IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 8 September 2023, accepted 14 September 2023, date of publication 22 September 2023,
date of current version 29 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3318479

RESEARCH ARTICLE

Scalable Flow Isolation With Work Conserving Stateless Core Fair Queuing for Deterministic Networking

JINOO JOUNG[1], JUHYEOK KWON[2], JEONG-DONG RYOO[3,4], AND TAESIK CHEUNG[3]
[1]Department of Human-Centered Artificial Intelligence, Sangmyung University, Seoul 03016, South Korea
[2]Department of Artificial Intelligence and Informatics, Sangmyung University, Seoul 03016, South Korea
[3]Electronics and Telecommunications Research Institute (ETRI), Daejeon 34129, South Korea
[4]Information and Communication Engineering Major, University of Science and Technology, Daejeon 34113, South Korea

Corresponding author: Jeong-Dong Ryoo (ryoo@etri.re.kr)

This work was supported by the Electronics and Telecommunications Research Institute (ETRI) Grant funded by the Information and Communication Technology (ICT) Research and Development Program of Korean Government [Ministry of Science and ICT (MSIT)/Institute for Information and Communication Technology Planning and Evaluation (IITP)], Development of End-to-End Ultra-High Precision Network Technologies, under Grant 2021-0-00715.

ABSTRACT We consider guaranteeing end-to-end (E2E) latency bounds to flows in a network. It is desirable that flows are isolated from other flows. The bursts from other flows or the network utilization level should not affect a flow's latency bound. The fair queuing technique, which includes Packetized Generalized Processor Sharing (PGPS) and Virtual Clock (VC), is based on the concept of ideal packet service completion time called the Finish Time (FT). The fair queuing is known to provide the near perfect flow isolation but has to maintain the flow states. Alternative schemes were suggested, in which the entrance node in a network generates FT for a packet and records it in the packet with other necessary information. Subsequent nodes, based on these records, decide the service eligibility and service order of the packets. A packet is served only when it is eligible, thus the system is non-work conserving. In this paper, a simpler framework for deriving such FTs in core nodes without flow state is presented, in which initial FT is updated by adding a delay factor per node, which is a function of parameters of nodes and flow. The proposed scheduler is work conserving and has the property that, for a certain choice of the delay factor, the expression for E2E latency bound can be found. This E2E latency bound function is the same as that of a network with stateful fair queuing schedulers in all the nodes. Moreover, utilizing the fact that the service order of packets passing through the same path can be unaltered in the middle of the path, we also present a FIFO-based architecture whose performance is similar to that of a priority queue-based architecture. The extensive simulations prove that the proposed framework shows an ideal flow isolation performance over a wide range of delay factor values, with superior scalability.

INDEX TERMS Finish time, deterministic networking, latency bound, scheduler, fair queuing.

I. INTRODUCTION

There are emerging applications that require both latency and latency variation (jitter) bounds in large-scale networks [1]. In environments such as smart factories and in-vehicle networks, there is a demand for a network service that guarantees the upper bound of latency and jitter. Accordingly, standardization of related technologies has progressed in the IEEE 802.1 time-sensitive networking (TSN) task group [2], [3]. This demand has also arisen in large-scale networks such as Metaverse, and the IETF deterministic networking (DetNet) working group is also dealing with issues in higher layers such as IP/MPLS and issues in large-scale networks [4].

In large-scale networks, end-nodes join and leave, and a large number of flows are dynamically generated and terminated. Achieving satisfactory deterministic performance in such environments would be challenging. The current Internet, which has adopted the differentiated services (DiffServ)

The associate editor coordinating the review of this manuscript and approving it for publication was Guangjie Han.

architecture [5], has the problem of the burst accumulation and the cyclic dependency, which is mainly due to FIFO queuing and strict priority scheduling. Cyclic dependency is defined as a situation wherein the graph of interference between flow paths has cycles. The existence of such cyclic dependencies makes the proof of determinism a much more challenging issue and can lead to system instability, that is, unbounded delays [6].

A class of schedulers called Fair Queuing (FQ) limits interference between flows to the degree of maximum packet size. Packetized generalized processor sharing (PGPS) and weighted fair queuing (WFQ) are representative examples of FQ [7]. In FQ, the ideal service completion time, called Finish Time (FT), of a packet is obtained from an imaginary system, which can provide the ideal flow isolation. Packets in the queue are served in an increasing order of the FT. Then the service of the packet is completed approximately the same as the FT. When this technique is applied, the worst delay at each node is proportional to the maximum packet length among flows sharing the link. However, the FT of the previous packet within a flow has to be remembered for the calculation of the current packet's FT. This information can be seen as the flow state. The complexity of managing such information for a large number of flows is a problem, so it is not usually adopted in practice.

The objective of this work is to provide a framework, which is scalable and work conserving, for latency guarantee through flow isolation. In this paper, the following four novel contributions are provided.

- A framework for deriving FT at core nodes is proposed, based on the initial FT value, flow intrinsic parameters, and nodal information. This framework yields a work conserving scheduler. The initial FT calculated at the entrance node and the "delay factors" at intermediate nodes are accumulated to determine the packets' FT at core nodes.
- It has proved that the scheduler of the proposed method can have a mathematical expression for the end-to-end (E2E) latency upper bound, under condition that the node specific delay factor function to be accumulated is carefully selected. Moreover, this E2E latency bound function is the same as that of a *FQ network*, which is defined as a network having stateful fair queuing schedulers in all the nodes.
- It is also proposed a first in first out (FIFO) based architecture whose performance is similar to that of a priority queue-based architecture. This is based on the fact that the service order of packets of all the flows passing through the same path does not change much in the middle of the path. The packets from flows with similar characteristics can be put into a FIFO queue. The packet to be served can be easily decided by comparing only the FTs of packets at head of queue (HoQ) of the FIFO queues. We have shown that this simple FIFO implementation can achieve a similar maximum E2E latency to that of a FQ network.
- It is shown that, over a wide range of delay factor functions, including a simple fixed value such as the propagation delay of a link, the proposed framework provides a maximum E2E latency similar to that of a stateful FQ network.

The structure of this paper is as follows. Section II summarizes the overview of the scheduling algorithms. Section III examines the existing studies related to DetNet technology and the standardization trend. Section IV presents a framework that derives the FT values at core nodes based on the initial FT, intrinsic flow parameters, and nodal information, without state management for each flow. In this framework, a new FT of a packet at a core node is decided by adding the FT value in the previous node plus the delay factor. In Section V, various nodal delay factor values are presented and examined. Their worst E2E latency performances are compared with the existing technologies in a moderately sized network with cycles in topology. It is shown that the proposed framework can achieve superior E2E latency bounds in a broad range of delay factor values while securing the same level of scalability comparable to the simple priority-based scheduling techniques. Section V describes the conclusion and future research directions.

II. BACKGROUND

In this section we summarize the overview of the scheduling techniques, which has been studied for many years in the name of quality of service provisioned to various types of traffic.

One of the keys to the latency and jitter performance of a network is the packet scheduling technique. Packet scheduling refers to an algorithm that decides when the packets in an output port leave the system. Packets usually have variable length. The objective of scheduling can be diverse. The two most important performance metrics for scheduling algorithms are throughput and fairness. In a deterministic networking environment, a flow requests a service specification and traffic specification. A service specification specifies the required maximum end-to-end latency and jitter. A traffic specification specifies the maximum burst size, maximum packet length, and average arrival rate of the flow. In order to meet such requirements for flows with such different characteristics, it is not enough to just serve packets indiscriminately. Some schedulers do not serve packets even when the output link idles. They are called non-work conserving schedulers. Regulators, which do not serve packets until they are allowed, are good examples of the non-work conserving schedulers. A time-division multiplexer is another example of non-work conserving scheduler, in which a packet is sent only in an allowed slot.

On the other hand, a majority of packet schedulers are work conserving. They transmit packets whenever there are packets. In this paper we focus on the work conserving schedulers, since they have superior average throughput. Note that any two schedulers have an identical throughput, only if they are both work conserving. In deterministic networking, it is

required to guarantee various service specifications of flows. Therefore, the flows have to be differentiated and isolated from other flows.

Our objective is to devise a scheduler that can completely isolate a flow from other flows sharing a link. For an ideal flow isolation, a scheduler would provide service to a flow, as if there is no other flow in an imaginary link whose capacity is equal to the allocated service rate to the flow. In this case the latency upper bound is a function of the parameters, which depend only on the flow.

There are mainly three groups of work conserving packet schedulers, according to their levels of flow isolation. The first is the FQ technique. This technique limits interference between flows to the degree of maximum packet size. FT of a packet represents the time that an ideal fluid system would complete its service of the packet. In a real packetized FQ system, the packets in the queue are served in the increasing order of the FT. The FT is recorded in the packet. Then the service of the packet is completed approximately the same as the FT. When this technique is applied, the worst delay at each node is proportional to the maximum packet length among flows sharing the link. However, in this method, FT of the previous packet within a flow has to be remembered for the calculation of the current packet. This information can be seen as the flow state. The complexity of managing such information for a large number of flows is a problem, so it is not usually adopted in practice.

The second is the round-robin based technique that maintains only the rough information of flow states. Based on the service history of the flows, the service order of the flows is determined. This technique includes weighted round robin (WRR) and deficit round robin (DRR) [8]. When such a technique is applied, the worst delay at each node is proportional to the sum of the maximum packet sizes of all the flows.

The third technique is to store flows in a queue grouped by priority and simply to serve the queues according to the priority. The scheduling method adopted in Differentiated Services (DiffServ) is a representative example of the third technique [5]. When this technique is applied, the worst delay is determined by the sum of the maximum bursts of the flows. The maximum burst of a flow means the total amount of data that the source can send at once as allowed for that flow. In general, it represents a bundle of a large number of packets, and therefore it is considerably larger than the packet length. Therefore, the worst delay proportional to their sum may be an unacceptable level in some cases. Moreover, in a network to which this technique is applied, if there is a cycle in the topology, the maximum burst size of a flow exponentially grows as the flow passes through the nodes. There can be a situation where the worst delay cannot be guaranteed depending on the level of network utilization [6]. The phenomenon of burst accumulation is illustrated in Figure 1. When bursts from input ports collide, then the flow under observation, whose burst arrived a little later than the other bursts, can experience burst accumulation with a FIFO scheduler. A larger blue box represents the burst of the flow under observation. The size of the blue box increases as the it departs from a FIFO scheduler. The burst with FQ scheduler, on the other hand, is dispersed evenly.

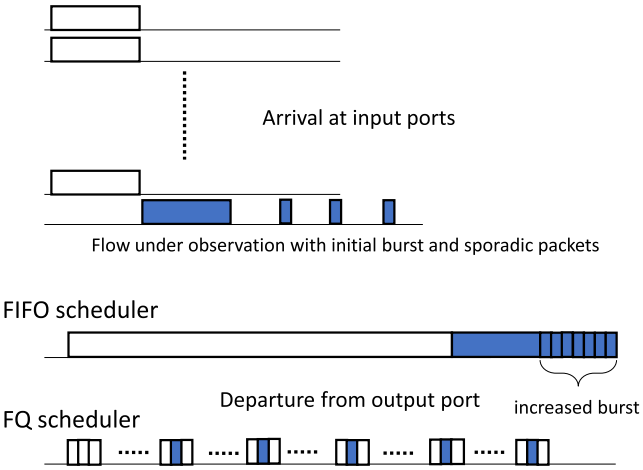

FIGURE 1. Illustration of burst accumulation with FIFO scheduler. In contrast, FQ schedulers can evenly disperse the packets over time according to the service rate of a flow.

The level of flow isolation is inversely proportional to the latency performance as we have described so far. However, the higher the isolation level, the more complex to implement. The fair queuing technique of the first type is preferable, but it has been impossible to manage the states of millions of flows in real time in core nodes of large-scale networks. To solve this problem, Core-Jitter Virtual Clock (CJVC) fair queuing [9] was proposed, which enables isolation between flows by updating FT based only on nodal parameters and initial FT information as needed [9], [10]. The node at the edge of the network creates state information for each packet, including FT and other necessary information, and records them in the packet. Subsequent core nodes infer the exact flow states at the node based on these records without managing flow state information. However, CJVC mandates a packet to wait until an eligible time for a service start, thus a non-work conserving service. The non-work conserving schedulers can suffer from a worse statistical performance.

III. RELATED WORKS

In this section we summarize the prior works related to deterministic networking and the approach presented in this paper.

A. STANDARDIZATION IN DETERMINISTIC NETWORKS

For small networks such as in-car networks or 5G fronthaul networks, the IEEE 802.1 TSN task group defines a set of solutions for latency and jitter minimization. TSN's packet forwarding technology can be largely divided into synchronous and asynchronous technologies. In synchronous technology, the solutions rely on the time synchronization of every node in the network and slot scheduling. For scheduling and allocating slots for the flows, coordination among nodes throughout the network is required. The TSN

task group has standardized multiple functional components for jitter-sensitive services based on time synchronization. Among them, the IEEE 802.1Qbv time aware shaping (also known as the time-aware shaper, TAS) [11], and IEEE 802.1Qch cyclic queuing and forwarding (CQF) [12] are built for jitter minimization as well as latency guarantee. Here, time is divided into appropriately sized slots, and these slots are aggregated to form a frame. The frame is repeated with a fixed length. Transmittable flows are specified for each slot. It exhibits characteristics similar to those of TDMA. In this method, 1) the size of the slot should be determined, 2) the problem of how to allocate the flows to the slots should be solved, and 3) the source should generate packets only in the allocated slots. If these limiting factors are solved, latency and jitter bounds can be guaranteed. The latency bound is proportional to the slot size. If the slot size is reduced, the latency is reduced, but the problem of slot allocation becomes complicated and real-time allocation can be limited to a small number of flows. If the slot size is increased to make the slot scheduling easier, the waste of resources is severe, and the worst delay also increases proportionally. Moreover, this mechanism requires every node in the network to be synchronized and requires collaboration. In a large-scale network, this method is too expensive and inappropriate.

TSN's asynchronous technology uses a regulator to prevent burst accumulation of packets. Accumulation of packet bursts can occur in the work conserving scheduler, which does not idle when there are packets in the queue. Mitigating the accumulation with the regulation function has been proposed for more than two decades. However, it has not been widely used due to the complexity of determining when to send packets by placing a queue for each flow in the core node. TSN has solved this complexity to some extent by adopting an interleaved regulator (IR) under the name of asynchronous traffic shaping (ATS) [13], [14]. However, if we have to regulate, it has to be non-work conserving, thus the efficiency by statistical multiplexing is inevitably reduced, which is the most important success factor of the Internet. The regulator function of ATS is also disadvantageous in terms of average latency.

The IETF DetNet WG is trying to solve the problem of guaranteeing latency in a large-scale network in three directions.

The first is the extension of TSN synchronous technology. The slot size is kept the same, but the start and end times are not synchronized across the nodes. In this way, it is claimed that the problem of propagation delay and dead time can be solved. However, it is necessary to specify the slot in which the packet should be serviced as tag information between adjacent nodes, so it is called Tagged CQF. The biggest disadvantage of this method is that it is difficult to use in a network that dynamically changes in real time because it involves considerable complexity in determining the slot size and scheduling slots to flows. The problem of slot scheduling in TSN remains the same.

The second is the extension of the ATS of TSN asynchronous technology. It divides the network into appropriately sized domains and places IRs between domains to avoid burst accumulation. By grouping and processing flows having the same path within the domain together, the complexity of flow-based processing technology is largely offset. This technology has also been approved as a standard by ITU-T study group (SG) 13 [15], [16].

The third is a group of technologies that record various types of meta-data in a packet and determine a forwarding policy based on them. The first type of meta-data is directly related to the latency. The accumulated latency from network entering up to the current time, the latency bound, and the latency budget, which is a value obtained by subtracting them, are being mentioned. However, it is yet to be proved that this latency related meta-data alone can guarantee the latency upper bound. The second type of meta-data is the FT and related information, which are the ones discussed in this paper. This method can guarantee the highest flow isolation performance thus the smallest latency bound. This meta-data-based network operation technology has recently been spotlighted for various purposes, so it is worth paying attention. For example, the segment routing technique, which uses packet meta-data, allows the end-host or edge router to determine the path of the flow, and also to determine the forwarding function as suggested by the standards in the IETF, thus actively reflecting the user's intention and providing more detailed service differentiation seems possible.

ITU-T SG13 has approved, as of September 2023, four Recommendations on deterministic networking services; Y.3113, Y.3118, Y.3120, and Y.3121. ITU-T Y.3113 and Y.3120 specify the requirements, framework, functional architecture, and operational procedures for latency guarantee in large-scale networks. Y.3118 specifies the requirements and framework for jitter guarantee in large-scale networks. Y.3121 specifies the requirement and framework for deterministic networking services in heterogeneous forwarding technology domains interwork. Y.3113 and Y.3120 define the use of the flow aggregate and regulators to be used over aggregation domains. Recently, the usage of meta-data of FT and related information, similar to the ones proposed in IETF DetNet WG, is also proposed in ITU-T SG13.

B. FAIR QUEUING SCHEDULERS

Since the 1990s, a technology that guarantees the requested service rates to all the flows based on an ideal fluid-based scheduling model has been developed. A flow is a set of packets belonging to the same application with the same source and destination. Generalized processor sharing (GPS) suggested a paradigm for a fair service for flows as fluid. Packetized GPS (PGPS or Weighted fair queuing), which implemented GPS in the realistic packet-based environment, played a pioneering role in this type of packet-based schedulers [7]. PGPS determines the service order of packets in

ascending order of the FT derived by the following equation.

$$F(p) = \max\{F(p-1), V(A(p))\} + L(p)/r, \quad (1)$$

where p and $p-1$ are the pth and (p-1)th packet of a flow, $F(p)$ is the FT, $A(p)$ is the arrival time, $L(p)$ is the length of the packet p, and r is the service rate guaranteed to the flow. Note that the index for the flow i is omitted. $V(t)$ is called the virtual time function [7] or a system potential [17] and is a value representing the current system progress at time t. If the backlogged flows almost fill the link capacity, then the system slowly progresses in terms of a flow's view, and so does the virtual time. If there is only a handful of backlogged flows, then the virtual time increases with a higher rate. It can be obtained, for example [7], by $V(0) = 0$ and $V(t_k + \tau) - V(t_k) = \tau / \sum_{j \in f(h,t)} r_j$, where $f(h, t_k)$ is the set of backlogged flows in node h at t_k, t_k is a moment of $f(h, t_k)$ element change, $\tau < t_{k+1} - t_k$, and r_j is the service rate of flow j. It prevents an unfair situation in which flows that entered late have relatively small FTs thus receive services earlier for a considerable period of time, compared to existing flows.

$F(p)$, the FT of p, represents the time that an ideal fluid system would complete its service of packet p. In a real packetized fair queuing system, the packets are served in the increasing order of the FT. The FT can be calculated at the moment the packet arrives in the node, so it can be recorded and used in the node in the form of meta-data of the packet before it is stored in the buffer. In general, there is a queue for each flow, the queues are managed by FIFO manner, and the scheduler serves the queue having the HoQ packet with the smallest FT. Alternatively, it is possible to put all packets in one queue and sort them during enqueue and dequeue processes according to the value of the FT. This implementation requires a priority queue. The point of (1) is that, in the worst case, when all flows are active and the link is fully used, a flow is served at an interval of $L(p)/r$. At the same time, by using the work conserving scheduler, any excessive link resources are shared among the flows. How fair it is shared is the main difference of various fair queuing schemes.

In order to obtain (1), $F(p-1)$ of the flow, the FT of the previous packet of the flow must be remembered. When a packet is received, it is necessary to find out which flow it belongs to and find out the FT of the latest packet of the corresponding flow. $F(p-1)$ of this latest packet is a value representative of the so-called 'flow state'. The fact that such state information must be memorized and read means a considerable complexity at a core node managing millions of flows. This is the main reason why such fair queuing schedulers are not actually used on the Internet. In this paper, we pay attention to the fact that the fair service time interval information between packets is already included in the FTs of the entrance node of a flow. Instead of deriving a new FT at each core node, we propose a method of deriving the FT at downstream nodes by using the FT information calculated at the entrance node.

On the other hand, calculating $V(t)$ also involves the complexity of calculating the sum of r by tracking the flows currently being serviced in real time. It must be a fairly difficult calculation, considering that the beginning of flows can be countless in any short time period. Therefore, instead of calculating $V(t)$ accurately, methods for estimating it in a simple way have been suggested [18], [19], [20]. Among them, Virtual clock (VC) [18] uses the current time t instead of $V(t)$ to determine the FT. Self-clocked fair queuing [19] uses the FT of the recently serviced packet of another flow instead of $V(t)$.

Stiliadis et al. showed that this series of FQ schedulers can belong to the rate-proportional server (RPS) group and can be implemented with its packet-based version, packetized RPS (PRPS) [17]. For example, PGPS and VC are PRPS, while self-clocked fair queuing is not. Furthermore, it was proved that a network with all the nodes having one of these PRPSs guarantee the E2E latency D_i for flow i as follows.

$$D_i \leq \frac{B_i - L_i}{r_i} + \sum_{h=0}^{H} SL_i^h, \quad (2)$$

where B_i and r_i are the maximum burst size and service rate of the flow i, L_i is the maximum packet length of flow i, and SL_i^h is the *service latency* of flow i at the node h. H is the last node for the flow i to traverse.

The service latency, SL, in a node that is applicable to every PRPS is expressed as in (3). The service latency was first introduced in the concept of the latency-rate server model [21], which can be interpreted as the worst delay until the first packet of a newly arrived flow can be served. L_{max} is the maximum packet length over all flows in the server,, and R is the link capacity of the node h.

$$SL_i^h = \frac{L_i}{r_i} + \frac{L_{max}}{R}. \quad (3)$$

C. CORE-STATELESS FAIR QUEUEING

The FQ scheduling technique stores state information for each flow, extracts this information when a packet of the corresponding flow arrives, calculates the FT of the packet, and stores it again. If $F(p-1)$ in (1) represents state information, the newly calculated $F(p)$ becomes the updated state of the flow. This complexity is the reason why it is difficult to implement FT-based scheduling in a core node where millions of flows are active simultaneously. Therefore, if $F_h(p)$ is the FT of p at the hth node in the path, methods of estimating $F_h(p)$ with $F_0(p)$ instead of $F_h(p-1)$ have been proposed. These methods are collectively called core-stateless fair queuing (CSFQ) [9], [10]. In particular, in the study of Stoica et. al. [9], the seminal work of similar studies, $F_h(p)$ is derived in the following way. First, $E_h(p)$ is defined as the eligible time of the packet p to start the service. Each packet becomes a candidate for service after the eligible time has elapsed. The packet having the lowest $F_h(p)$ value among these candidates is served.

$$E_0(p) = A_0(p),$$

$$E_h(p) = A_h(p) + G_{h-1}(p) + \delta_h(p),$$
$$F_h(p) = E_h(p) + \frac{L(p)}{r}. \quad (4)$$

Here, $G_h(p)$ is the delay margin at h (the difference between the FT and the actual service completion time), and $\delta_h(p)$ is the delay that forces $E_h(p)$ to be always greater than $E_{h-1}(p)$. This term guarantees the service order preservation between packets in the flow. Consequently, $E_h(p)$ can be expressed as a function of $E_0(p)$. That is, if $G_h(p)$, r, $\delta_h(p)$, and $L(p)$ are stored in the header of the packet and sent to the next node, the eligible time and FT of (4) can be derived without managing the flow state information. This scheduling method, CJVC, is based on the VC [18], and is claimed to minimize jitter as well.

The key idea in (4) is to use $F_0(p)$ instead of $F_h(p-1)$ in (1) to obtain $F_h(p)$. For example, suppose a flow continues to send packets and is continuously backlogged on the entrance node. In this case, $F_0(p)$ is calculated as $F_0(p-1)+L(p)/r$. That is, the interval between the consecutive packets' FTs is maintained at $L(p)/r$. These characteristics are independent of the states of other flows. That is, the FT of a packet can be inferred, ahead of traveling to the downstream nodes, only from the packet lengths and the assigned service rate of the flow as $L(p)/r$ during the backlogged period. Even in the downstream nodes, during the backlogged periods, we can predict $F_h(p) = F_{h-1}(p) + d_h(p)$, where $d_h(p)$ is some delay function specific to the node and the packet. Note that $d_h(p)$ may not be a function of p, but of p's flow. In this case, it can maintain the intervals between packets' FTs identical to those of $F_0(p)$.

Comparing (4) with (1), we can see the following facts about the CJVC.

- A packet cannot be serviced until its eligible time is reached. It is a non-work conserving scheduler.
- With $G_h(p)$, the delay variation between packets is minimized.
- The service order between packets within a flow is preserved. However, small packets from other flows traveling in the same path can catch large packets and be serviced earlier.

Regardless of these properties, the CJVC is claimed to have the same E2E delay bound as that of PGPS [9]. However, we will show the inefficiency of the CJVC when compared to the proposed work conserving fair queuing scheduler in terms of both the maximum and the average E2E latency, with simulations in Section V-G. It will be also shown that the jitter of this non-work conserving scheduler is not better than the one of the proposed work conserving scheduler.

IV. WORK CONSERVING STATELESS CORE FAIR QUEUING (C-SCORE)

In this section we present the framework for the stateless fair queuing in core nodes, the requirements for existence of E2E latency bound, and the practical considerations for implementations.

A. FRAMEWORK FOR FINISH TIME DETERMINATION IN CORE NODES

The basic assumptions in this paper on the traffic and network events are as follows.

- All the flows conform to their traffic specification (TSpec) parameters. In other words, with the maximum burst size B_i and the arrival rate r_i, the accumulated arrival from a flow i in any arbitrary time interval $(t_2 - t_1)$ does not exceed $B_i + (t_2 - t_1)\, r_i$.
- Actual allocated service rate to a flow can be larger than or equal to the flow's arrival rate.
- Total allocated service rate to all the flows in a node does not exceed the node's link capacity.
- In this paper, the service rate is considered to be the same as the arrival rate r_i. However, as it will be shown in Theorem 3 that, by adjusting the service rate to a flow, the E2E latency bound of the flow can be adjusted. Note that r_i is used interchangeably with the symbol r.
- A node, or equivalently a server, means an output port module of a switching device. It is also assumed that ideal non-blocking output-queued switches are used. Internal stages within a switch, which may cause additional latency, are not considered. Note that this assumption can affect the scalability, since the most complex core switching devices may not support ideal output-queued switching.
- The entrance node for a flow is the node located at the edge of a network, from which the flow enters into the network. A core node for a flow is a node in the network, which is traversed by the flow and is not the entrance node. Note that a single node can be both an entrance node to a flow and a core node for another flow.
- A packet is arrived, or serviced; when its last bit has arrived, or left the node, respectively.
- Propagation delays are neglected for the simplicity of representation. However, it can be easily incorporated into the equations in this paper, if necessary. This issue will be covered in Section IV-C.

Before going into the detailed description of the framework, let us summarize the symbols used in this section in Table 1.

It is preferred that the FTs at core nodes satisfy the following conditions. The flow index in the equations are omitted.

C1) Keep the fair distance between FTs of consecutive packets: $F_h(p) \geq F_h(p-1) + L(p)/r$.
C2) Preserve the FT orders: $F_h(p) \geq F_h(p-1)$. This condition is a subset of Condition 1.
C3) Reflect the time lapse as hops progress: $F_h(p) > F_{h-1}(p)$.

C1 is needed for the service provided to the flow does not exceed the allocated rate. C2 is the basic requirement for service order preservation within a flow. C3 is necessary in order for flows from different paths to be served fairly. In essence the three conditions are to ensure that (1) is closely approximated even in core nodes.

TABLE 1. Mathematical symbols used in this section.

Symbol	Quantity
p	p^{th} packet of the flow under observation.
h	h^{th} node in the path of the flow under observation
$F_h(p)$	finish time of packet p at node h
$A_h(p)$	arrival time of packet p at node h
$C_h(p)$	actual service completion time of packet p at node h
$L(p)$	length of packet p
L, L_i	max packet length of flow under observation
r, r_i	service rate of the flow under observation
L_h^{max}	max packet length over all the flows at node h
R_h	link capacity of node h
$d_h(p)$	delay factor function of packet p at node h
$U_h(p)$	latency upper bound of packet p's flow in node h
$W_h(p)$	latency lower bound of p's flow in node h
SL_h	*service latency* of the flow under observation at node h

In the following, we describe the proposed framework to obtain the FT in core nodes. First, at the flow entrance node 0, where the flow of interest arrives to the network, the FT is calculated in the same way as a fair queuing scheduler, for example the VC [18].

$$F_0(p) = \max\{F_0(p-1), A_0(p)\} + L(p)/r, \tag{5}$$

where $F_0(p)$ is the FT of p at the entrance node, and so on. $F_0(0) = 0$. Next, in a core node h,

$$F_h(p) = F_{h-1}(p) + d_h(p). \tag{6}$$

$d_h(p)$ is called the delay factor, which is a function of nodes and the packet. By using (5) and (6), only $F_{h-1}(p)$ and $d_h(p)$ values are required when calculating the FT in the core node. We call this framework the work conserving stateless core fair queuing (C-SCORE). Note that in most of the cases considered in this paper, $d_h(p)$ is a function of node $h-1$, not h. Therefore, $F_{h-1}(p)$ and $d_h(p)$ can be delivered from $h-1$ to h by storing them in the form of meta-data in the header of the packet, or be calculated based on the information carried as meta-data. An update is required at every node, but sufficient time would be given because the update is allowed at an appropriate time between the arrival to a switching device and the enqueuing to the output port queue. In some cases, $d_h(p)$ is only a function of the current node or the upstream node, not the function of packet p. Then it may be acknowledged by exchanging information through out-of-band communication between nodes. On the other hand, the different values of d_h in core nodes may affect the fairness between flows arriving at a core node through different paths. That is, the FTs of different flows should be aligned with each other to ensure fair service between flows.

Let us consider various choices for $d_h(p)$. For example, the first two conditions C1 and C2 can be achieved with the following simple way.

$$d_h(p) = U_{h-1}(p). \tag{7}$$

$U_h(p)$ is the latency upper bound of the p's flow in node h, such that

$$A_h(p) + U_h(p) \geq A_{h+1}(p).$$

Theorem 1: $F_h(p)$ obtained by (5), (6), and (7) meets the inequality: $F_h(p) \geq A_h(p) + \frac{L(p)}{r}$.

Proof: We prove it by induction over h. First, we show that it is satisfied with $F_0(p)$, and then we show that it is satisfied with $F_h(p)$ where $h > 0$. From (5), $F_0(p) = \max\{F_0(p-1), A_0(p)\} + \frac{L(p)}{r}$. Therefore

$$F_0(p) \geq A_0(p) + \frac{L(p)}{r}.$$

Now, for $F_h(p)$ with $h > 0$, assume that the theorem holds for $F_{h-1}(p)$. In other words, $F_{h-1}(p) \geq A_{h-1}(p) + \frac{L(p)}{r}$. From (6),

$$\begin{aligned} F_h(p) &= F_{h-1}(p) + d_h(p) \\ &\geq A_{h-1}(p) + \frac{L(p)}{r} + d_h(p). \end{aligned}$$

Let $U_{h-1}(p)$ be the latency upper bound of the p's flow in node $h-1$, i.e. $A_{h-1}(p) + U_{h-1}(p) \geq A_h(p)$. Then

$$\begin{aligned} F_h(p) &\geq A_{h-1}(p) + \frac{L(p)}{r} + d_h(p) \\ &\geq A_h(p) - U_{h-1}(p) + \frac{L(p)}{r} + d_h(p). \end{aligned}$$

From (7), $d_h(p) = U_{h-1}(p)$. The theorem holds for any $h > 0$, as well as for $h = 0$. ■

However, (7) can be disadvantageous to flows with larger numbers of hops. Another method for determining the delay factor function would be

$$d_h(p) = W_{h-1}(p), \tag{8}$$

where $W_h(p)$ is the latency lower bound of the p's flow in node h. It can be the propagation delay of the output link of h. (8) would be advantageous to the flows with larger numbers of hops. For end users who do not know the topology of the network or the number of hops needed for the flow, this treatment can be considered fair. It is to see the whole network as a single virtual node.

As a compromise in between $U_h(p)$ and $W_h(p)$, $d_h(p)$ can be the average latency within the node. However, in this case, in order not to flip the service order between the packets in a flow, the average latency used as the delay factor must be a non-decreasing function of time. When compared with (4), it can be seen that (7) or (8) may not increase the FT as much as $L(p)/r$ *at every node*. This reflects the fact that it may be fair to view the entire network as a single virtual node and maintain the initially set distances between packets' FTs. The exact value of $U_h(p)$ can be obtained, for example, by measurement.

Another advantage of (7) or (8) is that it maintains the initial FT order of packets within a flow, thereby minimizing

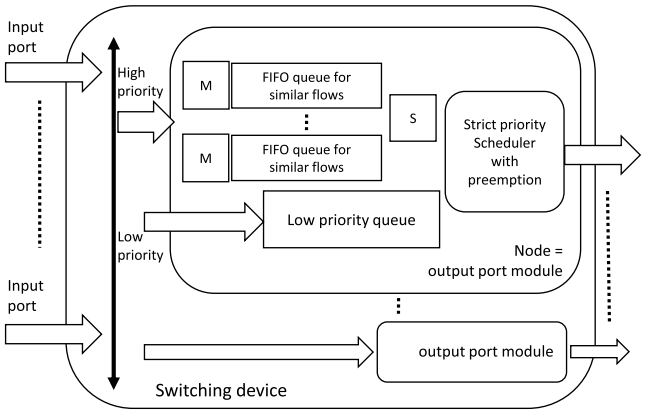

FIGURE 2. Example node architecture having each FIFO queue assigned for flows with similar characteristics. M obtains FT and writes it on the packet header. S examines the HoQ of FIFO queues and services the packet having minimum FT.

the chance of service order flips between packets in the same path in a core node. Note that the FT order within a flow is still maintained. Using this framework, the schedulers based on FT, which require a sorting algorithm [22], can be implemented with the FIFO queue with minimal performance degradation. We will see that this possibility can be fulfilled, especially when the queue is assigned according to the flows' characteristics, in Section V. Finding out the packet to service can be also performed by comparing only the FTs of the HoQs of the corresponding queues. With such a simple scheduler configuration, it is possible to secure scalability in the core node. Figure 2 shows an example of this implementation. All traffic is divided into high priority and low priority, and preemption is enabled. By adopting such an architecture, high priority traffic can be processed regardless of the presence or absence of low priority traffic.

B. E2E LATENCY GUARANTEE WITH THE PROPOSED FRAMEWORK

With a certain choice of the delay factor function, the E2E latency can be bounded. For the concrete understanding, let us start with the definitions for some terminologies. They mainly follow the definitions from [17] and [21].

Definition 1: A node busy period is a maximal interval of time during which the server has packets to send. During a node busy period a work conserving server is always transmitting packets.

Definition 2: A backlogged period for flow is any period of time during which packets belonging to that flow are continuously queued in the system.

Definition 3: A flow i busy period is a maximal interval of time such that at any time, t, within the interval $(t_0, t_1]$, the accumulated arrivals of the flow since the beginning of the interval do not fall below the total service received during the interval at exactly the allocated rate r_i, i.e. $r_i(t-t_0)$.

The flow busy period is defined only in terms of the arrival function and the allocated rate, and does not reflect the real system's instantaneous service variations to actual flows. Note that a busy period may contain intervals during which the actual backlog of flow in the system is zero and the flow is not receiving service.

Definition 4: An active period for flow is a maximal interval of time, over which the FT of its last packet is greater than the virtual time (equivalently the system potential). Any other period is an inactive period for the flow.

Definition 5: The first packet of an active period is an activation packet.

As pointed out in [17], a transition from inactive to active period can occur only by the arrival of a packet of a flow that is currently idle, as stated in Lemma 1.

Lemma 1: [17] If time T is the beginning of a flow busy period in an RPS, then T is also the beginning of an active period for flow.

When a flow is in an inactive period, it cannot be backlogged and, therefore, cannot be receiving service. On the other hand, an active period need not be the same as a backlogged period for the flow.

We assume for simplicity, that the propagation delay is zero for every link between nodes. Non-zero values of propagation delays can be easily incorporated. Define L_i as the maximum packet length of flow i. Define L_h^{max} as the maximum packet length over all the flows in node h. Also define R_h as the link capacity of the node h. We will show that when

$$d_h(p) = \frac{L_{h-1}^{max}}{R_{h-1}} + L_i/r_i, \tag{9}$$

the E2E latency of a flow is bounded. Let us call this expression on the right-hand side the *service latency* (*SL*) of the flow under observation i at node h-1, i.e.

$$SL_h = \frac{L_h^{max}}{R_h} + L_i/r_i.$$

This expression is indeed the service latency of a PGPS or VC server. The service latency was first introduced in the concept of the latency-rate server model [21], which can be interpreted as the worst delay until the first packet of a newly arrived flow to be served.

In the following, without loss of generality, we assume that the packet 0 is an activation packet of the active period during which packet p arrived at node 0.

Lemma 2: If (5), (6), and (9) are satisfied, then for $h > 0$,

$$F_h(p) = B(p)/r_i + A_0(0) + \sum_{k=0}^{h-1} SL_k,$$

where $B(p)$ is the sum of the packet lengths from packet 0 to packet p, i.e. $B(p) = \sum_{k=0}^{p} L(k)$.

Proof: The proof is straightforward. Applying (9) in nodes $1, 2, \ldots, h$, $F_h(p) = F_{h-1}(p) + SL_{h-1}(p) = F_0(p) + \sum_{k=0}^{h-1} SL_k$. By (5), the rule of FT assignment in a VC scheduler at node 0, $F_0(p) = \max\{F_0(p-1), A_0(p)\} + \frac{L(p)}{r_i}$. Because 0 is the activation packet of the active period in which p arrived at the node 0, $F_0(k-1) \geq A_0(k)$, for k, $0<k\leq p$. Therefore $F_0(p) = A_0(0) + B(p)/r_i$. $F_h(p) = F_0(p) + \sum_{k=0}^{h-1} SL_k = A_0(0) + B(p)/r_i + \sum_{k=0}^{h-1} SL_k$ follows. ■

Lemma 3: If (5), (6), and (9) are satisfied, then

$$F_h(p) \geq F_h(p-1) + L(p)/r_i,$$

for any p and $h > 0$.

Proof: From Lemma 2, for p > 0, $F_h(p) = F_0(p) + \sum_{k=0}^{h-1} SL_k = A_0(0) + B(p)/r_i + \sum_{k=0}^{h-1} SL_k = A_0(0) + B(p-1)/r_i + L(p)/r_i + \sum_{k=0}^{h-1} SL_k = F_{h-1}(p) + L(p)/r_i$. For p $= 0$, by the definition of an activation packet, $F_0(0) \geq F_0(-1) + L(0)/r_i$, where packet -1 is the previous packet of packet 0 in the flow. Thus $F_h(0) = F_0(0) + \sum_{k=0}^{h-1} SL_k \geq F_0(-1) + L(0)/r_i + \sum_{k=0}^{h-1} SL_k = F_0(-1) + L(0)/r_i$. We just proved that for p that is either activation packet or not, the inequality is satisfied. Lemma follows. ∎

Lemma 3 states that any packet including activation packets satisfies the inequality. Now consider an imaginary node with an ideal preemptive scheduler, h∗, whose input pattern and FT values of the packets are identical to the node h in our system. In this ideal preemptive node, the served portion of a packet before it is preempted is not served again. We call such a pair of systems are *equivalent* to each other. In h∗, a preempted packet can be infinitesimally divisible, such that it receives the remaining service without any overhead.

Let $C_h(p)$ be the actual service completion time of packet p at node h.

Lemma 4: If h is a work conserving non-preemptive scheduling node, which is *equivalent* to the work conserving preemptive scheduling node h∗, and the service order of packets is not changed dynamically, then $C_h(p) - C_{h*}(p) \leq \frac{L_h^{max}}{R_h}$.

Proof: Lemma 4 is stated and proved in Theorem 1 of [23]. It is also similar to Theorem 1 of [7]. ∎

Because both systems are work conserving, the node busy periods are identical. It is enough to show that within a single busy period, the inequality in Lemma 4 holds. The intuition behind the inequality can be understood with the following example. Just before a packet under observation's arrival, another maximum length packet with a larger FT from another flow arrives at an idle node. In a non-preemptive system, the packet under observation waits for $\frac{L_h^{max}}{R_h}$ until it starts to get service. In a preemptive system, however, the packet gets service immediately upon arrival. Therefore, the service start times in two systems, or the actual service completion times, differ as much as $\frac{L_h^{max}}{R_h}$ in the worst case. Any following packets within the busy period are sorted thus having the same actual service completion time in both systems.

Now we can state the bound of actual service completion time in a core node as the following.

Theorem 2: In a network which satisfies (5), (6), and (9), $C_h(p) \leq F_h(p) + \frac{L_h^{max}}{R_h}$.

Proof: The proof is by induction. ∎

For $h = 0$, the server is VC, and that $C_{h*}(p) \leq F_h(p)$ is proven in various literatures [7], [23]. Intuitively, since every flow is assigned the FT values according to its allocated rate, the sum of the allocated rates is less than the link capacity, and the scheduler is work conserving, therefore the VC scheduler guarantees the actual service completion time to be not greater than the FT. The Theorem follows from lemma 4, when $h = 0$.

For $h > 0$, we assume that the theorem holds from node 0 to $h-1$, i.e. $C_{h-1}(p) \leq F_{h-1}(p) + \frac{L_{h-1}^{max}}{R_{h-1}}$. Equivalently $A_h(p) \leq F_{h-1}(p) + \frac{L_{h-1}^{max}}{R_{h-1}}$, since we also assume zero propagation delay.

We first show that $C_{h*}(p) \leq F_h(p)$ for $h > 0$. By (9) and the induction hypothesis,

$$\begin{aligned} F_h(p) &= F_{h-1}(p) + SL_{h-1} \geq A_h(p) - \frac{L_{h-1}^{max}}{R_{h-1}} + SL_{h-1} \\ &= A_h(p) + L_i/r_i \geq A_h(p) + L(p)/r_i. \end{aligned} \tag{10}$$

Note that $F_h(p) = F_{h*}(p)$ and $A_h(p) = A_{h*}(p)$, regardless of the use of preemption.

Now consider an arbitrary time interval $[t_1, t_2]$. Consider a set of packets of flow i, P(t_1, t_2), which arrived in the interval $[t_1, t_2]$ and whose FTs are not greater than t_2. Now define a quantity $T_{h*}(p) = F_h(p) - \max(A_{h*}(p), F_h(p-1))$. If $A_{h*}(p) \leq F_h(p-1)$, from Lemma 3, $T_{h*}(p) \geq F_h(p-1) + \frac{L(p)}{r_i} - F_h(p-1) \geq \frac{L(p)}{r_i}$. If $A_{h*}(p) \geq F_h(p-1)$, from (10), $T_{h*}(p) \geq L(p)/r_i$. In either case $T_{h*}(p) \geq L(p)/r_i$. The function $T_{h*}(p)$ represents the time taken for p to be served in an ideal preemptive system, or equivalently the fair amount of time for p to be served. Further, define the function $r_{i,h*}(t) = r_i$ if there is a packet p in flow i, such that $A_{h*}(p) \leq t$ and $F_h(p-1) < t \leq F_h(p)$; else $r_{i,h*}(t) = 0$. $r_{i,h*}(t)$ represents the instantaneous service rate given to flow i in an ideal fluid system. Since $T_{h*}(p) \geq L(p)/r_i$,

$$\begin{aligned} \int_{t_1}^{t_2} r_{i,h*}(t) &= \sum\nolimits_{p \in P(t_1,t_2)} r_i * T_{h*}(p) \\ &\geq \sum\nolimits_{p \in P(t_1,t_2)} r_i * \frac{L(p)}{r_i} = \sum_{p \in P(t_1,t_2)} L(p). \end{aligned} \tag{11}$$

Therefore, the cumulative length of all the packets from flow i, which arrive in interval $[t_1, t_2]$ and whose FT not greater than t_2, is less than or equal to $\int_{t_1}^{t_2} r_{i,h*}(t)$. Intuitively, (11) states that any set of consecutive packets from a flow receive a greater amount of service ($\int_{t_1}^{t_2} r_{i,h*}(t)$) than what is required for them to be served before their FT ($\sum_{p \in \hat{o}P(t_1,t_2)} L(p)$), in a preemptive system.

Now we prove that $C_{h*}(p) \leq F_h(p)$ with contradiction. Assume that for packet p, $C_{h*}(p) > F_h(p)$. Also let t_0 be the start of the node busy period that packet p is served, and $t_2 = F_h(p)$. Let t_1 be the least time less than t_2 during the node busy period such that no packet with FT value greater than t_2 is served in the interval $[t_1, t_2]$. In other words, those packets that served later than t_1 has FT values less than $t_2 = F_h(p)$.

If no such t_1 exists, then p is the first packet of the busy period, or the first packet served in the busy period by preemption. In either case, the packet served immediately upon arrival, and $C_{h*}(p) = A_{h*}(p) + L(p)/R_h \leq F_h(p)$, which contradicts the assumption $C_{h*}(p) > F_h(p)$.

If such t_1 exists, all the packets served in the interval $[t_1, t_2]$ arrive in this interval have FTs less than or equal

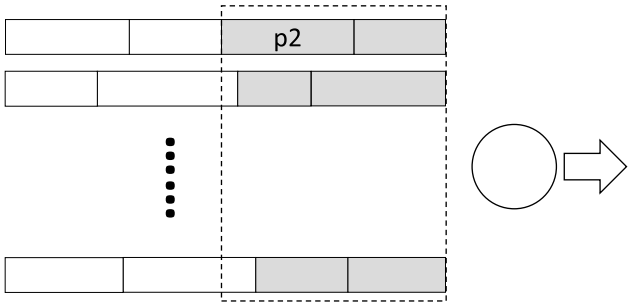

FIGURE 3. Illustration of inequality between $C_{h*}(p2)$ and $F_h(p2)$. Packets are queued per flow. The amount of bits in the dotted box ($F_h(p2)*R_h$) is larger than or equal to the amount of bits in the shaded packets ($C_{h*}(p2) * R_h$). Thus $F_h(p2) \geq C_{h*}(p2)$.

to $t_2 = F_h(p)$. Let us use the convention that j such that $j \in h$ represents any flow in node h. Since the server is busy in $[t_1, t_2]$ and packet p is not serviced by t_2 by the assumption $C_{h*}(p) > F_h(p)$. From (11), $\int_{t_1}^{t_2} \sum_{j \in h} r_{j,h*}(t) = \sum_{j \in h} \int_{t_1}^{t_2} r_{j,h*}(t) \geq \sum_{j \in h*} \sum_{p \in P(t_1,t_2)} L(p) \geq (t_2 - t_1) R_h$.

However, the link capacity is not exceeded therefore $\sum_{j \in h} r_{j,h*}(t) < R_h$, and $\int_{t_1}^{t_2} \sum_{j \in h} r_{j,h*}(t) < (t_2 - t_1) R_h$. This contradicts with each other, so the assumption $C_{h*}(p) > F_h(p)$ is proved wrong. Thus $C_{h*}(p) \leq F_h(p)$ is proved.

From Lemma 4 and that $C_{h*}(p) \leq F_h(p)$, the induction step is proved. The Theorem follows.

Theorem 2 can be understood by the following rationale. First, consider a preemptive network, in which every node serves packets with preemption. It is easy to understand that in an entrance node $C_{h*}(p) \leq F_h(p)$ is met. Note that the FT is calculated as if the flow under observation is the only flow served in an imaginary link which has the allocated service rate. However, in a real packet system, the link is shared with other flows whose sum of service rates is less than or equal to the link capacity. The remaining service rates $(C - \sum_i r_i)$ is shared among the flows. Therefore, the actual service completion time of a packet is always less than or equal to the FT. This can be further illustrated by an example in Figure 3.

In Figure 3, assume for simplicity, all the flows are allocated the same service rates and their bursts arrive at the node at the same time, 0. Also assume that the queues are assigned per flow. In the fluid model, the actual service completion time of p2 is $F_h(p2)$. During $[0, F_h(p2)]$, the bits served by the link are the bits in the dotted box, whose amount is $R_h * F_h(p2)$. In the preemptive packet system using FT, the actual service completion time of p2 is $C_{h*}(p2)$. Since all the packets are sorted, the sum of lengths of packets that have been served at $C_{h*}(p2)$ is the area of the shaded packets, $R_h * C_{h*}(p2)$, which are less than or equal to $R_h * F_h(p2)$. Thus, $C_{h*}(p2) \leq F_h(p2)$.

In core nodes, the FT is updated by adding the *service latency*, which is the worst delay a packet of a newly arriving flow to the node can experience. With this FT update, the actual service completion time of a packet is less than or equal to the FT. Intuitively, if this worst delay is added to FT, a packet which actually experienced the worst delay gets advantage compared to the other packets, since the difference between its FT and arrival time is smaller. It is also true that this service latency is the only choice for the FT increment value, which guarantees (11) is tightly met with equality.

Now consider a non-preemptive network. From Lemma 4, $C_h(p) - C_{h*}(p) \leq \frac{L_h^{max}}{R_h}$, and therefore, $C_h(p) \leq F_h(p) + \frac{L_h^{max}}{R_h}$.

In order to understand the meaning of the service latency, consider the worst case: Right before a new flow's first packet arrives at a node, the transmission of another packet with length L_h^{max} has just started. This packet takes the transmission delay of $\frac{L_h^{max}}{R_h}$. After the transmission of the packet with L_h^{max}, the flow under observation could take only the allocated share of the link, as in Figure 3 with all the flows fill up the link capacity, and the service of the packet under observation would be completed after L_i/r_i. Therefore, the packet has to wait, in the worst case, $\frac{L_h^{max}}{R_h} + L_i/r_i$.

The proof of Theorem 2 takes similar steps with the proof of Lemma 3 of [24]. However, Lemma 3 of [24] is not concrete in the sense that, in its proof, all the schedulers in the network have to be preemptive. In the meantime, Lemma 4 in this paper, which is also used in [24], considers only a single node. We have clarified this issue by considering Lemma 4 and the fact $C_{h*}(p) \leq F_h(p)$ at the same time in deriving Theorem 2.

The E2E latency bound in a C-SCORE network with (5), (6), and (9) is given as follows.

Theorem 3: The latency of flow i, in the network with C-SCORE and VC entrance nodes is bounded as

$$D_i \leq (B_i - L_i)/r_i + \sum_{k=0}^{H} SL_k,$$

where H is the last node of flow i.

Proof: From Theorem 2, $C_H(p) \leq F_H(p) + \frac{L_H^{max}}{R_H} = F_0(p) + \sum_{k=0}^{H-1} SL_k + \frac{L_H^{max}}{R_H}$. The E2E latency of p is $D_i(p) = C_H(p) - A_0(p) \leq F_0(p) + \sum_{k=0}^{H-1} SL_k + \frac{L_H^{max}}{R_H} - A_0(p) = F_0(p) - A_0(p) + \sum_{k=0}^{H-1} SL_k + \frac{L_H^{max}}{R_H}$. However, it is proved in Theorem 3 of [25] that the $(F_0(p) - A_0(p))$ of a VC node 0 is bounded by B_i/r_i. Intuitively, a burst of packets from a flow can arrive simultaneously, and the last packet of the burst can be assigned the FT of $B_i/r_i - A_0(p)$, at most. Therefore $D_i(p) \leq B_i/r_i + \sum_{k=0}^{H-1} SL_k + \frac{L_H^{max}}{R_H} = (B_i - L_i)/r_i + \sum_{k=0}^{H} SL_k$. ■

Note that the bound given in Theorem 3 is identical to that of a FQ network with stateful schedulers, e.g. PGPS or VC, in all the nodes.

One important observation on the E2E latency bound given in Theorem 3 is that the bound is the function on the flow intrinsic parameters, (B_i, L_i, r_i) and the ratios of the maximum packet length on the link to the link capacity

(L_h^{max}/R_h). Considering that R_h is much larger than r_i, the value of L_h^{max}/R_h is negligible. Therefore, the bound is almost adjustable by adjusting the flow's own parameters. A network can fulfill the requested E2E latency bound of a flow by allocating especially a proper service rate to the flow, regardless of the number of flows in the network. This argument cannot be met by other schedulers such as ATS in TSN, whose E2E latency bound is a function of the sum of other flows' initial maximum burst sizes.

C. CONSIDERATIONS FOR TIME DIFFERENCE BETWEEN NODES

As it has been stated in Section IV-A, we have assumed zero propagation delays between nodes. In reality, there are time differences between nodes, including the differences due to the propagation delays. This time difference can be defined as the difference between the actual service completion time measured at the upstream node and the arrival time measured at the current node. In other words,

$$td_{h-1,h}(p) = A_h(p) - C_{h-1}(p),$$

where $td_{h-1,h}(p)$ is the time difference between node h-1 and h, and $C_{h-1}(p)$ is the actual service completion time measured at node h-1, for packet p respectively. Note that FT does not need to be precise. It is used just to indicate the packet service order. Therefore, if we can assume that the propagation delay is constant and the clocks do not drift, then $td_{h-1,h}(p)$ can be simplified to a constant value, $td_{h-1,h}$. In this case the delay factor in (9) can be modified to be

$$d_h(p) = SL_{h-1} + td_{h-1,h}.$$

The E2E latency bound given in Theorem 3 increases as much as the sum of propagation delays from node 0 to h. Moreover, the time difference $td_{h-1,h}$ can be updated only once in a while, and signaled out-of-band. Even with the clock differences and propagation delays, C-SCORE does not need global time synchronization.

V. ANALYSIS OF VARIOUS C-SCORE SCHEMES

In this section the performance of C-SCORE with various choices of delay factors are examined through simulations in a medium sized network topology with different types of flows.

A. SIMULATION ENVIRONMENT

The simulations use SimPy, a Python-based process-based discrete-event simulation framework that provides various types of resources [26]. Since these resources are shared by all processes, a process detects and performs the defined action when the trigger for increase/decrease, release, and creation events of the resource operates through the generator function inherent in the process [26].

The network topology used for the simulations is shown in Figure 4. Nodes 1, 2, 3, 13, 14, and 15 are directly connected to the source. They are the entrance nodes, while the rest are core nodes. In C-SCORE, an output port of a core node has a queue for each input port, so there are two queues in an output port. These queues operate on a PIFO basis unless otherwise specified. The packets are sorted according to their FTs upon enqueuing. An entrance node maintains the state for each flow and the FIFO queue for each flow. The link capacity of all links in the topology is 1 Gbps. In Figure 4, arrows indicate the flow direction.

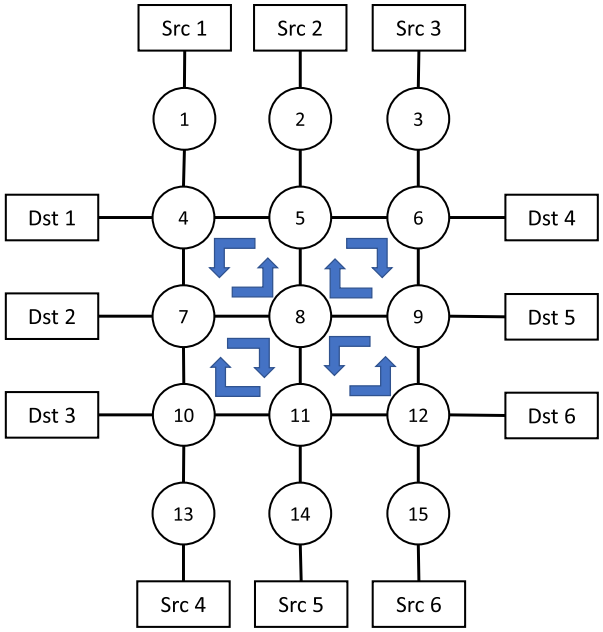

FIGURE 4. Network topology used in simulation.

A source creates one flow to each destination for a total of 6 flows. 36 flows are created throughout the network. Table 2 and Table 3 describe characteristics of the three different flow types used in the simulations. The destination of a flow decides the flow type. For example, all the flows destined to node 1 are of type A. There are 6 flows for each destination. There are 12 flows for each type. Flow types are classified with the maximum burst size, maximum packet length, and input rate of a flow. The flows generate packets of different lengths from 1K to 10Kbit, in units of 1Kbit.

TABLE 2. Characteristics of each flow type.

Flow type	Maximum burst size	Packet length	Destination
A	200Kbit	1K-10Kbit	1, 6
B	200Kbit	1K-10Kbit	3, 4
C	20Kbit	1K, 2Kbit	2, 5

Among the flows, the flows of interest are C type flows. The C type flows have a higher input rate, a smaller burst size, and a smaller packet size. They should prefer to be isolated from the other types. The A type flows have the opposite characteristics. They would experience better latencies when

TABLE 3. Peak input rate for each flow type according to utilization.

	Peak input rate [Mbps]		
Utilization	A	B	C
70%	9.857	98.571	98.571
75%	10.571	105.714	105.714
80%	11.262	112.619	112.619
85%	11.976	119.762	119.762
90%	12.667	126.667	126.667
95%	13.381	133.81	133.81

TABLE 4. Longest path of each flow type in the topology.

Flow type	Longest path
A	Src5-14-11-10-7-8-5-4-Dst1 Src2-2-5-6-9-8-11-12-Dst6
B	Src5-14-11-10-7-8-5-6-Dst4 Src2-2-5-6-9-8-11-10-Dst3
C	Src3-3-6-9-8-5-4-7-Dst2 Src6-15-12-9-8-5-4-7-Dst2 Src1-1-4-7-8-11-12-9-Dst5 Src4-13-19-7-8-11-12-9-Dst5

the flows are not isolated. We will compare the observed maximum E2E latencies of each flow type. The flows with longest paths within the same flow type are of interest. Table 4 shows the path of the flows with longest paths for each flow type. For all the flow types, the number of hops in the longest paths is the same. The utilizations on a path may differ for different links.

A probabilistic packet generation process is necessary for statistical performance observation. Therefore, we choose to use the combination of a Markov On-off model and a token bucket as the packet generation process. The source generates packets along the Markov on-off model independently for each flow. The Markov on-off model used is shown in Figure 5. The state changes every microsecond according to the transition rate specified in Figure 5. When the Markov on-off model for each flow is in the ON state, packets are generated according to the peak rate specified in Table 3. The packet generation rate over a long period of time, along the on and off states of the Markov model, is called the average input rate. Note that the Markov model generates packets probabilistically. Therefore, in a short period of time the flow may generate more packets than its TSpec allows. In order to guarantee a flow does not violate the TSpec constraints of the maximum burst and average input rate, the token bucket algorithm is used. The token size is set to 1. The token generation rate is the same as the peak input rate. Arrival time of a packet in an entrance node is the time it departs the token bucket. Token bucket is used to keep the average arrival rate of flow under the TSpec. It is a part of the packet generation process. The probability of being in the ON state in

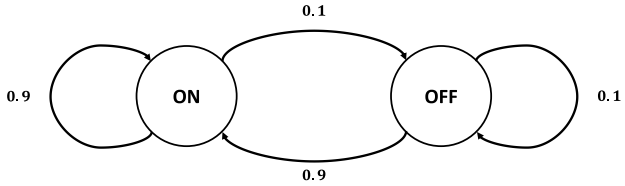

FIGURE 5. The Markov on-off model used in the source and the transition rate of the model.

the Markov on-off model is set at 90%, so that the difference between the average input rate and the peak input rate is not large.

The simulation covers the utilization levels from 70% to 95% with an increment of 5%. Utilization is computed based on the peak input rate. In our topology, the link between 5-6 and the link between 10-11 are the bottleneck. One type A flow, six type B flows, and one type C flow pass these links.

With C-SCORE, to sort packets based on their FT values, push in first out (PIFO) queues, or equivalently priority queues, are used. With a recent implementation of programmable PIFO switch, it is shown to be possible to sort the packets in specified order upon arrival up to 10 Gbps line-speed with 64 ports sharing a single shared memory [27]. However, note that [27] utilizes the fact that the packets within a flow are always served in FIFO manner. The maximum of 1000 flows are assumed in its implementation. Having more than 1000 flows likely results in a degraded performance. Another notable example of priority queue is the pipelined heap (P-heap) [28]. It showed that a priority queue is supported up to 15 Gbps, 2^{32} priority levels, for 53 Byte ATM cells, with 0.35 micro technology, which is translated into around 100 MHz clock. Moreover, the throughput of P-heap is independent of the queue size or the number of flows.

We have compared FIFO, ATS, WRR, DRR, VC and C-SCORE. The C-SCOREs in the comparisons may have different $d_h(p)$ values. FIFO is the basic scheduler used in DiffServ. It is well known that if the utilization is low and the sum of allocated rates to all the flows in a link is less than the link capacity, then the E2E latency can be bounded even with the FIFO. Thus the FIFO can be the simplest candidate for E2E latency guarantee. The ATS of IEEE TSN TG is a combination of per-input port interleaved-regulators and a FIFO scheduler. It prevents the accumulation of bursts over hops, and theoretically has a better latency bound than the FIFO. The DRR and WRR are used in real Internet deployments. They still require per-flow queues therefore per-flow state maintenance, however. The weight of WRR was determined as the ratio of the peak input rate. The quantum size of DRR was used by multiplying the weight used in WRR by 10K, the maximum packet length. The VC is the representative of all the stateful fair queuing schedulers, defined as the class of PRPS [17] by Stiliadis et al. The schedulers in PRPS class

have the identical E2E latency bounds, therefore it is enough to observe only the VC's E2E latency performance.

Each flow generates 1000 packets. We get E2E latency of a total of 36000 packets per simulation. The simulation was repeated 100 times with random seeds for each utilization. For each scheduler simulation, the traffic generation pattern must remain constant, so we used the same random seed. The maximum E2E latency for each flow type observed through each simulation was obtained as a result. These observed maximum E2E latencies are depicted with box plots.

B. NODE-SPECIFIC DELAY FACTOR

As the simplest way to determine $d_h(p)$ of C-SCORE, a function that depends only on node, d_h, can be used. In other words, the delay factor is a fixed value for an output port of a node. We call this a node-specific delay factor. Since all packets use the same d_h value in a node, there is no need to inform this value through a packet meta-data. Instead, d_h value can be negotiated and acknowledged among nodes in the control plane. In this case, only $F_h(p)$ is used as a packet meta-data, and there is no further information needed to be stored in the node.

This approach of selecting d_h as the delay factor clearly satisfies Condition 1 to 3 in Section IV-A. Condition 4 can also be satisfied if d_h value appropriately reflects the actual delay of the hop. We have examined the possibility of these node-specific delay factors through simulations.

Figure 6 is the graph of the maximum E2E latency of type C flows with varying d_h and utilization. In these simulations d_h is the same for all the nodes. Each plane of different color represents the maximum, average, and minimum of the observed maximum E2E latency. As d_h increases, the maximum E2E latency also increases. This is because among the same types, flows with larger numbers of hops determine the maximum E2E latency. Since d_h is a fixed value, the FT of packets with more hops becomes larger thus suffering more delay, regardless of the route. Moreover, this effect can be observed between the flows with different types. When utilization is high, this symptom is observed more clearly with larger d_h.

When d_h is 0, i.e., when d_h is the minimum value, packets generated earlier have an advantage. The FT of a packet that has traveled more hops gets advantage compared to a packet that has traveled less hops. Note that in this paper, the propagation delay of a link is assumed to be zero. In practical implementations, the minimum value of d_h should be equal to the propagation delay. In this case, the d_h should reflect the difference among the propagation delays of links.

We have compared the maximum E2E latencies of each flow type, of C-SCORE with the node-specific delay factor, with the other schedulers. Figure 7 depicts the distribution of the maximum E2E latency of each flow type for each scheduler at 90% utilization. For C-SCORE, $d_h = 0$.

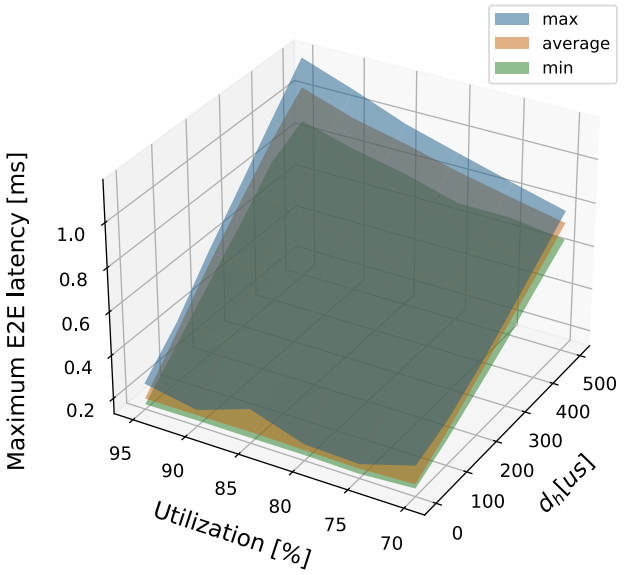

FIGURE 6. 3D surface plot of the maximum E2E latency of a type C flow in C-SCORE according to d_h and utilization.

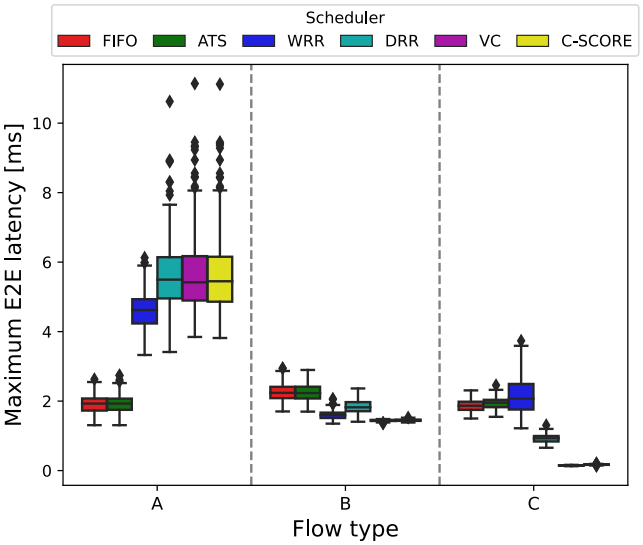

FIGURE 7. The maximum E2E latency distributions for each scheduler is shown when the utilization is at 90%, with d_h of C-SCORE being the minimum value. The order of boxplots is the same as the order in the legend.

In Figure 7 and the subsequent simulations, FIFO and ATS show no meaningful difference in maximum E2E latency for each flow type. It can be conjectured that the burst accumulation in this simulation is not significant, thus the IR of ATS does not contribute to reducing the latency. Some degree of flow isolation is observed for WRR and DRR. However, in flow type C, WRR shows worse performance than DRR because WRR does not consider packet length. Flow type C suffers more in WRR due to its small packet length. C-SCORE provides a good flow isolation, as it achieves smallest latency bounds like those of VC for all flow types.

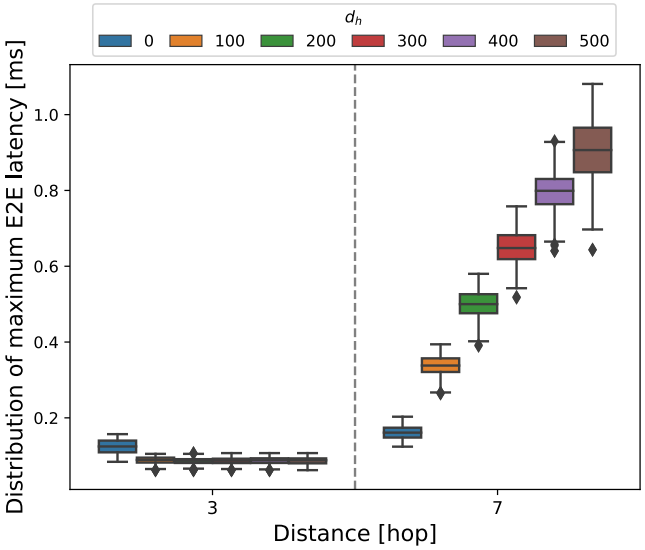

FIGURE 8. The maximum E2E latency distributions of C type flows according to dh value of C-SCORE, when the number of hops is either 3 or 7. The order of boxplots is the same as the order in the legend.

As it has been expected, FIFO does not isolate flows, therefore all the types of flows show similar maximum E2E latencies. On the contrary, VC and C-SCORE successfully isolate flows, thus the E2E latencies are differentiated dramatically with these schedulers.

We examined the case where the arrival process of a flow follows deterministically the arrival curve. Compared to the results in Figure 7, in this case the maximum E2E latencies are higher for all the schedulers. The Variances are also smaller. However, the performance of C-SCORE is still similar to that of VC, and the differences in maximum E2E latencies are not significant.

We have also examined the effect of the hop count on maximum E2E latency, with varying delay factors. Figure 8 shows the distribution of maximum E2E latency by number of hops traveled for type C flows according to d_h at 90% utilization. As shown in Figure 8, as d_h increases, the maximum E2E latency of packets traveling a large number of hops increases, and that of packets traveling a small number of hops decreases. This causes a quality of service difference within the same flow type, which cannot be considered fair. In order to reflect the delay of packets experienced in a node, we may want to determine a proper constant d_h value instead of the minimum value. However, it can still result in unfavorable service for packets traveling a large number of hops. Therefore, in the following subsection, we propose a method to determine the $d_h(p)$ function for each node based on the actual delay experienced in the node.

C. NODE-ADAPTIVE DELAY FACTOR

Since a fixed value of d_h causes the unfairness to flows according to their number of hops, d_h needs to be able to vary appropriately for each node, reflecting the actual delay in the node. We call this a node-adaptive delay factor $d_h(t)$, which is a function of node and time. If this nodal delay is reflected in the FT, the difference between the FT and the actual service time is reduced, allowing for a more accurate alignment between the flows from different paths. For example, we can think of three choices of delay factor reflecting the actual delay.

1) Delay of a packet in the previous node is added to FT. However, doing so could flip the order of services of packets within a flow.
2) Average delay of a flow is added to FT. However, doing so requires flow state maintenance.
3) Average delay of all the packets in a node is added to FT. However, the average delay becomes smaller as time passes, the FTs can be flipped within a flow.

These should be avoided. Therefore, a non-decreasing function of time, which reflects the nodal delay should be used as $d_h(t)$. When the delays of all the packets over all the flows are used, the differentiation between flow types seems to be weakened. However, we have observed in Figure 7 that the max E2E latencies can be differentiated even with a fixed d_h. Therefore, the node-adaptive delay factor has the potential to be an appropriate choice.

Since $d_h(t)$ must be a non-decreasing function, the maximum observed value of the average (max_avg) so far is used as $d_h(t)$, rather than a simple average. Each time a packet is transmitted, the average queuing delay is calculated. If the average calculated is greater than the current max_avg value, the max_avg is updated with the recent average value. The method for determining $d_h(t)$ with max_avg is described in Algorithm 1. To perform this operation, the packet's metadata should include both the arrival time at the node and $d_h(t)$. The arrival time should be updated as soon as the packet arrives at the node, while $d_h(t)$ should be updated every time the packet leaves the node. The queuing delay should be calculated just before the packet leaves the node, and the average of the queuing delays should be computed. Additionally, the node should keep a record of both the average and max_avg value. Continuous increase of $d_h(t)$ in the max_avg method can result in a service differentiation due to the number of hops, just like the cases with a fixed positive value of d_h. To prevent this issue, we reset the max_avg value when the node is empty. Although the average value itself is not reset, $d_h(t)$ continues to accumulate. If the node is empty, adding a relatively small $d_h(t)$ value can reduce the effect of the number of hops. This approach performs better than initializing the max_avg value and the average together.

Figure 9 depicts the maximum E2E latency distributions of type C flows for each scheduler based on utilization, with d_h set as the max_avg value. WRR is excluded from the graph due to a significant difference in maximum E2E latency. C-SCORE shows maximum E2E latency like VC at all utilization levels. In C-SCORE, the max_avg was observed to be up to 351us at utilization 90%. The results are comparable to those obtained when d_h is fixed to the minimum value. However, since d_h is non-zero in the maximum average method,

Algorithm 1 Max_Avg Calculation in C-SCORE

Input: p: packet
n: number of packets transmitted
avg: average of queueing delay
max_avg: maximum value of average

```
n←0
avg←0
max_avg←0
while true do
    if packet p is ready to be transmitted then
        queueing delay← current time – p.node_arrival_time
        avg←(avg × n + queueing delay)/(n+1)
        n←n+1
        if max_avg < avg then
            max_avg←avg
        end
        d_h(t) ←max_avg
    end
    if all queues are empty then
        max_avg←0
    end
end
```

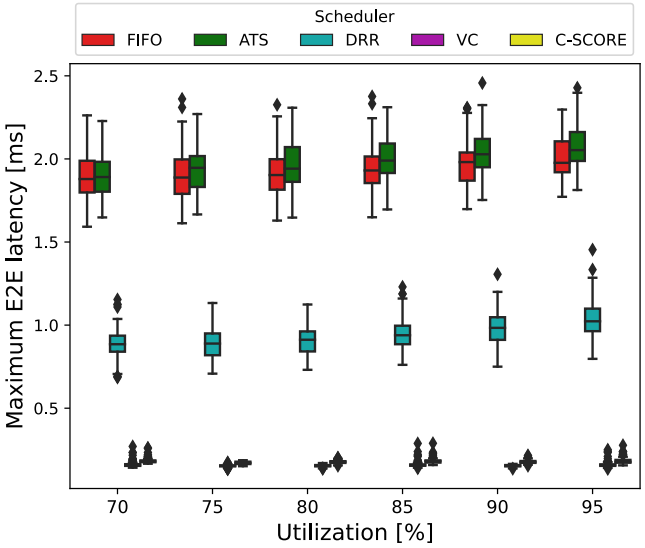

FIGURE 9. The distributions of the maximum E2E latency of C type flows with different schedulers and varying utilization. $d_h(p)$ of C-SCORE is the max_avg function. The order of boxplots is the same as the order in the legend.

resulting in a larger maximum E2E latency for the flows with larger numbers of hops.

Figure 10 depicts the distribution of the maximum E2E latency of type C flows based on the number of hops traveled at 90% utilization, with the delay factor set to the minimum value and with the delay factor set as the max_avg. As mentioned earlier, the maximum E2E latency for packets traveling a large number of hops is increased, while the maximum

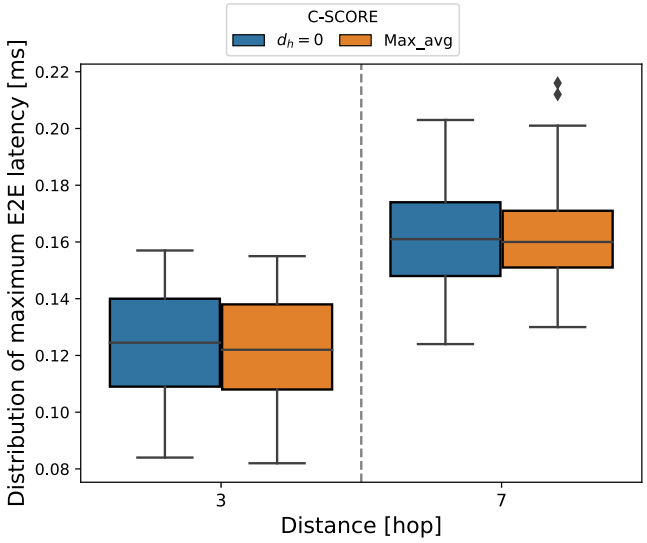

FIGURE 10. Comparison of maximum E2E latency distributions by the number of hops traveled by C type flows with different $d_h(p)$ and 90% utilization. The order of boxplots is the same as the order in the legend.

E2E latency for packets traveling a small number of hops is decreased, although the differences are negligible.

D. FLOW-ADAPTIVE DELAY FACTOR

The disadvantage of using the max_avg as the delay factor is that both the average queuing delay and a packet's queuing delay itself must be continuously calculated and maintained. Since this calculation should be performed right before transmission, it can be burdensome to be executed in real time. Moreover, the delay differences among the different types of flows have to be addressed as well. Intuitively, flows with larger service rate should have experienced lesser delay in a node. This difference has to be reflected in FT.

To solve this issue, we propose a flow-adaptive delay factor that incorporates flow characteristics. In this method, $d_h(p)$ is determined as the *service latency* in the previous node $h-1$, as described in (9) of Section IV. It is repeated here for readability.

$$d_h(p) = \frac{L_{h-1}^{max}}{R_{h-1}} + L_i/r_i, \tag{12}$$

(12) incorporates various parameters, including L_{h-1}^{max} and R_{h-1}, which are information from the previous node and represent the maximum packet length and link capacity at node $h-1$, respectively. L_i and r_i are flow specific information, representing the maximum packet length and service rate of the flow, respectively. This method requires meta-data, including the maximum packet length observed at the previous node, the link capacity of the previous node, the maximum packet length generated by the flow to which the packet belongs, and the service rate. Upon receiving a packet, the proposed method calculates $d_h(p)$ using this meta-data, adds it to the FT, and updates the meta-data from the previous node with those of the current node. Alternatively, the $F_h(p)$ can be pre-calculated at node $h-1$ and written as

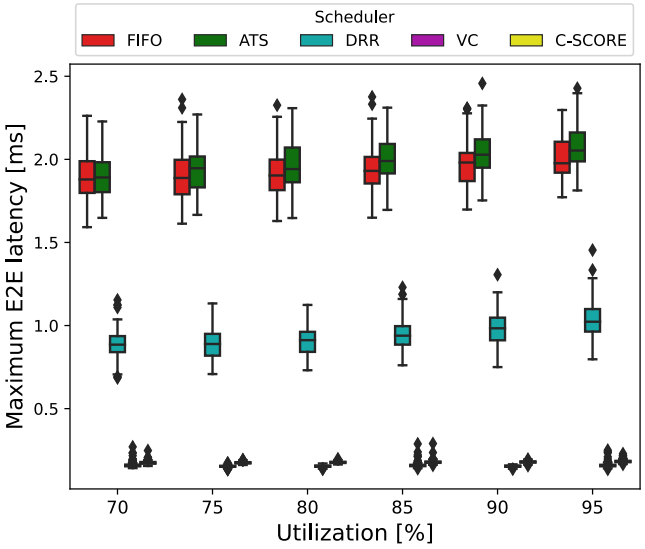

FIGURE 11. The distributions of the maximum E2E latency of C type flows with different schedulers and varying utilization. The C-SCORE is with the flow-adaptive $d_h(p)$. The order of boxplots is the same as the order in the legend.

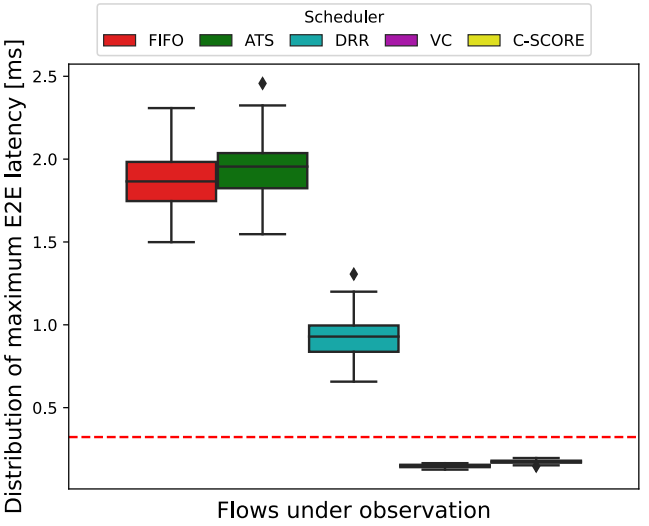

FIGURE 12. The maximum E2E latency distributions of C type flows using various schedulers when utilization is 90%. The C-SCORE is with the flow-adaptive $d_h(p)$. The red dotted line indicates the theoretical E2E latency bound of both VC and C-SCORE. The order of boxplots is the same as the order in the legend.

TABLE 5. The observed maximum E2E latency in millisecond.

Utilization	FIFO	ATS	WRR	DRR	VC	C-SCORE
70%	2.262	2.228	2.026	1.154	0.27	0.248
75%	2.361	2.27	2.19	1.133	0.173	0.192
80%	2.326	2.308	2.819	1.124	0.17	0.198
85%	2.376	2.311	3.4	1.230	0.288	0.29
90%	2.308	2.457	3.736	1.306	0.169	0.196
95%	2.297	2.428	5.228	1.454	0.25	0.229

TABLE 6. Theoretical E2E Latency Upper Bounds At Utilization 90%.

Scheduler	E2E Latency upper bound
ATS	8.481ms
DRR	5.218ms
VC	0.3226ms
C-SCORE (flow-adaptive)	0.3226ms

the meta-data. It has been proven in Section IV, that the proposed flow-adaptive method can guarantee a E2E latency bound for each flow.

Figure 11 depicts the maximum E2E latency distribution of type C flows with varying utilization, obtained using the proposed flow-adaptive $d_h(p)$ as in (9). Like the previous delay factors used, the results demonstrate effective flow isolation, as indicated by a maximum E2E latency similar to that achieved using the stateful VC scheduler in every node.

Figure 12 depicts the distribution of maximum E2E latency for flow type C at a utilization of 90%. The red dotted line in Figure 12 represents the theoretical maximum E2E latency of C-SCORE using the flow-adaptive method at 90% utilization, which is 0.3226ms. Table 5 shows the maximum E2E latency results observed for each scheduler at different levels of utilization. The results for the flow-adaptive method are reported under the label 'C-SCORE' in Table 5. The observed maximum E2E latency for C-SCORE was 0.29ms, confirming that it did not exceed the theoretical maximum at all levels of utilization.

Considering the fact that the most stringent flows require less than 1ms E2E latency bounds [29], the ATS, FIFO, and DRR cannot fulfill this requirement. The C-SCORE is the only scalable solution here.

Figure 13 depicts the distribution of maximum E2E latency by the number of hops for type C flows at a utilization of 90%, when $d_h(p)$ is set to the minimum value and when $d_h(p)$ is set to the flow-adaptive method. Compared to the case where $d_h(p)$ is set to the minimum value, the average maximum E2E latency of packets traveling a large number of hops increases, but the maximum value is similar. Additionally, the maximum E2E latency of packets traveling a small number of hops is greatly reduced. Unlike the previous two methods, the flow-adaptive method assigns different $d_h(p)$ values between flows, resulting in better flow isolation and ensuring that type C flows receive fair service regardless of the number of hops.

The theoretical E2E latency bounds of ATS, DRR, and C-SCORE, when the utilization is 90%, are shown in Table 6. They are calculated based on the inequalities obtained in [8] and [16], and Theorem 3, respectively for ATS, DRR, and C-SCORE. VC has the same bound with C-SCORE. For FIFO, because of the loop in the network, the E2E latency explodes and the bounds cannot be obtained. Note that all the bounds are much larger than the observed maximum latency. However, the order of the bounds and the order of the observed maximum values are identical.

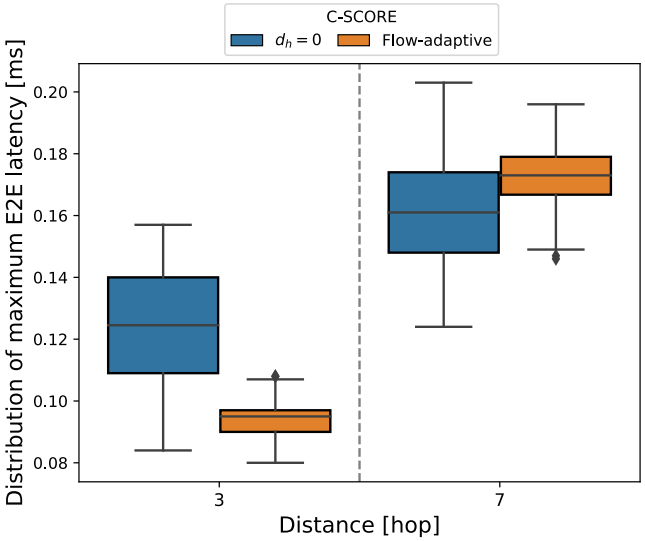

FIGURE 13. Comparison of maximum E2E latency distributions by number of hops traveled by C type flows with different dh(p) and 90% utilization. The order of boxplots is the same as the order in the legend.

E. C-SCORE WITH ALIGNMENT (C-SCORE-A)

All three methods of determining $d_h(p)$ described above assume the use of a priority queue such as heap or PIFO. Since the implementation of a priority queue can be complicated, we would like to find a way to approximate the a priority queue service using only FIFO queues. The reason for using a priority queue was to sort the packets according to their FT order, which can be different from their arrival order.

We first propose the C-SCORE with alignment (C-SCORE-A), which aligns a packet's FT to match the order of arrivals with the order of FTs of packets. Algorithm 2 describes how to determine $d_h(p)$ in C-SCORE-A. There is a FIFO queue per input port. Packets are put into the FIFO queues according to their input ports. The essence of Algorithm 2 is that, if the FT of the arriving packet, $F_h(p)$, would be larger than the port's recent FT value, $E_i(p)$, or smaller than the current nodal alignment target, $S_h(p)$, then to force $F_h(p)$ to become the $\max\{E_i(p), S_h(p)+L(p)/r(p)\}$.

$E_i(p)$ is the maximum FT of packets arrived before p, from the input port i. $S_h(p)$ is the maximum FT among packets serviced so far in the node. By forcing the FT to be such a value, the FT order matches the arrival order of the packets. Therefore, the FTs of packets departing from the node are guaranteed to be sorted.

Note that the delay factor, $d_h(p)$, proposed in this paper is usually a function of node $h-1$, except in the cases with an alignment function.

Figure 14 depicts the maximum E2E latency distribution of type C flows using C-SCORE-A and other schedulers with varying utilization. Unlike C-SCORE, C-SCORE-A's performance changes according to utilization. The higher the utilization, the more packets are aligned at once. The alignment process consequently makes it operate like a FIFO, so the performance of C-SCORE-A is close to that of a FIFO as utilization increases. Conversely, the smaller the utilization, the rarer the alignment between flows, and the order of services is determined by the FT set by the entrance node. Therefore, as the utilization is lowered, the performance becomes like that of VC.

Algorithm 2 C-SCORE-A

Input: i: input port
d_h*: current d value of the input port
p: packet
pi: preceding packet of the input port of packet p
ps: packet being served by the node, or if the node is empty then packet served most recently
$F_h(p)$: FT of packet p in node h
A(p): node arrival time of packet p
L(p): length of packet p
r(p): service rate of flow to which packet p belongs

```
while true do
    if packet p is received then
        Ei(p) ← max{Fh(pi)} + max_i{Li/Ri}
        Sh(p) ← max{Fh(ps(t))}
        if Fh−1(p) + dh* > Ei(p) or
           Fh−1(p) + dh* < Sh(A(p)) then
            dh(p) ← max{Ei(p),
              Sh(A(p)) + L(p)/r(p)} − Fh−1(p)
            Fh(p) ← Fh−1(p) + dh(p)
            dh* ← dh(p)
        end
    end
end
```

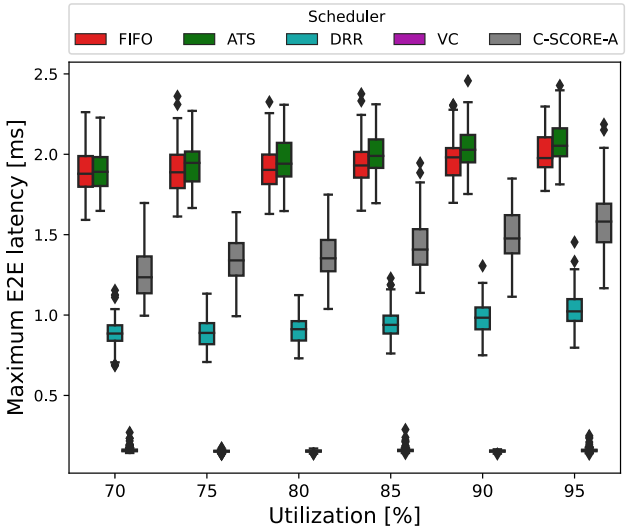

FIGURE 14. The distributions of maximum E2E latency per scheduler with varying utilization. The order of boxplots is the same as the order in the legend.

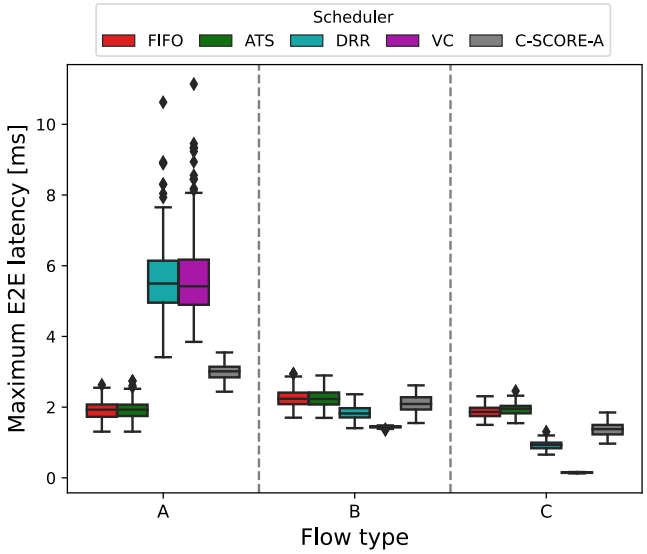

FIGURE 15. The distributions of maximum E2E latency of different flow types with various schedulers at 90% utilization. The order of boxplots is the same as the order in the legend.

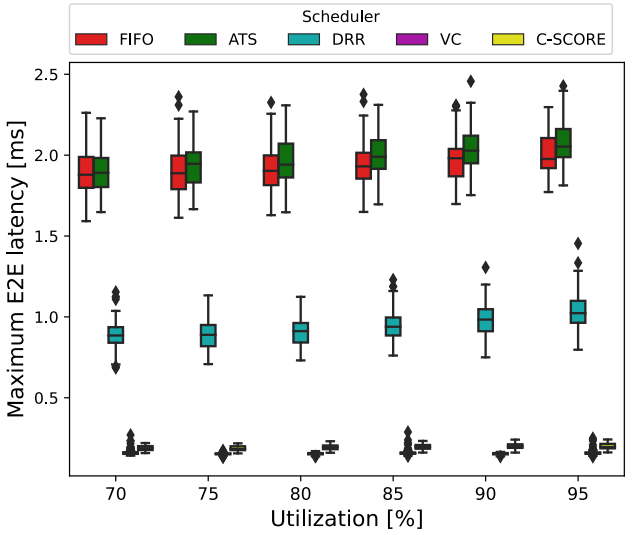

FIGURE 16. The distributions of maximum E2E latency per scheduler according to utilization. C-SCORE uses a FIFO queue for each flow type, with the flow-adaptive $d_h(p)$. The order of boxplots is the same as the order in the legend.

Figure 15 depicts the maximum E2E latency distribution of different flow types using C-SCORE-A and other schedulers when utilization is 90%. Unlike the previous C-SCOREs, flow isolation is not well executed, but still better than FIFO or ATS. C-SCORE-A can be concluded to show the performance between FIFO and VC. Compared to DRR, C-SCORE-A uses a queue for each input port, so it can be used even on core nodes with many flows. Also, when compared to the previous C-SCOREs, it has the advantage of using the simpler FIFO.

F. FIFO QUEUE PER FLOW TYPE

We proposed and examined the C-SCORE-A using FIFO queue with alignment between flows. However, the performance of C-SCORE-A is similar to that of the simple FIFO scheduler, and is inferior to the previously examined C-SCOREs. One important insight gained through the simulations of C-SCORE-A is that, when we put the different types of flows into a single FIFO queue, the actual service times in core nodes can be affected by the parameters of other flows, when using the alignment function. The parameters of different types of flows also alter the flow's own FT distances. Therefore, we propose to allocate FIFO queues for each flow type. The flows of the same type are defined as the flows having the same maximum packet length and the same service rate. As we have noted in (9), the flow-adaptive delay factor function, the service rate and maximum packet size are important pieces of information for determining the FT values in core nodes. A FIFO queue is assigned to the set of flows of a similar or same type. Compared to implementing a separate queue for each flow, using a queue for each flow type requires fewer queues, which can be predicted based on the environment, making implementation easier than VC or DRR. Since there are three types of flows passing through

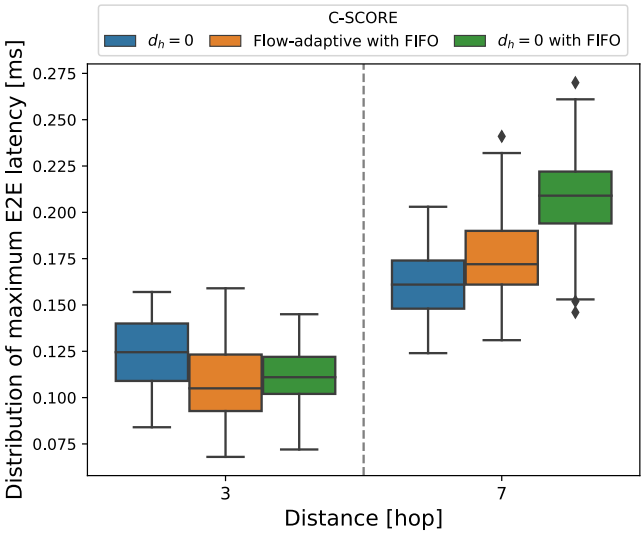

FIGURE 17. Comparison of maximum E2E latency distributions of C type flows having 3 or 7 hops. Utilization is 90%. C-SCORE with PIFO queue with $d_h(p) = 0$ (Blue), FIFO with the flow-adaptive $d_h(p)$ (Orange), and FIFO with $d_h(p) = 0$ (Green). The order of boxplots is the same as the order in the legend.

the simulation environment, three queues are implemented for each output port in the node.

We have observed through simulations that, when FIFO queue per flow type is used, flow isolation performance is improved when $d_h(p)$ is determined either by the flow-adaptive or zero delay factor. Figure 16 shows the maximum E2E latency distribution of C type flows with different schedulers with varying utilization. The C-SCORE, using a FIFO queue for each flow type, shows similar performance to VC. When compared to the C-SCORE with PIFO, using a FIFO queue for each flow type results in a more consistent distribution of maximum E2E latency, with only slight changes observed across different levels of utilization.

Figure 17 shows the maximum E2E latency distribution with varying number of hops traveled by C type flow at 90% utilization, when a PIFO queue with $d_h(p) = 0$ and FIFO queues with the flow-adaptive or zero delay factor. Using a FIFO queue for each flow type is disadvantageous in terms of maximum E2E latency compared to the case with PIFO. It is due to the fact that the service order of packets with small FT values still can be behind the HoQ packet with larger FT. This increase in latency is observed regardless of the number of hops traveled by the packet. Nevertheless, the difference in maximum E2E latency is not significant.

G. COMPARISON WITH NON-WORK CONSERVING FAIR QUEUING SCHEDULERS

We compare the E2E latencies of type C flows, with non-work conserving counterparts, Jitter-VC and CJVC [9]. Jitter-VC is a non-work conserving VC. It requires flow state maintenance in core nodes. Unlike VC, Jitter-VC holds a packet until it is eligible. For the explanation of CJVC, see section III-C. In short, CJVC is a stateless Jitter-VC in core nodes.

Figure 18, 19, and 20 depict the distributions of the maximum E2E latencies, the average E2E latencies, and the E2E latencies, respectively. In a single simulation run, a maximum and an average E2E latencies of the 7-hop C-type flows are observed and determined. With 100 simulation runs, the distributions are obtained and plotted in Figure 18 and 19. The E2E latencies of the 7-hop C-type flows obtained throughout the simulation runs are gathered and plotted in Figure 18. The C-SCORE in these figures uses $d_h(p) = \frac{L_{h-1}^{max}}{R_{h-1}} + \frac{L_i}{r_i}$, the service latency at the previous node.

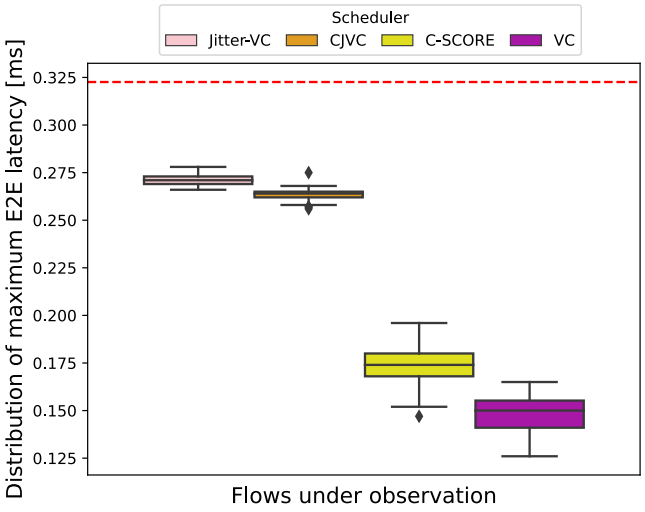

FIGURE 18. Comparison of maximum E2E latency distributions of 7-hop C type flows. Utilization is 90%. C-SCORE with PIFO queue with $d_h(p)$ = service latency. The order of boxplots is the same as the order in the legend.

The red dotted line in Figure 18 represents the theoretical bound for all the four schedulers. It can be seen that this bound is not violated. Figure 18 and Figure 19 show that, for both the maximum E2E latency and the average E2E latency, the work conserving schedulers are better. Figure 20 also shows that

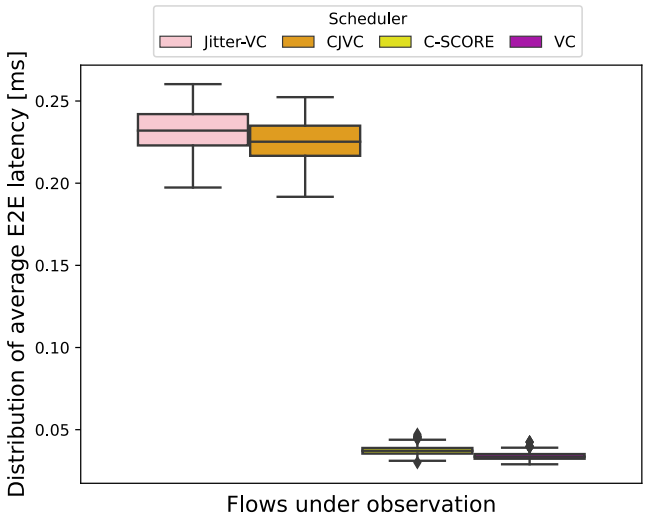

FIGURE 19. Comparison of average E2E latency distributions of 7-hop C type flows. Utilization is 90%. C-SCORE with PIFO queue with $d_h(p)$ = service latency. The order of boxplots is the same as the order in the legend.

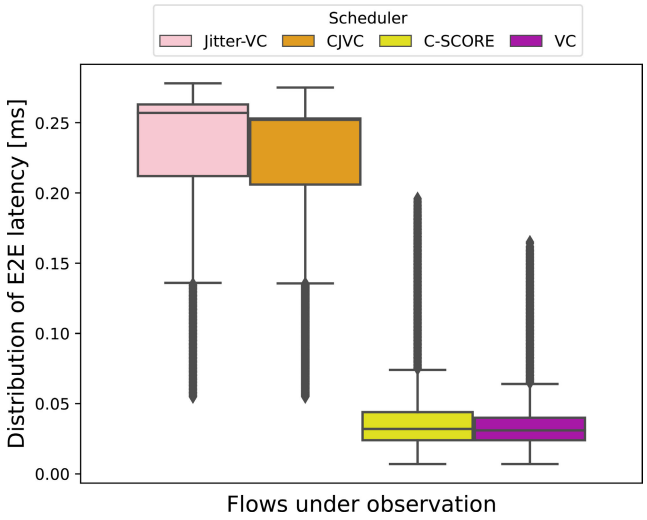

FIGURE 20. Comparison of E2E latency distributions of 7-hop C type flows. Utilization is 90%. C-SCORE with PIFO queue with $d_h(p)$ = service latency. The order of boxplots is the same as the order in the legend.

even in terms of the delay variation, or the jitter, the non-work conserving schedulers do not perform better, contrary to the claim in [9]. This is due to the fact that the eligible time itself used in Jitter-VC and CJVC can vary widely, according to the packet's order within a burst. For a packet placed last in a burst, the eligible time is quite different from the arrival time, while for a packet placed first in a burst has an eligible time that is identical to the arrival time.

H. SIMULATION WITH A COMPLEX TOPOLOGY

To observe performance variations based on changes in C-SCORE's topology, simulations were conducted using a different topology. A topology consisting of a hierarchical structure comprising local area networks (LANs), metropolitan area networks (MANs), and a core network;

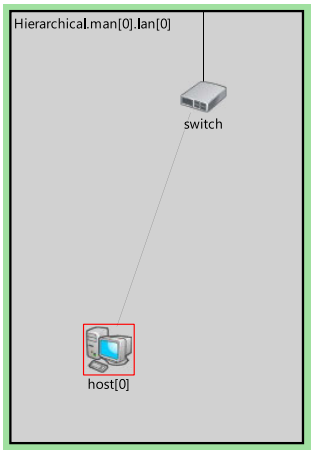

FIGURE 21. The architecture of the LAN in the simulation.

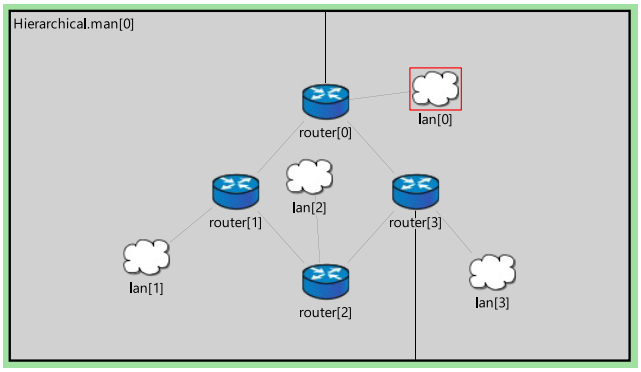

FIGURE 22. The architecture of the MAN in the simulation.

altogether having a diameter of 16 hops was employed. The network was implemented using OMNeT++ [30] and INET [31], based on the example topology provided by INET called 'Hierarchical'.

The LAN represents the lowest level network that includes hosts, as shown in Figure 21. The link capacity is 1 Gbps. For implementation convenience, only one host was instantiated. This host runs six applications: three of them transmit flows, while the other three receive flows. The characteristics of flows, except for the destination, remain consistent with those outlined in Table 2. There are also three types of flows, maintaining a service rate ratio of 1:10:10. The service rates of flows were set to achieve a 72% utilization in the core network. The host transmits one flow type each, totaling three flows, with the average traffic rate of 450 Mbps. The packet generation process for applications remains the same as in previous simulations, utilizing a Markov on-off model and token bucket.

The MAN employs a ring topology, as illustrated in Figure 22. The link capacity between routers and between routers and LANs is 1 Gbps, while the links connecting to the core network have a capacity of 10 Gbps. Each router is connected to one LAN, and two routers are linked to the core network.

The core network employs a grid topology, as depicted in Figure 23. The link capacity is 10 Gbps. Each router is

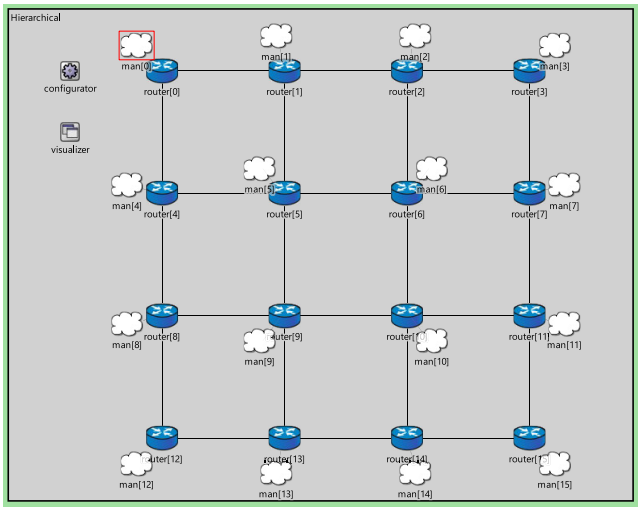

FIGURE 23. The architecture of the core network in the simulation.

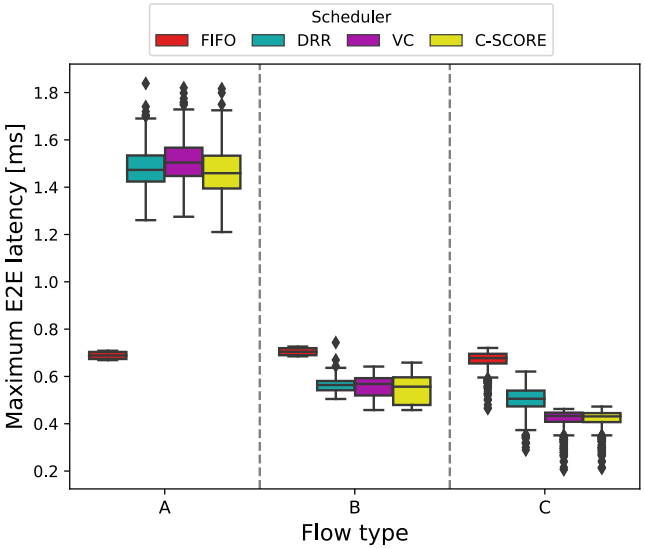

FIGURE 24. The distributions of maximum E2E latency of different flow types with various schedulers at 72% utilization. The order of boxplots is the same as the order in the legend.

connected to one MAN with two links. The core network achieves a maximum utilization of 72%, with both directions of the respective links showing a 72% utilization. Flows have symmetric source and destination pairs with respect to the center of the topology. For example, in Figure 23, flows emanating from man[0] are headed towards man[15], while flows originating from man[8] are directed towards man[7].

A scheduler was implemented on every switch and router at all the layers. The schedulers used in the simulation were FIFO, DRR, VC, and C-SCORE. C-SCORE utilized PIFO and a flow-adaptive delay factor. Hosts exclusively used FIFO scheduling. Each transmitting application sent 1000 packets. The simulation was repeated 30 times. Since there was little difference between the results for the entire set of hosts in the simulation and the symmetric half, we obtained results using only the corresponding half of the results.

Figure 24 depicts box-and-whisker plots of the maximum end-to-end latency for each flow in each simulation. It demonstrates that C-SCORE successfully protects the target flow type, denoted as C. Furthermore, it is shown that in a topology where networks with different link capacities are connected, C-SCORE exhibits performance similar to VC.

I. ANALYSIS SUMMARY

In Sections IV-B ∼ V-F, we have examined various delay factor functions, $d_h(p)$, regarding their performances in terms of observed maximum E2E latency. A fixed value of the delay factor in a node independent of the flow, i.e. $d_h(p) = d_h$, shows surprisingly good performance considering its simplicity of updating the FT in core nodes. A smaller value of d_h gives better performance for flows with larger numbers of hop, and vice versa. Setting d_h to be zero, or the propagation delay of the link in the realistic case, is similar to considering the network as a single service entity. If, regardless of the hop count, the flows should have the same level of E2E latencies, then this choice of d_h would be fair, and would give a better overall E2E latency bound.

The flow-adaptive choice of $d_h(p)$ that is dependent on the flow the packet belongs to, $d_h(p) = \frac{L_{h-1}^{max}}{R_{h-1}} + L_i/r_i$ as in (9), yields an explicit expression for the E2E latency upper bound. The simulation results confirm this mathematical expression over various network utilizations. When compared to the case of a fixed delay factor value, the flow-adaptive delay factor requires additional packet meta-data, that is $\frac{L_{h-1}^{max}}{R_{h-1}}$ and L_i/r_i. However, $\frac{L_{h-1}^{max}}{R_{h-1}}$ value can be signaled out-of-band between neighboring nodes. Moreover, $F_h(p)$ itself can be calculated at node $h-1$ and be put into the packet before sending to the node h.

For the fixed or the adaptive delay factor function, a proper implementation requires a priority queue. In order to avoid the use of a priority queue, we have examined the possibility of aligning the order of FTs and the order of arriving times of packets to coincide. This was called the C-SCORE with alignment. However, the alignment algorithm makes the C-SCORE have the characteristics of a FIFO scheduler.

From the observation that the order of FTs is similar to the order of arrival times of packets in core nodes, we have put flows with the same characteristics into a single FIFO queue, with the flow-adaptive delay factor in (9) or zero delay factor. The maximum E2E latencies of schedulers with such FIFO queues are shown to be similar to that with the PIFO queue. The C-SCORE with FIFO per similar flows with zero delay factor can be seen as an alternative choice of the most sophisticated implementation suggested in this paper, which is the one using PIFO with the delay factor in (9).

VI. CONCLUSION

In this paper, a framework for deriving FTs in core nodes without maintaining flow states is presented, in which initial FT obtained in the entrance node is updated in a core node by adding a nodal delay factor, which is a function of parameters that can depend on upstream nodes and flow itself. The packets are to be put into a queue, if there is a packet currently being served. The packets in the queue have to be served in the increasing order of FTs. Therefore, in general the proposed scheduler requires a priority queue and is work conserving. It is proven that, for a certain choice of the delay factor function, the mathematical expression for the E2E latency upper bound exists. It is remarkable that this E2E latency bound function is the same as the one with a network having stateful fair queuing schedulers in all the nodes. It is well understood that the fair queuing schedulers are the best in terms of flow isolation and latency bound. Moreover, the bound is the function of the flow intrinsic parameters, except the ratios of the maximum packet length on a link and the capacity of the link. Therefore, the bound is adjustable with the flow's own parameters. A network can fulfill the requested E2E latency bound of a flow by allocating especially a proper service rate to the flow.

It is also proposed a FIFO-based architecture, by using the fact that the service order between packets of the flows passing through the same path can be unaltered in the middle of the path if the delay factors added to the FTs of these flows are non-decreasing function of time. In this FIFO-based architecture, flows having the similar parameters, such as the maximum packet size and the service rate, are put into the same FIFO queue. It is shown that this scheduler's E2E latency performance matches those of priority queue-based architectures.

Through the extensive simulations, we have shown that the proposed mechanism provides an ideal flow isolation performance, without having to maintain flow states in core nodes. The framework is shown to be also robust, as it provides a stable maximum E2E latency throughout the various choices of delay factor function, at various utilization levels within 70 ∼ 95%. We also showed that the proposed work conserving fair queuing scheduler is superior to the existing non-work conserving fair queuing schedulers, Jitter-VC and CJVC, in terms of both the maximum and the average E2E latency with simulations. It was also shown that the jitters of these non-work conserving schedulers are not better than the ones of the proposed work conserving scheduler.

In practice, it is anticipated the simplest implementation can be adopted, in which a core node maintains a fixed delay factor value, e.g. the propagation delay of a link, for all the flows; and assigns a FIFO queue for flows with similar types. This simplest implementation shows a comparable performance with the most sophisticated implementation of C-SCORE, which uses PIFO and the service latency as the delay factor. Considering the fact that the types of traffic that require deterministic networking are limited, we conjecture that the number of FIFO queues required is within a reasonable range. The fixed delay factor value could be determined through long-term observations or by an advanced approach such as the one with reinforcement learning. Moreover the FT at node h can be pre-calculated at node $h-1$ with the delay factor value that is the function of node $h-1$.

In this case the FT is the only meta-data necessary. This practical implementation would provide an enough level of flow isolation and E2E latency upper bound.

The flow states still have to be maintained in the entrance nodes. The notion of the entrance node, however, can be mitigated into various edge devices, including the source itself. The FT value of a packet is decided based on the maximum of $F_0(p-1)$ and $A_0(p)$; and $L(p)/r$. These parameters are flow-specific. There is no need to know any other external parameters. The arrival time of p to the network, $A_0(p)$, can be approximated by the generation time of p at the source. Then $F_0(p)$ is determined at the packet generation time and can be recorded in the packet. Therefore, we can simplify the proposed solution to a great degree, and can apply to any network with robustness and scalability. The possibility of further simplifying the framework will be studied in the future.

The network used in most of the simulations has a moderate size and a symmetric topology. Although its topology includes loops, so that burst accumulation effect can be maximized, the moderate size can be its limit. To verify performance in a realistic large-scale network, a network with a hierarchical structure is being implemented using OMNeT++, with a goal of 50 or more nodes. The link capacities and topology can be chosen randomly. The characteristics of the flow are set to follow those of the actual flow being serviced on the Internet. The initial results from such a complex network are shown in Section V-H.

The framework in this work has been proposed in various International standard organizations by the authors, as a solution for deterministic networking, including IETF DetNet WG [32], [33], [34] and ITU-T SG 13 [35]. Considering there are a number of standard working groups that try to utilize various meta-data for management and packet handling, such as routing in IETF SPRING WG [36], adding the FT as another meta-data will not introduce too much burden.

REFERENCES

[1] *Network 2030 Services: Capabilities, Performance and Design of New Communication Services for the Network 2030 Applications*, document Rec. ITU-T Y.Sup66, ITU-T Y.3000-Series, International Telecommunication Union, Geneva, Switzerland, Jul. 2020.

[2] *Time-Sensitive Networking Task Group Home Page*, Standard IEEE 802.1. Accessed: Sep. 25, 2023. [Online]. Available: http://www.ieee802.org/1/pages/tsn.html

[3] J. L. Messenger, "Time-sensitive networking: An introduction," *IEEE Commun. Standards Mag.*, vol. 2, no. 2, pp. 29–33, Jun. 2018, doi: 10.1109/MCOMSTD.2018.1700047.

[4] *IETF Deterministic Networking (DetNet) Working Group Home Page*. Accessed: Sep. 25, 2023. [Online]. Available: https://datatracker.ietf.org/wg/detnet/about/

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, *An Architecture for Differentiated Services*, document RFC 2475, 1998.

[6] L. Thomas, J.-Y. Le Boudec, and A. Mifdaoui, "On cyclic dependencies and regulators in time-sensitive networks," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, York, U.K., Dec. 2019, pp. 299–311.

[7] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Trans. Netw.*, vol. 1, no. 3, pp. 344–357, Jun. 1993, doi: 10.1109/90.234856.

[8] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE/ACM Trans. Netw.*, vol. 4, no. 3, pp. 375–385, Jun. 1996, doi: 10.1109/90.502236.

[9] I. Stoica and H. Zhang, "Providing guaranteed services without per flow management," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 81–94, Oct. 1999, doi: 10.1145/316194.316208.

[10] Z.-L. Zhang, Z. Duan, and Y. T. Hou, "Virtual time reference system: A unifying scheduling framework for scalable support of guaranteed services," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 12, pp. 2684–2695, Dec. 2000, doi: 10.1109/49.898750.

[11] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 25: Enhancements for Scheduled Traffic*, Standard IEEE 802.1Qbv-2015, 2015.

[12] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 29: Cyclic Queuing and Forwarding*, Standard IEEE 802.1Qch-2017, 2017.

[13] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 34: Asynchronous Traffic Shaping*, Standard IEEE 802.1Qcr-2020, 2020.

[14] E. Mohammadpour, E. Stai, M. Mohiuddin, and J. Y. Le Boudec, "Latency and backlog bounds in time-sensitive networking with credit based shapers and asynchronous traffic shaping," in *Proc. IEEE 30th Int. Teletraffic Congr. (ITC)*, vol. 2, Sep. 2018, pp. 1–6, doi: 10.1109/ITC30.2018.10053.

[15] *Requirements and Framework for Latency Guarantee in Large Scale Networks Including IMT-2020 Network*, document Rec. ITU-T Y.3113, 2021.

[16] J. Joung, "Framework for delay guarantee in multi-domain networks based on interleaved regulators," *Electronics*, vol. 9, no. 3, p. 436, Mar. 2020, doi: 10.3390/electronics9030436.

[17] D. Stiliadis and A. Varma, "Rate-proportional servers: A design methodology for fair queueing algorithms," *IEEE/ACM Trans. Netw.*, vol. 6, no. 2, pp. 164–174, Apr. 1998, doi: 10.1109/90.664265.

[18] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," in *Proc. ACM Symp. Commun. Architectures Protocols*, Aug. 1990, pp. 19–29, doi: 10.1145/99508.99525.

[19] S. J. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proc. Conf. Comput. Commun. (INF)*, 1994, pp. 636–646, doi: 10.1109/INFCOM.1994.337677.

[20] D. Stiliadis and A. Varma, "Efficient fair queueing algorithms for packet-switched networks," *IEEE/ACM Trans. Netw.*, vol. 6, no. 2, pp. 175–185, Apr. 1998, doi: 10.1109/90.664266.

[21] D. Stiliadis and A. Varma, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans. Netw.*, vol. 6, no. 5, pp. 611–624, Oct. 1998, doi: 10.1109/90.731196.

[22] R. Bhagwan and B. Lin, "Design of a high-speed packet switch with fine-grained quality-of-service guarantees," in *Proc. IEEE Global Converg. Through Commun. Conf. Rec. (ICC)*, vol. 3, Jun. 2000, pp. 1430–1434, doi: 10.1109/ICC.2000.853733.

[23] P. Goyal and H. M. Vin, "Generalized guaranteed rate scheduling algorithms: A framework," *IEEE/ACM Trans. Netw.*, vol. 5, no. 4, pp. 561–571, Aug. 1997, doi: 10.1109/90.649514.

[24] J. Kaur and H. M. Vin, "Core-stateless guaranteed rate scheduling algorithms," in *Proc. IEEE Conf. Comput. Commun. 20th Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, vol. 3, Apr. 2001, pp. 1484–1492, doi: 10.1109/INFCOM.2001.916644.

[25] P. Goyal, S. S. Lam, and H. M. Vin, "Determining end-to-end delay bounds in heterogeneous networks," *Multimedia Syst.*, vol. 5, no. 3, pp. 157–163, May 1997, doi: 10.1007/s005300050052.

[26] *SimPy*. Accessed: Mar. 15, 2023. [Online]. Available: https://simpy.readthedocs.io/en/latest/

[27] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown, "Programmable packet scheduling at line rate," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 44–57, doi: 10.1145/2934872.2934899.

[28] R. Bhagwan and B. Lin, "Fast and scalable priority queue architecture for high-speed network switches," in *Proc. IEEE Infocom Conf.*, Tel Aviv, Israel, Mar. 2000, pp. 26–30. [Online]. Available: http://www.ieeeinfocom.org/2000/papers/565.ps

[29] P. J. Chaine, M. Boyer, C. Pagetti, and F. Wartel, "TSN support for quality of service in space," presented at the 10th Eur. Congr. Embedded Real Time Softw. Syst. (ERTS), Toulouse, France, Jan. 2020. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02441327/document

[30] *OMNeT++ Discrete Event Simulator*. Accessed: Sep. 6, 2023. [Online]. Available: https://omnetpp.org/
[31] *INET Framework*. Accessed: Sep. 6, 2023. [Online]. Available: https://omnetpp.org/
[32] J. Joung, J.-D. Ryoo, T.-S. Cheung, Y. Li, and P. Liu, *Asynchronous Deterministic Networking Framework for Large-Scale Networks*, Standard draft-joung-detnet-asynch-detnet-framework-02, IETF DetNet WG, Internet draft, IETF, Mar. 2023.
[33] J. Joung, J.-D. Ryoo, T.-S. Cheung, Y. Li, and P. Liu, *Latency Guarantee With Stateless Fair Queuing*, Standard draft-joung-detnet-stateless-fair-queuing-00, IETF DetNet WG, Internet draft, IETF, Jul. 2023.
[34] J. Joung, J. Kwon, J.-D. Ryoo, and T. Cheung, "Asynchronous deterministic network based on the DiffServ architecture," *IEEE Access*, vol. 10, pp. 15068–15083, 2022, doi: 10.1109/ACCESS.2022.3146398.
[35] *Requirements and Framework for Stateless Fair Queuing in Large Scale Networks Including IMT-2020 and Beyond*, document ITU-T SG13-TD341/WP1, Draft new Recommendation ITU-T Y.det-FQ-RF, Mar. 2023.
[36] *IETF Source Packet Routing in Networking (spring) Working Group Home Page*. Accessed: Sep. 25, 2023. [Online]. Available: https://datatracker.ietf.org/wg/spring/

JINOO JOUNG received the B.S. degree in electronics engineering from the Korea Advanced Institute for Science and Technology, Daejeon, South Korea, in 1992, and the M.S. and Ph.D. degrees in electrical and electronics engineering from New York University, New York City, NY, USA, in 1994 and 1997, respectively. From 1997 to 2005, he was with Samsung Electronics and the Samsung Advanced Institute for Technology, where he was involved in various network SOC research and developments, especially for general packet radio service for 3G mobile systems, network processors for high-speed IP routers, and mobile application processors for smart handheld devices. In 2005, he joined Sangmyung University, Seoul, South Korea. His research interests include network SOCs, high-speed network processing, switch architecture, network calculus, network QoS control, and autonomous wireless network scheduling/routing. He is active in international standardization activities. He is the main Editor of various ITU-T recommendations, including Y.2122, Y.3113, Y.3118, and Y.3120.

JUHYEOK KWON received the B.S. degree in human-centered artificial intelligence and the M.S. degree in artificial intelligence and informatics from Sangmyung University, Seoul, South Korea, in 2020 and 2022, respectively, where he is currently pursuing the Ph.D. degree in artificial intelligence and informatics. His research interests include deterministic network services, wireless autonomous networks, and network SOC design.

JEONG-DONG RYOO received the B.S. degree in electrical engineering from Kyungpook National University, Daegu, Republic of Korea, and the M.S. and Ph.D. degrees in electrical engineering from New York University, New York, NY, USA. After completing the Ph.D. degree in the area of telecommunication networks and optimization, he was with Bell Labs, Lucent Technologies, Holmdel, NJ, USA, from 1999 to 2004. While he was with Bell Labs, he was mainly involved in performance analysis, evaluation, and enhancement studies for various wireless and wired network systems. He is currently a Principal Researcher with the Electronics and Telecommunications Research Institute (ETRI), Daejeon, Republic of Korea, and a Professor of information and communication engineering major with the ETRI School, University of Science and Technology, Daejeon. Since he joined ETRI in 2004, he has been focused on next-generation networks, carrier-class Ethernet, MPLS-TP and DetNet technology research, especially participating in standardization activities in ITU-T and IETF. He has coauthored *TCP/IP Essentials: A Lab-Based Approach* (Cambridge University Press, 2004). He is a member of the Eta Kappa Nu Association. He served as the Vice Chairperson of the ITU-T Study Group 15, from 2013 to 2022. He is an Editor of various ITU-T recommendations, including G.8131 (MPLS-TP linear protection), G.8132 (MPLS-TP ring protection), G.8331 (MTN linear protection), G.873.1 (OTN linear protection), G.808.1 (Generic protection-Linear), and G.808.4 (Linear protection for fgOTN and fgMTN). He is the co-editor or the coauthor of several IETF RFCs, including RFC 7271, RFC 7347, RFC 7412, RFC 8150, RFC 8234, and RFC 9270.

TAESIK CHEUNG received the B.S., M.S., and Ph.D. degrees in electronics engineering from Yonsei University, Seoul, Republic of Korea. Since 2000, he has been a Principal Researcher with the Electronics and Telecommunications Research Institute (ETRI), where he has been engaged in the development of network systems. Since 2005, he has participated in ITU-T Q9/15 and contributed to the standardization of protection mechanisms for transport networks. Since 2010, he has participated in the IETF MPLS working group and has contributed to MPLS-TP standardization, especially in the area of survivability. He is the co-editor of the ITU-T Rec. G.873.2 and G.808.2, and the coauthor of IETF RFC 7271 and RFC 8234. He is also involved in time-sensitive packet networking technologies, such as IEEE 802.1 TSN and IETF DetNet.