-\lambda}\\ &< \frac{1}{\lambda} \cdot \frac{1}{R_N + 1} \sum_{k=R_N+2}^{\infty} \frac{\lambda^k}{k!} e^{-\lambda}\\ &= \frac{A_l}{R_N + 1}.\end{aligned} \tag{18}$$

From (18), the upper bound of *A* is derived as

$$A < A_l + \frac{A_l}{R_N + 1} = A_l \cdot \frac{R_N + 2}{R_N + 1} = A_u. \tag{19}$$

With A_l, the lower bound of $G_o(N)$ is expressed as

$$\begin{aligned}G_{o,l}(N) = &\frac{\sum_{n=1}^{N} [Q_s(C_n) - Q_s(M)] \cdot C_n \cdot \frac{B_{C_n}}{Q_s(C_n)}}{[1 - Q_s(M)] \cdot M \cdot \lambda}\\ &\times (1 - F_N - Pr\{Y = R_N + 1\}).\end{aligned} \tag{20}$$

Using A_u, the upper bound of $G_o(N)$ is calculated as

$$G_{o,u}(N) = \frac{R_N + 2}{R_N + 1} \cdot G_{o,l}(N). \tag{21}$$

When the caching buffer overflow happens at S_{C_n}, some cached packets are dropped from the buffer. Later on, if the retransmission requests are triggered for those cached packets, the packets need to be retransmitted from the ingress edge switch, which requires C_n hops more than that if retransmitted from S_{C_n}. Also, the caching resources for those overflowed packets at S_{C_n} are wasted without any performance gain. Since caching buffer overflow leads to a decrease of performance gain and a wastage of caching resources, the caching buffer non-overflow probability F_N for the optimal caching strategy tends to 1. Hence, R_N is expected not to be less than λ, which is the median of *Y* as discussed in Section III-C. With λ ranging from tens to hundreds of packets per *T*, the gap between $G_{o,u}(N)$ and $G_{o,l}(N)$ is small. Therefore, we use $G_{o,l}(N)$ to estimate $G_o(N)$. Then, the objective function *G*(*N*) in (**P1**) is simplified to $G_r(N) + G_{o,l}(N)$, and (**P1**) is transformed to

$$(\mathbf{P2}):\ \max_{N, \mathbb{C}(N)}\ G_r(N) + G_{o,l}(N)$$

$$\text{s.t.}\ \begin{cases} 1 \le N \le M - 1, N \in \mathbf{Z}^+ & \text{(22a)}\\ 1 \le C_1 < C_2 < \cdots < C_N \le M - 1 & \text{(22b)}\\ C_n \in \mathbf{Z}^+, n = 1, 2, \ldots, N. & \text{(22c)}\end{cases}$$

B. Optimized Probabilistic Caching

By solving (**P2**), an optimal number of caching nodes, *N*, and its corresponding $\mathbb{C}(N)$ can be obtained to maximize *G*(*N*). However, (**P2**) is a nonlinear integer programming problem, which is difficult to solve [31]. For tractability, we design a low-complexity heuristic algorithm as Algorithm 1 to jointly optimize *N* and $\mathbb{C}(N)$.

If a caching node is placed near the receiving node, a large number of retransmission hops (C_n) is avoided for each individual retransmitted packet, but the cached packets experience a low caching efficiency (i.e., the probability of

Algorithm 1 The Probabilistic Caching Algorithm

```
Input: Loss probability $\{q_1, q_2, ..., q_M\}$, caching buffer
    resource $\{B_1, B_2, ..., B_{M-1}\}$;
    Traffic load information, packet arrival rate $\lambda$
Output: Caching placement set $\mathbb{C}(N^*)$ and caching proba-
    bility set $\mathbb{P}(N^*)$
1:  Initialization: Set performance gain $G^*$ to 0
2:  Calculate $W_m$ for each in-network switch
3:  Rank $W_m$ in a descending order
4:  for $N = 1{:}M-1$ do
5:      Find $N$ switches with largest $W_m$
6:      Get the index set as caching node set $\mathbb{C}(N)$
7:      Calculate $G(N)$ for $\mathbb{C}(N)$ based on (15) and (20)
8:      if $G(N) > G^*$ then
9:          Set $N^*$ to $N$
10:         Set $\mathbb{C}(N^*)$ to $\mathbb{C}(N)$
11:         Set $G^*$ to $G(N)$
12:     end if
13: end for
14: Calculate caching probability $\mathbb{P}(N^*)$ with $N^*$ and $\mathbb{C}(N^*)$
    based on (11) and (13)
```

being requested for retransmission). Therefore, for caching placement $\mathbb{C}(N)$, there is a trade-off between the reduced retransmission hops and the caching efficiency. In terms of the number of caching nodes, a large value of N means more caching resources are available to guarantee a low caching buffer overflow probability. However, packets are distributively cached at caching nodes, with decreased caching resource utilization. A maximal $G(N)$ can be achieved through balancing the trade-off between the buffer overflow probability and the utilization of caching resources of switches.

Specifically, we define caching weight W_m for in-network switch S_m as the product of its caching efficiency, reduced retransmission hops of one cached packet (m), and caching buffer size (B_m) to represent its contribution to the overall performance gain. That is

$$W_m = \frac{[Q_s(m) - Q_s(M)]}{Q_s(m)} \cdot m \cdot B_m \tag{23}$$

where the caching efficiency of S_m is given by $\frac{[Q_s(m)-Q_s(M)]}{Q_s(m)}$. The higher the value of W_m, the greater the contribution. Therefore, the caching nodes are selected based on W_m in our proposed probabilistic caching algorithm (Algorithm 1). Given N, $\mathbb{C}(N)$ is determined in line 5 and line 6 of Algorithm 1, and $G(N)$ is calculated based on (15) and (20). Then, we iterate N to find the N value that achieves the maximal $G(N)$ (from line 8 to line 12).

C. Time Complexity

The time complexity for calculating $[G_r(N) + G_{o,l}(N)]$ in **(P2)**, which linearly increases with the number of caching nodes N, i.e., $\mathcal{O}(N)$. Based on Algorithm 1, $[G_r(N) + G_{o,l}(N)]$ in **(P2)** is calculated for each N ($N = 1, 2, \ldots, M-1$). Thus, the time complexity is given by

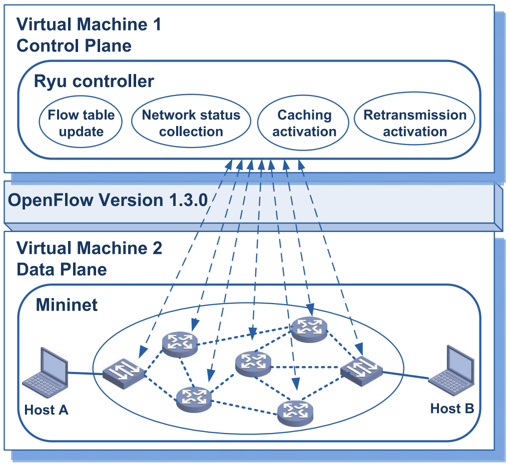


Fig. 5. Simulation platform for the proposed protocol to support time-critical service.

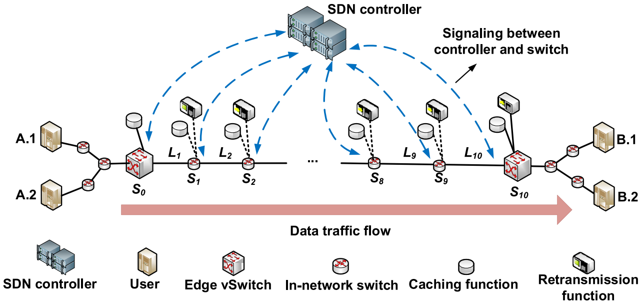


Fig. 6. Network topology for time-critical service.

$$\sum_{N=1}^{M-1} \mathcal{O}(N) = \mathcal{O}\left(\sum_{N=1}^{M-1} N\right) = \mathcal{O}\left(M^2\right). \tag{24}$$

For the brute-force method, there are C_{M-1}^{N} candidate sets for caching placement for a given N value, and $[G_r(N) + G_{o,l}(N)]$ needs to be calculated for each candidate set. Therefore, the total time complexity is derived as

$$\sum_{N=1}^{M-1} \mathrm{C}_{M-1}^{N} \cdot \mathcal{O}(N) = \mathcal{O}\left(\sum_{N=1}^{M-1} \frac{(M-1)!}{N!(M-1-N)!} \cdot N\right) = \mathcal{O}\left(M \cdot 2^M\right). \tag{25}$$

Compared with the brute-force method, our proposed probabilistic caching policy generates a solution with a much lower time complexity.

VI. Numerical Results

To demonstrate the effectiveness of the proposed SDATP, both analytical and simulation results are presented in this section. The SDN-based network architecture is shown in Fig. 5, in which two separated virtual machines, with 8 GB physical memory and 12 virtual processors, are utilized to simulate the data and control planes, respectively. In the data plane, the network elements are emulated by Mininet, including end hosts, edge/in-network switches, and transmission

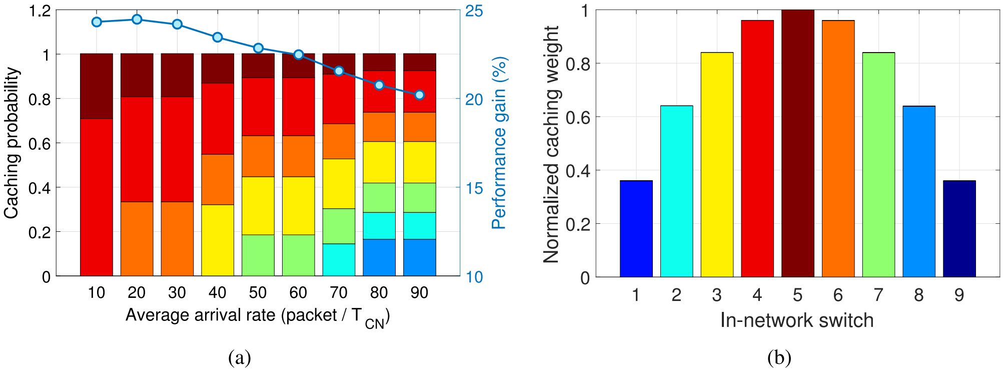


Fig. 7. Scheme adaptiveness to the packet arrival rate: a) caching node placement and performance gain; b) normalized caching weight for each node.

TABLE II
SIMULATION PARAMETERS

Packet Size	1400 byte	
Between user and edge node		
Source node	Data sending rate	1.25 Mbps
Link	Transmission rate	2.5 Mbps
	Propagation delay	10 ms
Between edge nodes		
Switch	Transmission data buffer	100 packets
	Caching buffer	5 to 20 packets
Link	Transmission rate	5 Mbps
	Propagation delay	10 ms
	Packet loss rate	0.05 %

links [32]. Both edge and in-network switches are OpenFlow vSwitch [33]. In the control plane, the SDN controller is implemented by the Ryu framework [34]. For the SDN southbound interface, we use OpenFlow version 1.3.0 [35] for simulation.

An SDN-based linear network topology is deployed for supporting a time-critical service flow, as shown in Fig. 6, where two E2E source-destination node pairs, (A.1, B.1) and (A.2, B.2), are connected via eleven switches (two edge switches and nine core network switches) under the SDN control. Traffic flows from both source nodes are aggregated at edge switch S_0 and are then sent to corresponding destination nodes through egress edge switch S_{10}. A detailed simulation setting is given in Table II. The caching function is activated at ingress edge switch, S_0, while both caching and retransmission functions are activated at egress edge switch, S_{10}. In addition, in-path functions are activated based on the proposed caching placement scheme in order to adapt to varying network conditions.

A. Adaptive Probabilistic Caching Policy

We evaluate the adaptiveness of the proposed probabilistic caching policy with respect to the varying packet arrival rate. Analytical results are obtained using MATLAB, where link packet loss rates and switch caching buffer sizes are set using parameters in Table II.

The number of active devices (e.g., activating devices in smart alarming system) varies with time, leading to different traffic volumes during peak and off-peak hours. To evaluate the adaptiveness of probabilistic caching policy, we vary the packet arrival rate of an aggregated flow, from 10 packets per T to 90 packets per T. The performance gain defined in (6) indicates the improvement of using in-path caching on the reduced number of retransmission hops. Given packet arrival rate, the caching policy (i.e., caching placement and caching probability) is shown in Fig. 7(a). The caching nodes with normalized weight $W_m(m = 1, 2, \ldots, M - 1)$ are differentiated by colours as shown in Fig. 7(b). The height of each coloured bar in Fig. 7(a) denotes the caching probability (i.e., $P_t(C_n)$ in (3)) of a specific node. To ensure all transmitted packets are cached in path, the summation of caching probabilities equals to 1. It shows that more caching nodes are activated upon a traffic load increase, which is necessary to guarantee caching reliability. However, to accommodate the increasing traffic, some nodes with a small weight are selected, which leads to a decreased performance gain.

We also compare the proposed policy with two other in-path caching policies, called *RP* and *HP* policies. In RP policy, the in-path caching nodes are randomly selected. In HP policy, the nodes immediately before the links with high packet loss probability are activated as in-path caching nodes. Given the packet arrival rate is 50 packets per *T*, caching placement and packet caching probability for each node are designed based on the proposed probabilistic caching (PC) algorithm, RP, and HP methods. In addition, we compare the PC method with a modified PC (*MPC*), in which each packet can be cached at multiple locations. For the proposed PC scheme, the number of activated caching nodes is 4, and nearly 24% of retransmission hops can be avoided as shown in Fig. 8, which outperforms the other three methods. In addition, since PC outperforms MPC, it is reasonable that we make each packet only be cached once over the network path between the edge switches in the simulated networking environment.

To evaluate the accuracy of proposed algorithm, we compare the performance gain of the PC policy and the optimal policy achieved by brute-force algorithm shown in Fig. 9, where packet arrival rate is 30 packets per T. It shows that the optimal number N^* of activated caching nodes exists and the gap of optimal performance gains between the PC policy and the brute-force method is small. We also demonstrate the performance gain through simulations. It can be seen that the

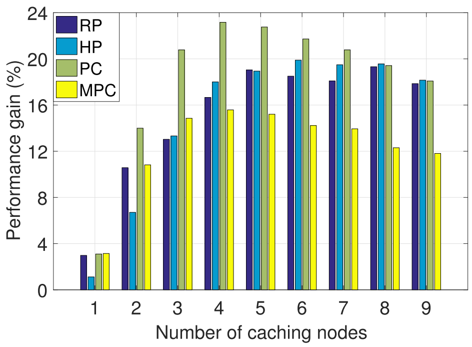


Fig. 8. Performance gain achieved by different methods.

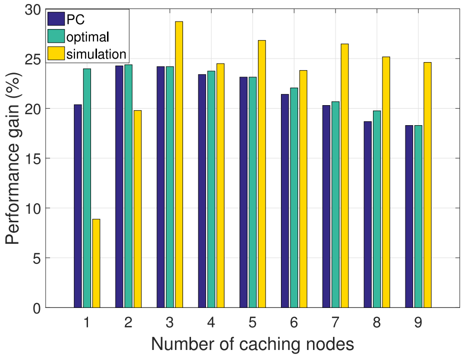


Fig. 9. Performance gain versus number of caching nodes.

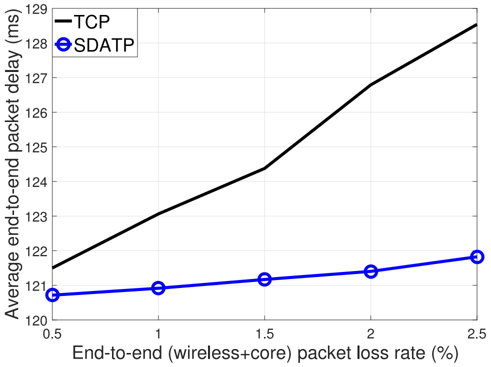


Fig. 10. A comparison of average E2E packet delay between TCP and SDATP without congestion.

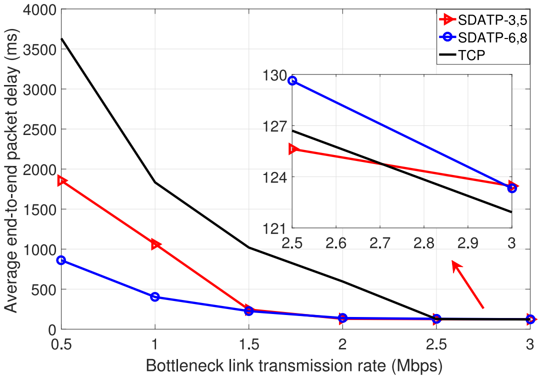


Fig. 11. A comparison of average E2E packet delay between TCP and SDATP with congestion.

simulation results closely match the analytical results under different number of caching nodes.

B. E2E Packet Delay

The E2E packet delay of our proposed SDATP is compared with that of TCP. We compare the average E2E delay (i.e., the duration from the time instant that a packet is sent from the source node till the instant that it is received by the destination node, averaged over all received packets) between SDATP and TCP for packets sent from $A.1$ to $B.1$. Two scenarios are considered: a) Packet loss happens due to link errors; b) Packet loss happens due to congestion. Based on the link propagation delay setting, the E2E delay for packet transmission, assuming no packet loss happens, is 120 ms, which is the lower bound of the E2E packet delay. Due to time consumed in packet loss detection and packet retransmission, the average E2E delay in both scenarios are larger than 120 ms.

1) Packet Loss Due to Link Errors: Fig. 10 shows the average E2E delay of TCP and SDATP when the E2E packet loss rate varies, where packet loss rate between a user and an edge switch ranges from 0% to 1% and between in-network switches is 0.05%.

It is observed that SDATP achieves a shorter average E2E delay than that of TCP. For TCP, packet loss is detected at the source, with an E2E round trip packet transmission time, before a packet retransmission is performed. For SDATP, the in-path caching-based retransmission enables packet loss detection within a CR segment. The time duration for loss detection and retransmission triggering is much shorter than the E2E round trip time, leading to a shorter average E2E delay.

2) Packet Loss Due to Congestion: Consider that packet loss over all links in the network are due to congestion. The transmission rate over the physical link between S_5 and S_6 is denoted by $\mathcal{C}$ Mbps. We set $\mathcal{C}$ smaller than 5 Mbps such that congestion can happen due to transmission bottleneck between S_5 and S_6.

Fig. 11 shows a comparison of the average E2E delay between TCP and SDATP, where $\mathcal{C}$ varies from 0.5 Mbps to 3 Mbps to indicate different levels of congestion. For SDATP, the locations of congestion and caching nodes have impact on both congestion control and retransmission performance. Therefore, we place the caching nodes before (i.e., at S_3 and S_5) and after (i.e., at S_6 and S_8) the congestion link, respectively. When $\mathcal{C}$ varies from 0.5 Mbps to 2 Mbps, the congestion condition is severe, and packets are lost frequently due to data buffer overflow. Compared with TCP, the average E2E packet delay is reduced for SDATP due to early packet loss detection and shortened packet retransmission delay, as shown in Fig. 11. Moreover, placing caching nodes after the congestion link outperforms placing before the congestion, since the congestion is detected earlier by S_0, with CN packets sent from S_6. However, when the congestion level is low (i.e., $\mathcal{C}$ varies from 2.5 Mbps to 3 Mbps), SDATP has a higher average E2E packet delay than that of TCP, due to time consumption for packet caching in SDATP. In contrast to heavy congestion, placing caching nodes before the congestion link acheives better performance, due to fast packet retransmission from S_5.

VII. Conclusion

In this paper, we propose an SDN-based traffic adaptive transport-layer protocol for customized services with low latency and high reliability requirements in 5G core networks. With SDN, in-network intelligence is enabled in SDATP, such as in-path caching-based retransmission and in-network caching-based congestion control. To further enhance the SDATP performance, we jointly optimize the placement and packet caching probability for enabled caching nodes to reduce retransmission hops. Due to the high computational complexity in solving the problem, we propose a low-complexity heuristic algorithm to achieve the solution. We evaluate the adaptiveness of SDATP, and show that high performance gain can be achieved under a varying traffic load. The proposed probabilistic caching scheme outperforms three other in-path caching schemes in reducing retransmission hops. Simulation results are also presented to demonstrate the advantages of the proposed SDATP over the conventional TCP in terms of E2E packet delay for both link error and link congestion cases. For future work, we will design an adaptive in-network caching policy for uncertain network dynamics. Model-free machine learning methods (e.g., reinforcement learning) will be considered for the policy development.

Appendix A
Thresholds for Packet Loss Detection

We provide details of how packet loss is detected based on the threshold design. Specifically, a packet loss detection depends on the measurements of the time intervals for consecutive packet receptions (denoted by *InterTime*) and the number of received disordered packets (denoted by *InterCnt*), while retransmission delay is measured for retransmitted packet loss detection. If at least one of these measurements exceeds corresponding threshold, a packet loss is detected.

Interarrival Timeout – The time interval between consecutive packet receptions is interarrival time, which indicates the conditions of link congestion and packet loss. At a retransmission node, there is an elapsed timer initiating at the instant of receiving a packet. If time duration is longer than a threshold, *expected interarrival time*, a packet loss is detected due to interarrival timeout. We obtain the expected interarrival time by linear prediction, based on sampled interarrival time durations at the retransmission node.

Interarrival Counter Threshold – Since the packets from one service slice are forwarded following the linear topology, out-of-order packet reception at a retransmission node indicates packet loss. After a packet loss is detected by a retransmission node, an RR packet is sent from the retransmission node to its upstream caching nodes, and an RD packet will be generated and sent by a caching node. When the RD packet is received by the downstream retransmission nodes, it leads to out-of-order packet reception. Thus, a retransmission node can detect packet loss depending on the level of packet disorder (i.e., packet disorder length).

To avoid spurious packet loss detection, the retransmission nodes should estimate an updated packet disorder length as threshold for InterCnt in the expected list. To do so, each retransmission node needs to determine where a packet loss actually happens. If the packet loss happens in its own segment, the retransmission node sends an RR packet to trigger retransmission; otherwise, the retransmission node estimates an updated disorder length for a retransmitted packet to avoid duplicate retransmissions. The updated disorder length is set as the interarrival counter threshold (CntThres) for the packet in the expected packet lists.

Based on signaling exchange with other retransmission nodes, one retransmission node can update the packet-level thresholds that are differentiated for each packet. To reduce the signaling overhead, the retransmission node can also obtain the segment-level threshold through sampling and estimation, based on the InterCnt of received out-of-order packets.

Retransmission Timeout – After packet loss is detected, RR and RD packets are transmitted for loss recovery. Since packet loss can happen to both RR and RD packets, the node that triggers a retransmission should be able to detect the retransmitted packet loss and resend an RR packet. To detect the loss, we use the expected retransmission delay as a threshold for RTTimer in the expected list, in which the estimation is based on the sampled packet retransmission delay. The time of sending an RR packet is recorded in its timestamp field, and is included in the corresponding RD packet when it is generated. When the RD packet is received at a retransmission node, the retransmission delay is calculated, i.e., the duration from the time instant the RR packet is sent till the instant the RD packet is received. The calculation of thresholds, including the expected interarrival time, packet-level and segment-level CntThres, and the expected retransmission delay, is presented in detail in our previous work [1].

Appendix B
Terminology for Packet Loss Detection

After the connection establishment, a *content window list* is established and maintained at each retransmission node, to record the packets successfully received at the node. At the same time, an *expected packet list* is established to record the packets that are expected to be received at the retransmission node. If packet loss is detected, contents in the expected packet list are referred for triggering packet retransmissions.

Content Window List – In the content window list, packets received in sequence are described by one window. Each window is bounded by its *left edge* and *right edge*, which indicate the first sequence number and the next expected sequence number of the received packets respectively. Thus, the sequence numbers of sequentially received packets lie between left edge and right edge of one content window. Due to packet loss, the packets may not be received in sequence, and a new window is generated in the list. Whenever a new packet is received, the content window list is updated by the retransmission node.

Expected Packet List – To record the information of packets that are expected by a retransmission node, the expected packet list has following fields:

(1) *Num* – When packets are lost discontinuously, expected packets are in separated packet windows, which are described by different entries in the expected list. The Num field represents the sequence of the entries;

(2) *StartSeq* and *EndSeq* – Specify an sequence number interval with a start and an end sequence numbers;

(3) *StartNum* – Indicate the number of packet offset between the target packet and the packet with StartSeq as its sequence number;

(4) *InterCnt* – Record how many packets have been received after the last sequentially received packet, when packets are received discontinuously;

(5) *CntThres* – Indicate the threshold set for InterCnt in packet loss detection;

(6) *WaitLen* – Measure the difference between CntThres and InterCnt;

(7) *RTCnt* – Record the number of retransmission requests sent for lost packets;

(8) *RTType* – Indicate how packet loss is detected by packet interarrival time exceeding a timeout, InterCnt greater than CntThres, or the duration from triggering the retransmission request to receiving the retransmitted packet exceeding a timeout;

(9) *RTTimer* – Denote an elapsed timer starting from the time instant of sending a retransmission request. If this time duration is larger than a retransmission timeout, another retransmission request of the lost packet is triggered.

The expected packet list is established based on the corresponding content window list at a retransmission node. For each entry in the expected packet list, StartSeq is set as the value in the right edge field of corresponding content window, and EndSeq is set as the maximum sequence number less than the left edge of its subsequent content window. In the case of a determined EndSeq, the packets with the sequence numbers lying between StartSeq and EndSeq are expected; In the case of an undetermined EndSeq (set as infinity), the expected packet is located by using StartSeq and StartNum. As the end sequence number for the last content window cannot be determined, we set StartNum as 0 to indicate the expected packet after the last content window. When a new entry is established, we initialize CntThres and WaitLen to 1, and the other fileds to 0.

When a caching node receives an RR packet from a retransmission node, it generates an RD packet whose header includes the StartSeq and StartNum information from the received RR packet. When receiving the RD packet, the retransmission node not only uses the RD sequence number, but also uses the StartSeq and StartNum pair to match the entries in the expected packet list. After an entry in the list is matched, it is removed from the list, indicating that the RD packet is successfully received. If the packet interarrival time at a retransmission node is longer than a threshold, indicating consecutive packet loss, the expected packets after the last entry of the content window list will be requested for retransmission one by one. StartSeq and StartNum in the expected packet list are used to locate each expected packet. Whenever packet loss is detected due to packet interarrival timeout, a new entry is added to the expected packet list, where StartSeq remains unchanged, and StartNum is incremented by one (pointing to the subsequent expected packet).

The header format of an SDATP packet is shown in Fig. 2, in which the Flag field indicates whether an RR packet is triggered by interarrival timeout, exceeding an interarrival counter threshold, or is triggered by retransmission timeout. For packet loss detected by exceeding interarrival counter threshold and retransmission timeout, both StartSeq and EndSeq are specified in the RR packet, indicating the range of sequence numbers of lost packets. For packet loss detected by interarrival timeout where EndSeq is unknown, StartSeq and StartNum in the RR packet are used to locate a specific expected packet for retransmission. The time instant of generating this RR packet is represented by the Timestamp field.

Data traffic from different slices may go though a common network switch sharing a set of resources. However, each slice has an independent set of content window list, expected packet list, and the associated parameters and variables maintained at each retransmission node.

Acknowledgment

The authors would like to thank Andre Daniel Cayen and Zichuan Wei (Undergraduate Research Assistants) for their help in conducting simulations.

References

[1] J. Chen *et al.*, "An SDN-based transmission protocol with in-path packet caching and retransmission," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–6.

[2] I. Parvez, A. Rahmati, I. Guvenc, A. I. Sarwat, and H. Dai, "A survey on low latency towards 5G: RAN, core network and caching solutions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3098–3130, 4th Quart., 2018.

[3] V.-G. Nguyen, A. Brunstrom, K.-J. Grinnemo, and J. Taheri, "SDN/NFV-based mobile packet core network architectures: A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1567–1602, 3rd Quart. 2017.

[4] J. Postel, "Transmission control protocol," IETF, Fremont, CA, USA, RFC 793, Sep. 1981.

[5] D. Borman, B. Braden, V. Jacobson, and R. Scheffenegger, "TCP extensions for high performance," IETF, Fremont, CA, USA, RFC 7323, Sep. 2014.

[6] E. Miravalls-Sierra, D. Muelas, J. Ramos, J. E. L. de Vergara, D. Morató, and J. Aracil, "Online detection of pathological TCP flows with retransmissions in high-speed networks," *Comput. Commun.*, vol. 127, pp. 95–104, Sep. 2018.

[7] S. Kadry and A. E. Al-Issa, "Modeling and simulation of out-of-order impact in TCP protocol," *J. Adv. Comput. Netw.*, vol. 3, no. 3, pp. 220–224, Sep. 2015.

[8] J. Zhou, Z. Li, Q. Wu, P. Steenkiste, S. Uhlig, J. Li, and G. Xie, "TCP stalls at the server side: Measurement and mitigation," *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, pp. 272–287, Feb. 2019.

[9] Y. Edalat, J.-S. Ahn, and K. Obraczka, "Smart experts for network state estimation," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 622–635, Sep. 2016.

[10] P. Yang, J. Shao, W. Luo, L. Xu, J. Deogun, and Y. Lu, "TCP congestion avoidance algorithm identification," *IEEE/ACM Trans. Netw.*, vol. 22, no. 4, pp. 1311–1324, Aug. 2013.

[11] S. Ferlin, Ö. Alay, T. Dreibholz, D. A. Hayes, and M. Welzl, "Revisiting congestion control for multipath TCP with shared bottleneck detection," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, San Francisco, CA, USA, Apr. 2016, pp. 1–9.

[12] S. P. Tinta, A. E. Mohr, and J. L. Wong, "Characterizing end-to-end packet reordering with UDP traffic," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Aug. 2009, pp. 321–324.

[13] K. Bao, J. D. Matyjas, F. Hu, and S. Kumar, "Intelligent software-defined mesh networks with link-failure adaptive traffic balancing," *IEEE Trans. Cogn. Commun. Netw.*, vol. 4, no. 2, pp. 266–276, Jun. 2018.

[14] H. Soni, W. Dabbous, T. Turletti, and H. Asaeda, "NFV-based scalable guaranteed-bandwidth multicast service for software defined ISP networks," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 4, pp. 1157–1170, Dec. 2017.

[15] N. Zhang *et al.*, "Software defined networking enabled wireless network virtualization: Challenges and solutions," *IEEE Netw.*, vol. 31, no. 5, pp. 42–49, May 2017.

[16] O. Alhussein *et al.*, "Joint VNF placement and multicast traffic routing in 5G core networks," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–6.

[17] J. Li, W. Shi, Q. Ye, W. Zhuang, X. Shen, and X. Li, "Online joint VNF chain composition and embedding for 5G networks," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–6.

[18] S. Yan *et al.*, "Transmission protocol customization for network slicing: A case study of video streaming," *IEEE Veh. Technol. Mag.*, vol. 14, no. 4, pp. 20–28, Dec. 2019.

[19] Y. Hao, M. Li, D. Wu, M. Chen, M. M. Hassan, and G. Fortino, "Human-like hybrid caching in software-defined edge cloud," *IEEE Internet Things J.*, to be published.

[20] Y. Lu, Z. Ling, S. Zhu, and L. Tang, "SDTCP: Towards datacenter TCP congestion control with SDN for IoT applications," *Sensors*, vol. 17, no. 1, p. 109, Jan. 2017.

[21] J. Hwang, J. Yoo, S.-H. Lee, and H.-W. Jin, "Scalable congestion control protocol based on SDN in data center networks," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, San Diego, CA, USA, Dec. 2015, pp. 1–6.

[22] T. Hafeez, N. Ahmed, B. Ahmed, and A. W. Malik, "Detection and mitigation of congestion in SDN enabled data center networks: A survey," *IEEE Access*, vol. 6, pp. 1730–1740, 2017.

[23] M.-H. Wang, L.-W. Chen, P.-W. Chi, and C.-L. Lei, "SDUDP: A reliable UDP-based transmission protocol over SDN," *IEEE Access*, vol. 5, pp. 5904–5916, 2017.

[24] M. P. McGarry, R. Shakya, M. I. Ohannessian, and R. Ferzli, "Optimal caching router placement for reduction in retransmission delay," in *Proc. 20th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2011, pp. 1–8.

[25] M. Zirak, M. H. Yaghmaee, and S. R. K. Tabbakh, "A distributed cache points selection scheme for reliable transport protocols with intermediate caching in wireless sensor networks," in *Proc. 16th Int. Conf. Adv. Commun. Technol. (ICACT)*, Feb. 2014, pp. 705–708.

[26] H. Chen *et al.*, "A reliable transmission protocol based on dynamic link cache," in *Proc. Int. Conf. Internet Things 4th Int. Conf. Cyber Phys. Soc. Comput.*, Dalian, China, Oct. 2011, pp. 752–755.

[27] Y. Liu, C. Yuen, X. Cao, N. U. Hassan, and J. Chen, "Design of a scalable hybrid MAC protocol for heterogeneous M2M networks," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 99–111, Feb. 2014.

[28] E. Soltanmohammadi, K. Ghavami, and M. Naraghi-Pour, "A survey of traffic issues in machine-to-machine communications over LTE," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 865–884, Dec. 2016.

[29] K. P. Choi, "On the medians of gamma distributions and an equation of Ramanujan," *Proc. Amer. Math. Soc.*, vol. 121, no. 1, pp. 245–251, May 1994.

[30] B. Davie and J. Gross, *A Stateless Transport Tunneling Protocol for Network Virtualization (STT)*, Internet Eng. Task Force, Fremont, CA, USA, Apr. 2016. [Online]. Available: https://tools.ietf.org/html/draft-davie-stt-08

[31] R. Hemmecke, M. Köppe, J. Lee, and R. Weismantel, "Nonlinear integer programming," in *50 Years of Integer Programming 1958–2008*. Berlin, Germany: Springer, Nov. 2009, pp. 561–618.

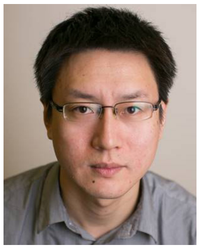
[32] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. ACM SIGCOMM Workshop*, Oct. 2010, p. 19.

[33] *Open VSwitch*. Accessed: Apr. 2019. [Online]. Available: http://www.openvswitch.org/

[34] (2015). *Ryu SDN Framework*. [Online]. Available: http://osrg.github.io/ryu/

[35] B. Pfaff, B. Lantz, and B. Heller, *Openflow Switch Specification, Version 1.3.0*, Open Netw. Found., Menlo Park, CA, USA, Jun. 2012.

Jiayin Chen received the B.E. and M.S. degrees from the School of Electronics and Information Engineering, Harbin Institute of Technology, Harbin, China, in 2014 and 2016, respectively. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. Her research interests are in the area of vehicular networks and machine learning, with current focus on intelligent transport system and big data.



Qiang Ye (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2016, where he was a Postdoctoral Fellow with the Department of Electrical and Computer Engineering from December 2016 to November 2018, and he was a Research Associate from December 2018 to September 2019. He has been an Assistant Professor with the Department of Electrical and Computer Engineering and Technology, Minnesota State University, Mankato, MN, USA, since September 2019. His current research interests include 5G networks, software-defined networking and network function virtualization, network slicing, artificial intelligence and machine learning for future networking, protocol design, and end-to-end performance analysis for the Internet of Things.

Wei Quan received the Ph.D. degree in communication and information system from the Beijing University of Posts and Telecommunications, Beijing, China, in 2014. He is currently an Associate Professor with the School of Electronic and Information Engineering, Beijing Jiaotong University. He has published more than 20 papers in prestigious international journals and conferences including *IEEE Communications Magazine*, IEEE WIRELESS COMMUNICATIONS, IEEE NETWORK, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE COMMUNICATIONS LETTERS, IFIP Networking, IEEE ICC, and IEEE GLOBECOM. His research interests include key technologies for network analytics, future Internet, 5G networks, and vehicular networks. He serves as an Associate Editor for the *Journal of Internet Technology*, *Peer-to-Peer Networking and Applications*, and *IET Networks*, and as a Technical Reviewer for many important international journals. He is also a member of ACM and a Senior Member of the Chinese Association of Artificial Intelligence.

Si Yan (Student Member, IEEE) received the B.S. degree in electronic information engineering from Tianjin University in 2013, and the M.E. degree in electrical and computer engineering from the University of Calgary in 2016. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Waterloo. His current research interests include fifth-generation networks, software-defined networking, network function virtualization, and network protocol design.

Phu Thinh Do received the B.S. degree in electrical engineering from the Ho Chi Minh City University of Technology in 2010, and the M.S.E. and Ph.D. degrees from the Department of Electronics and Radio Engineering, Kyung Hee University, South Korea, in 2016. From 2016 to 2019, he was a first Postdoctoral Fellow with the Digital Communication Lab, Kyung Hee University, then the Broadband Communications Research Lab, University of Waterloo, Canada. He is currently a Researcher with the Institute of Research and Development, Duy Tan University, Vietnam. His current research interests include 5G wireless communications, search engines, and recommendation systems for e-commerce platforms.

Peng Yang (Member, IEEE) received the B.E. degree in communication engineering and the Ph.D. degree in information and communication engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2013 and 2018, respectively. Since September 2018, he has been working as a Postdoctoral Fellow with the BBCR Group, Department of Electrical and Computer Engineering, University of Waterloo, Canada, where he was a visiting Ph.D. student from September 2015 to September 2017. His current research focuses on software defined networking, mobile edge computing, and video streaming.

Weihua Zhuang (Fellow, IEEE) has been with the Department of Electrical and Computer Engineering, University of Waterloo, Canada, since 1993, where she is currently a Professor and a Tier I Canada Research Chair of wireless communication networks. She was a recipient of the 2017 Technical Recognition Award from the IEEE Communications Society Ad Hoc and Sensor Networks Technical Committee, and several Best Paper Awards from IEEE conferences. She was the Technical Program Symposia Chair of the IEEE GLOBECOM 2011, and the Technical Program Chair/Co-Chair of the IEEE VTC, in 2016 and 2017. She is a fellow of the Royal Society of Canada, the Canadian Academy of Engineering, and the Engineering Institute of Canada. She is an Elected Member of the Board of Governors and VP Publications of the IEEE Vehicular Technology Society. She was an IEEE Communications Society Distinguished Lecturer from 2008 to 2011. She was the Editor-in-Chief of the IEEE Transactions on Vehicular Technology from 2007 to 2013.

Xuemin (Sherman) Shen (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990.

He is currently a University Professor with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research focuses on network resource management, wireless network security, social networks, 5G and beyond, and vehicular ad hoc, and sensor networks. He received the R.A. Fessenden Award in 2019 from IEEE, Canada, James Evans Avant Garde Award in 2018 from the IEEE Vehicular Technology Society, Joseph LoCicero Award in 2015 and Education Award in 2017 from the IEEE Communications Society. He has also received the Excellent Graduate Supervision Award in 2006 and Outstanding Performance Award 5 times from the University of Waterloo and the Premier's Research Excellence Award in 2003 from the Province of Ontario, Canada. He served as the Technical Program Committee Chair/Co-Chair for the IEEE Globecom'16, the IEEE Infocom'14, the IEEE VTC'10 Fall, the IEEE Globecom'07, the Symposia Chair for the IEEE ICC'10, the Tutorial Chair for the IEEE VTC'11 Spring, and the Chair for the IEEE Communications Society Technical Committee on Wireless Communications. He is the Editor-in-Chief of the IEEE Internet of Things Journal and the Vice President on Publications of the IEEE Communications Society. He is a Registered Professional Engineer of Ontario, Canada, an Engineering Institute of Canada Fellow, a Canadian Academy of Engineering Fellow, a Royal Society of Canada Fellow, a Chinese Academy of Engineering Foreign Fellow, and a Distinguished Lecturer of the IEEE Vehicular Technology Society and Communications Society.

Xu Li received the B.Sc. degree in computer science from Jilin University, China, in 1998, the MS.c degree in computer science from the University of Ottawa, in 2005, and the Ph.D. degree in computer science from Carleton University, in 2008. He worked as a Research Scientist (with tenure), Inria, France. He is a Senior Principal Researcher with Huawei Technologies Canada. He contributed extensively to the development of 3GPP 5G standards through over 90 standard proposals. He has published over 100 refereed scientific papers and is holding over 30 issued U.S. patents. His current research interests are focused in 5G. He is/was on the editorial boards of *IEEE Communications Magazine*, the IEEE Transactions on Parallel and Distributed Systems, the *Wiley Transactions on Emerging Telecommunications Technologies*, and a number of other international archive journals.

Jaya Rao received the B.S. and M.S. degrees in electrical engineering from the University of Buffalo, New York, in 2001 and 2004, respectively, and the Ph.D. degree from the University of Calgary, Canada, in 2014. Since 2014, he has been with Huawei Technologies Canada, Ottawa, and worked on research and design of CIoT, URLLC, and V2X-based solutions in 5G New Radio, where he is currently a Senior Research Engineer. He has contributed for Huawei at 3GPP RAN WG2, RAN WG3, and SA2 meetings on topics related to URLLC, network slicing, mobility management, and session management. From 2004 to 2010, he was a Research Engineer with Motorola Inc. He was a recipient of the Best Paper Award at IEEE WCNC 2014.