IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 28, 2021, accepted January 10, 2022, date of publication January 26, 2022, date of current version February 10, 2022.

*Digital Object Identifier 10.1109/ACCESS.2022.3146398*

# Asynchronous Deterministic Network Based on the DiffServ Architecture

**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 Intelligence Information Engineering, Sangmyung University, Seoul 03016, South Korea
[3]Electronics and Telecommunications Research Institute (ETRI), Daejeon 34129, South Korea
[4]Network Engineering Major, University of Science and Technology, Daejeon 34113, South Korea

Corresponding author: Jinoo Joung (jjoung@smu.ac.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 Korea Government [Ministry of Science and ICT (MSIT)/Institute for ICT Planning and Evaluation (IITP)] through the Development of End-to-End Ultra-High Precision Network Technologies under Grant 2021-0-00715.

**ABSTRACT** In this study, we propose a scalable framework that guarantees both latency and jitter bounds in large networks, including the Internet. The framework is composed of two parts: a latency-guaranteeing network and a jitter-guaranteeing end system. For latency bounds, we suggest regulators per class per input–output port pair of the DiffServ-type relay nodes. For jitter bounds, based on the guaranteed latency bounds, we suggest time-stamping and buffers at the network egress edge. The framework does not require network-wide time synchronization, frequency synchronization, flow state maintenance, or flow-level queuing/scheduling. Therefore, the complexity does not increase as the number of flows or network size increases. Moreover, the framework is based on the DiffServ architecture; therefore, it requires minimal modification to the current Internet. We demonstrate that the proposed regulators can achieve latency bounds comparable to the IEEE asynchronous traffic shaping technique. We prove that the jitter is bounded even with realistic limitations such as buffers without cut-through capability. We also prove that in the presence of clock drift, the jitter can still be upper bounded with a suggested compensation algorithm. We demonstrate through experiments on simple programmable microcontrollers that the jitter upper bound can be within a few tens of microseconds, even in a realistic situation with store-and-forward buffers, clock drift, and random network delays.

**INDEX TERMS** Buffer, deterministic network, DiffServ, jitter, latency, regulator, time synchronization.

## I. INTRODUCTION

### A. DEMANDS FOR LARGE-SCALE DETERMINISTIC NETWORKS

There are emerging applications that require both latency and jitter bounds in large-scale networks [1]. We define the services required for such applications as deterministic services. Consequently, a network that can provide such a service, even with dynamic input and various network topologies, is defined as a deterministic network. Machine-to-machine communications for ''cloudified'' industrial and robotic automation involve moderate-to-large-scale deterministic networks. This type of communication requires fine-grained timing accuracy for the dispersion of control commands and the collection of telemetry data over a wide area [1]. The ITU-T SG-13 has defined such services to support critical grade reliability and extremely low as well as highly precise latency for the delivery of packets [1]. This is because some industrial controllers require very precise synchronization and spacing of telemetry streams/control data, facilitating, for example, the precise operation of robotic effectors with multiple degrees of freedom [1]. For services including the tactile Internet and holographic-type communications, the importance of on-time service as well as in-time service in large-scale networks is also emphasized in Ref. [1].

### B. PROBLEM STATEMENTS AND RELATED WORKS

We consider the problem of guaranteeing both latency upper bounds and jitter upper bounds in arbitrarily sized networks with any type of topology, with random dynamic input traffic. The jitter is defined as the latency difference between two packets within a flow, not a difference from a clock signal

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

or from an average latency, as is clearly summarized in RFC 3393 [2].

For small networks such as in-car networks or 5G fronthaul networks, the IEEE TSN task group defines a set of solutions for latency and jitter minimization [3]. A part of the solutions relies on the time synchronization of every node in the network and slot scheduling, which is coordinated among nodes throughout the network. In time-synchronized systems, significant events occur simultaneously with the same rate [4]. In contrast, in frequency-synchronized systems, events occur at the same rate but not necessarily simultaneously [4]. The TSN task group has standardized multiple functional components for jitter-sensitive services based on time synchronization. Among them, the 802.1Qbv time sensitive queues (also known as the time-aware shaper, TAS), and 802.1Qch cyclic queuing and forwarding (CQF) are built for jitter minimization as well as latency guarantee.

The TAS defines the gate for each queue. The gates are either open or closed in a time slot. Multiple gates may open simultaneously. In such a case, strict priority scheduling applies to open queues. A central network controller determines which gates to open and when to open. The CQF function coordinates the slot openings in adjacent nodes such that flows traversing the nodes experience the minimum latency. Based on these functions, the TSN supports a model for deterministic packet services. In this study, we refer to these functions collectively as the TSN synchronous approach. The TSN synchronous approach requires three major efforts: 1) time synchronization along the nodes in the network, 2) slot scheduling and coordination by a central entity, and 3) feasibility testing for flow admission control. These requirements considerably affect network scalability. Obtaining a non-conflicting slot schedule has been proven to be NP-complete, even with fixed-sized slots similar to the TDMA [5]. Refs. [6], [7] proved that the problem of producing a non-conflicting schedule to multiplex multiple flows can be reduced to the classical graph-coloring problem, which is known to be NP-complete.

The time synchronization requirement across every node in the network has two difficulties. First, the implementation of the time synchronization function may impose hardware support and consequently too much overhead to lightweight embedded nodes. Second, synchronization accuracy may not be up to the level of traffic requirements. There are two representative protocols for network-wide time synchronization: the network time protocol (NTP) and the precision time protocol (PTP) [8], [9]. NTP uses timestamps in the application layer and has a limited synchronization accuracy of 1 ms [10]. In software-based implementations, the PTP mechanism achieves a mean synchronization accuracy of 30 $\mu$s and a standard deviation of 54 $\mu$s [11]. These results can be slightly improved by using real-time operating systems and communication channels, but even in this case, reaching submicrosecond synchronization accuracy remains a difficult task [12].

The PTP measures the one-way transit delay from a master to a slave by exchanging messages in both directions with time stamps. Based on the obtained latency value and the time stamp of the master, the slave node calculates the clock offset and corrects it. For this message exchanging protocol to accurately work, a few assumptions must be made. First, this exchange of messages occurs over a time interval so small that this offset can safely be considered constant over that interval. Second, the transit time of a message going from the master to the slave is equal to the transit time of a message going from the slave to the master. Third, both the master and slave can accurately measure the instance of sending or receiving a message. The degree to which these assumptions hold true determines the accuracy of the clock at the slave device [9]. These assumptions, especially the second one, may go wrong in networks with a large number of hops. In the PTP peer-to-peer delay mechanism, the transit delays in the nodes are measured and compensated for the end-to-end delay. Using this advanced method, the previous limitations of PTP can be meditated. However, 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 for application.

Recently, studies have been conducted on scalable deterministic services under the title of large deterministic networks (LDN), based on the technologies of the IETF DetNet Working Group [13]–[15]. These works are based on the CQF technique, which was originally initiated in the IEEE TSN task group. The modifications mainly deal with relaxing the time-synchronization requirements for slot scheduling to accommodate the uncertainties from various propagation delays along the paths. One of the solutions to predict the propagation delays of a flow is to specify the route at the source using the segment routing technology [15]. However, these approaches still require static network topology and flow setup. The slot scheduling, ahead of the network operation, is still NP-hard and requires optimization based on heuristics. The most important drawback of these LDN approaches is that they do not easily coexist with the Internet, which can be represented by statistical multiplexing and work-conserving scheduling.

In large-scale networks, the 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 more challenging. The current Internet, which has adopted the DiffServ architecture, has two problems for deterministic operations: The first one is the cyclic dependency problem, which is 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 [16]. 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 [17], [18]. Second, the current Internet does not have an explicit solution for the jitter bound. Therefore, solving

this problem as a joint optimization problem is even more difficult. We suggest decomposing the problem into two independent subproblems.

For the first problem of latency bound alone, the IEEE asynchronous traffic shaping (ATS) or the flow-aggregate interleaved regulators (FAIR) [19], [20] frameworks can be a solution. The key component of the ATS framework is the interleaved regulators (IRs) placed at the output ports per port per class. The IR has a single queue for all flows of the same class from the same input port. The head of the queue (HOQ) is examined if the packet is eligible to exit the regulator. To decide whether it is eligible, the IR must maintain the individual flow states. Therefore, the ATS suffers from two major complex tasks: flow state maintenance and HOQ lookup to determine the flow to which the packet belongs. Both tasks involve real-time packet processing and queue management. As the number of flows increases, the IR operation may become burdensome as much as the per-flow regulators.

The FAIR aims for a latency-bound guarantee in large-scale networks. Within an aggregation domain (AD), the flows are aggregated into a flow aggregate (FA) according to their ingress/egress ports to/from the AD, or additionally according to the flow requirements and characteristics. The FAs are treated with separated queues and fair-queuing schedulers, thus maintaining the FIFO property throughout the AD. Subsequently, at the boundary of the AD, the minimal IRs per FA were placed for burst suppression. The framework guarantees an end-to-end delay bound with reduced complexity compared to the traditional flow-based approach. Numerical analysis shows that the framework yields smaller bounds than both the flow-based frameworks such as the integrated services (IntServ) and the class-based ATS, at least in networks with identical flows and symmetrical topology. The ATS can be considered as a special case of FAIR, where an AD corresponds to a single node with a FIFO scheduler. The IntServ can also be considered as an extreme case of the FAIR, with an AD corresponding to an entire network; therefore, regulators are unnecessary. However, FAIR still requires IRs per FA at the AD boundary.

For the second problem of jitter bound, the buffered network (BN) [21] framework can be the solution. The BN framework is composed of a network that guarantees latency upper bounds, a timestamper for packets with a clock that is not necessarily synchronized with the other nodes, which resides in between, including the source and the network ingress interface, and a buffer that can hold the packets for a predetermined interval, which resides in between, including the destination and the network egress interface. However, the BN presumes that the buffer is capable of cutting through. It also does not consider the clock drift problem. As the BN does not synchronize the clocks in different nodes in a network, clock drift is unavoidable. Consequently, jitter may occur owing to the clock frequency difference or clock drift between the source and the buffer. We need to extend the framework to work in more realistic network environments.

### C. CONTRIBUTIONS AND CONTENTS

This study makes three major contributions. Based on these contributions, scalable determinism can be achieved even in large networks, including the Internet.

First, we propose a scalable architecture to guarantee latency upper bounds, which includes regulators per class and per port. The suggested regulators can act as cycle breakers. Regulators are scalable because they are not required to maintain flow states. Through numerical analysis, the architecture with regulators can achieve latency bounds comparable to those of the IEEE ATS technique.

Second, we generalize the BN architecture so that the jitter is upper bounded even with buffers without cut-through capability and with the existence of clock drift. We propose a clock drift compensation algorithm that does not require the exchange of messages between network nodes. Therefore, our algorithm is free from the requirements that are necessary in the NTP and PTP, such as the constant network delay or equal delays for both directions.

Third, we demonstrate through experiments on simple programmable microcontrollers that the jitter upper bound can be within a few tens of microseconds, even in a realistic situation with store-and-forward buffers, clock drift, and random network delays.

Consequently, we propose an overall framework for both latency and jitter bounds, which is scalable and can be applied to the Internet.

## II. PORT-BASED REGULATOR FOR LATENCY BOUND GUARANTEE

### A. PROPOSED ARCHITECTURE

We propose the use of per-port flow-aggregate regulators in a DiffServ network. The port-based FA (PFA) is defined as a set of flows with the same class, which share the input and output ports in a relay node, such as a switch or router. If there are $N$ ports in the switch and $C$ classes, then there can be at most $N^2 \cdot C$ PFAs in the switch. There can be $N \cdot C$ PFAs in a single output port module, ignoring the fact that there is no flow with an output port that is the same as the input port. As the baseline DiffServ architecture has a limited number of classes and queues, fine-grained QoS provisioning is difficult, if not impossible, for example, such as meeting the various latency or jitter-bound requirements for various applications. Even in the same class, the required latency bounds can be different for different applications. In practice, the minimum of these bounds should be set as the target.

Now, consider only the flows and PFAs of the highest priority class to simplify the problem. Further, assume that the packets of lower-priority classes can be completely preempted by the highest-priority packets. Subsequently, the entire system, for the highest priority traffic, can be thought of as a single-class FIFO queuing and scheduling system. There are $N$ PFAs in the output module of the switch. Packet preemption is considered a critical and necessary function in deterministic networks and has been standardized in the IEEE TSN [3].

We propose adding a regulator for each high-priority PFA in an output port module, just before the class-based queuing/scheduling system of the output port module. We call this regulator the port-based flow-aggregate regulator (PFAR). The PFAR sees a PFA as a single flow with the parameters $\{\Sigma_i\sigma_i \ \& \ \Sigma_i a_i\}$, where $\sigma_i$ is the maximum initial burst and $a_i$ is the initial arrival rate of flow $i$, which is an element of the PFA, and regulates the PFA to meet the parameters. Using the initial parameter of a flow, we indicate the parameter of a flow at the source as it generates the flow according to the traffic specification (TSPEC) negotiated and reserved through an admission process defined in the DiffServ framework. The PFARs can be placed at the output port of a DiffServ switch to regulate such traffic. Figure 1 depicts an example architecture of a switch with the proposed regulators within a DiffServ-based switch.

This example architecture is similar to that suggested in the IEEE ATS, except that in the ATS, the IRs are placed instead of the PFARs. By aggregating flows in such a manner, the complexity of the regulation is reduced. In ATS, two factors contribute to implementation difficulty. First, it must identify the flow to which the packet belongs to the HOQ. The current state of the flow is then retrieved, and the time required to send the packet can be determined. Second, the individual flow state must be maintained to be able to determine the eligible time of a packet. Both impact the real-time packet processing performance. With the PFAR, the HOQ flow identification process is unnecessary, and only hundreds of PFAs' states, instead of millions of flows' states, must be maintained at a switch. However, while the ATS IR benefits from the property that it does not increase the latency bound of the preceding FIFO system, the PFAR introduces additional latency in terms of the latency bound.

The proposed architecture, the PFAR, is also comparable to the FAIR architecture because the FAIR is a superset of the ATS in the sense that if the AD in the FAIR is set to be a single node, then the FAIR and the ATS become identical. However, the FAIR and PFAR are distinguishable in an important aspect: the regulator granularity, whether the regulation targets are flows or flow aggregates. The IR in the ATS system should maintain the individual flow states, which limits scalability. In contrast, the PFAR only needs to maintain the states of the PFAs and the aggregates based on the ports. While scheduling based on FAs is common, to the best of the authors' knowledge, this study is the first attempt at regulation based on FAs. We will investigate the performances of the ATS IR and PFAR architectures in the following subsections.

### B. NUMERICAL ANALYSIS OF PFAR

We first analyze the latency of a network with PFARs. PFAR, as a regulator, guarantees the output characteristic of the FA.

The PFAR itself introduces a latency-bound increment by the regulation. The PFAs' input burst into and output burst from the PFAR are different, causing a latency-bound increment. If the PFAR is minimal [22], then the latency bound within the PFAR is as follows:

$$D_f^{PFAR} \leq \{(B_{PFA}^{PFAR_{in}} - B_{PFA}^{PFAR_{out}})^+ + L\}/C, \tag{1}$$

where $D_f^{PFAR}$ is the latency in the PFAR of a packet belonging to a flow f within the PFA, $B_{PFA}^{PFAR\_in}$ and $B_{PFA}^{PFAR\_out}$ are the maximum bursts of the PFA into and out of the PFAR, respectively, $L$ is the maximum packet length of the PFA, and $C$ is the link capacity. $x^+$ denotes $\max(0, x)$. A minimal regulator [22] is a regulator that transmits packets as soon as they become eligible to leave according to the regulation rule, thereby guaranteeing a service curve with given parameters, maximum burst, and average rate. (1) can be easily proven with Figure 2 and the properties of the latency-rate (LR) servers [23]. In Figure 2, the *latency* of a FIFO queue, $L/C$, is different from the latency that is synonymous to delay and used elsewhere herein. *Latency* in italics is a notation used in the LR server mathematics, and it denotes the largest possible time taken to start the service after the backlog period has been initiated in a scheduler [23]. In the FIFO queue, *latency*

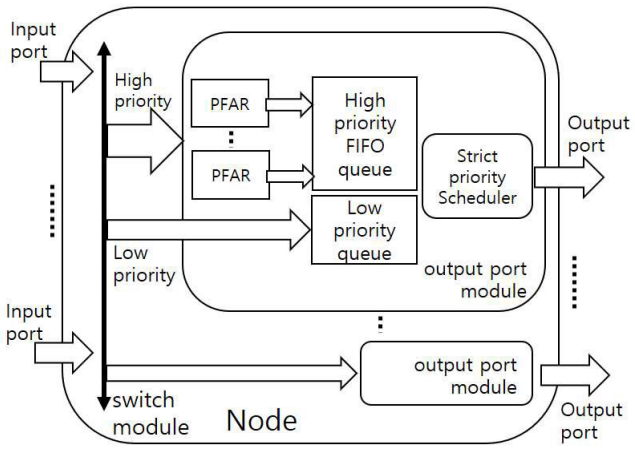


**FIGURE 1.** Example architecture of a switch with the proposed PFARs.

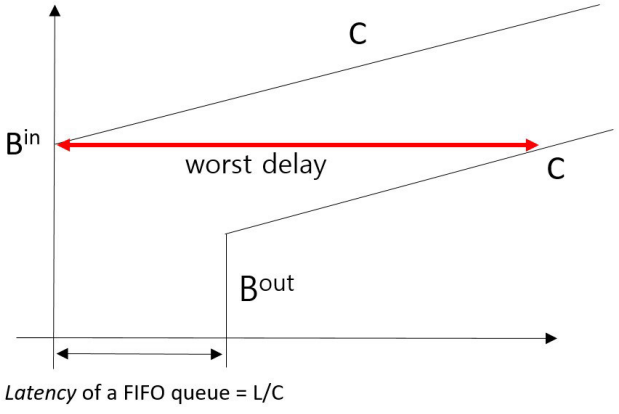


**FIGURE 2.** Calculation of maximum delay within a minimal PFAR with different input burst and output burst sizes.

can be seen as the transmission delay of the longest packet, which is inevitable in a store-and-forward system.

According to the regulation rule, $B_{PFA}^{PFAR\_out}$ in (1) is easily obtained because it is the sum of the initial bursts of flows within the PFA. $B_{PFA}^{PFAR\_in}$ in (1) can be calculated as the sum of the maximum bursts of the incoming flows into the PFAR. It can also be calculated as the sum of the maximum bursts of the incoming FAs from each input port into the PFAR. In cases where every node implements the PFAR, $B_{PFA}^{PFAR\_in}$ is obtained, as described in the next paragraph.

The next component of the latency within a switching node, after the latency within the PFAR, is due to the scheduler and its FIFO queues, which is a strict priority scheduler in the DiffServ frameworks. As we assume a perfect preemption of lower-priority packets, we can consider the scheduler as a FIFO scheduler with a single queue for packets from multiple input ports. It is also well known that the FIFO scheduler introduces latency bound as follows [23].

$$D_f^{FIFO} \leq (B_{FIFO}^{FIFO\_in})/C, \tag{2}$$

where $B_{FIFO}^{FIFO\_in}$ is the total maximum burst in the FIFO queue. Moreover, the maximum burst of a FA out of a FIFO scheduler is given as follows [23]:

$$B_{PFA}^{FIFO\_Out} \leq B_{PFA}^{FIFO\_in} + \left(\sum_{i \notin PFA} B_i^{FIFO\_in}\right) a_{PFA}/C, \tag{3}$$

where $B_{PFA}^{FIFO\_in}$ is defined as the maximum burst of the PFA as it enters the FIFO queue, and $a_{PFA}$ is the sustainable arrival rate of the PFA. Within our framework, $B_{PFA}^{FIFO\_in}$ can be easily obtained from the initial flow parameters because the PFA just passed through the PFAR before joining the FIFO scheduler. In other words, $B_{PFA}^{FIFO\_in} = B_{PFA}^{PFAR\_out}$. $B_i^{FIFO\_in}$, where $i \notin PFA$, can also be similarly obtained because $i$ in (3) can be seen as the FAs other than the PFA under observation. We can easily see that any subset of a PFA satisfies the following:

$$B_g^{FIFO\_Out} \leq B_{PFA}^{FIFO\_out}, \text{for any} g \subset PFA. \tag{4}$$

(4) implies that the maximum burst of any subset of a PFA out of a switching node, as it is divided into a number of output ports at the next node, can be obtained from (3) and (4). $B_{PFA}^{PFAR\_in}$ in (1) can be obtained from (3) and (4) under the condition that all upstream nodes implement the PFARs.

### C. CASE STUDY

We consider a simple network topology with a cycle dependency. It has four nodes connected in a ring topology, four flows under observation traveling two hops, and cross traffic, as shown in Figure 3. The flows under observation are colored red, green, brown, and blue in Figure 3. In contrast, the black flows in Figure 3 are considered as cross traffic. All flows were of high priority.

A flow under observation enters the network, joins another flow from a different input port of the node, travels to the next node, joins another flow from a different input port of the node, travels to the next node, and finally departs the network.

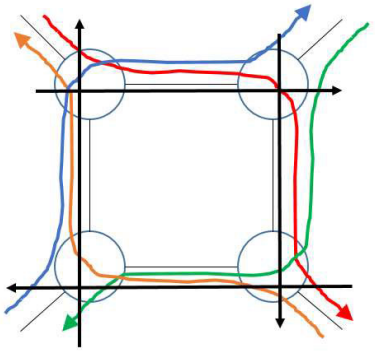

**FIGURE 3.** The network for the case study.

**TABLE 1.** Network and flow parameters used in the case study.

| Symbol | Meaning | Value |
|---|---|---|
| n | Number of microflows in the flow under observation | Variable |
| $a_f$ | Arrival rate of the flow under observation | n·480Kbps |
| $a_{CT}$ | Arrival rate of the cross traffic | ~0 |
| $B_f$ | Max burst of the flow under observation | n·2400bit |
| $B_{CT}$ | Max burst of the cross traffic | $= B_f$ |
| $L$ | Maximum packet length | 2400 bit |
| $C$ | Link capacity | 100 Mbps |

We compare the delay bounds of the flows under observation, in cases where there are 1) no regulators, 2) the ATS IRs at every node, and 3) the proposed PFARs at every node.

Table 1 summarizes the symbols and their values used in the numerical analysis. The flows under observation can be seen as PFAs, in each of which microflows exist. These microflows have identical parameters. Four PFAs, or flows under observation, also have the same parameters.

The IR, attached next to a FIFO system, has the following properties: as long as the following conditions are met, the combined system of the IR and FIFO has the same latency bound as the FIFO system alone.

(a) Every flow into the FIFO system conforms to an arrival curve with parameters {average arrival rate, maximum burst size}.

(b) The FIFO system guarantees FIFO for all packets.

(c) The IR regulates every flow with its initial parameters at the ingress of the FIFO system.

(d) (Minimal IR) The IR transmits immediately when the packet at the HOQ becomes eligible to leave according to the regulation rule. Such an IR is called the minimal IR.

(e) The IR should be able to cut through to transmit eligible packets at the instance they become eligible. For example, if a packet enters the IR when the queue is empty and the packet

is already eligible, the IR should be able to cut through the packet.

The latency bounds with the ATS IRs are obtained as follows: The FIFO queue and scheduler in the DiffServ system are known to satisfy conditions (a) to (d) [22]. Regarding condition (e), if cut-through is not available, then the maximum transmission delay of a packet L/C should be added to the latency bound in a node. Assuming that all these conditions can be met in the network in Figure 3, we can only count the latency within the FIFO scheduler for the case with the ATS IR. Furthermore, with the ATS IR, each individual flow is shaped at every node with its initial traffic parameters. As such, the end-to-end latency bound of a flow under observation can be obtained by the following equation [23]:

$$D_f^{E2E} \leq \frac{B_f^{initial}}{r} + \Theta_f^{1st\ node} + \Theta_f^{2nd\ node} + \frac{L}{C}, \qquad (5)$$

where $B_f^{initial}$ is the initial maximum burst, $\Theta_f^{1st\ node}$ is the *latency* at the 1st node, and L/C is the maximum transmission delay at the last node of the flow under observation. $r$ is the guaranteed service rate to the flow, which is C/2 in this case, because there are two colored flows with the same average rate in a link. Again, the *latency* in italic font is a notation used in the LR server mathematics, and it denotes the time to start the service after the backlog period initiation in a scheduler. The black flows in Figure 3 are cross traffic with an instantaneous burst with a negligible average rate. The latency of a FIFO scheduler is the total burst of the FIFO divided by the guaranteed service rate, which in this case is the link capacity. (5) takes into account that a protected flow gets the benefit of the ''pay burst only once''.

The latency bounds without any regulators can be obtained using the total flow analysis (TFA) algorithm suggested in [16]. In summary, the TFA algorithm works as follows:

- cuts an arbitrary link of a cycle,
- applies initial burst parameters of the flows at the start of the cut,
- calculates the bursts of the flows at the end of the cut (called the end bursts),
- applies the end bursts as new initial bursts,
- calculates the end bursts again,
- and repeats the process until the end bursts converge.

For example, if we cut the top link of the network in Figure 3, the red and blue flows get the cuts. These flows, starting from the cut point, are considered to have initial flow parameters. The bursts of these flows were calculated as they traverse the nodes. The burst of red flow affects the burst of green flows, and so on. At the upper-left node's output port, the burst of the blue flow affects the burst of the red flow. The maximum burst value of the red flow replaces the initial value, and the second round of calculation begins. As the rounds continue, the maximum burst value may converge. Subsequently, the latency bounds can be obtained based on the converged end bursts.

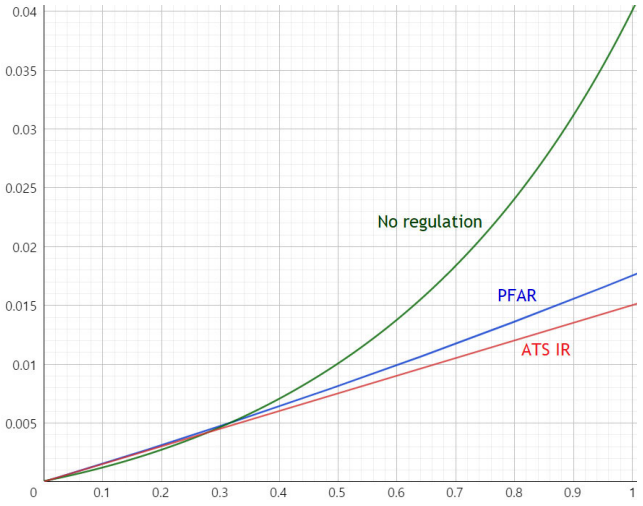


**FIGURE 4.** The latency bounds of three regulation strategies in the case study, as the link utilization varies from 0 to 1.

A colored flow can be considered a FA, a set of homogeneous flows with the same network ingress and egress. It actually becomes a single PFA in a node because it is the only flow with such input and output ports. A flow that is an element of a colored FA with the specified parameters is called a command and control (C&C) flow in Ethernet-based automotive networks [24], [25]. These C&C flows usually have the highest priority over audio or video flows. The number of flows in a PFA, n, is a critical factor for the utilization of the links. Figure 4 depicts the latency upper bounds of the three regulation strategies as functions of link utilization.

Without regulation, the latency bound increases quadratically, while with either regulator at every node, it increases linearly. In addition, with low utilization, the regulator may adversely affect the latency bound. However, when the links are highly utilized, in which the latency-bound value is crucial, the PFAR and ATS IR show much better performance. Moreover, the performance of PFAR was comparable to that of ATS. Considering the low complexity of the PFAR, this result is encouraging.

Now, we consider the more complex network depicted in Figure 5. The network in Figure 5 has nine switching nodes and 12 links. They formed four inner circles. It is roughly twice as large in terms of the network diameter, compared to the one in Figure 3.

Consider four imaginary circles in the network, each placed in 1st, 2nd, 3rd, and 4th quadrants, which rotate clockwise or counterclockwise, as depicted in Figure 5. We can construct symmetrical flows such that every link has two flows with the burst accumulated by each other, as in the flows in Figure 3. The rules for the construction are as follows. First, after a flow enters the network, it follows the direction of the imaginary circle. Second, a flow traverses exactly two hops in a circle, then enters another circle or leaves the network. For example, a flow may hop over nodes 1, 2, 5, 6, and 9. This flow is denoted as f(12569). Similarly, f(32547), f(98541),

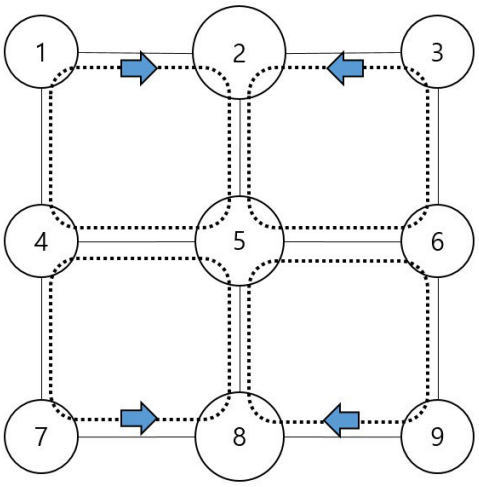


**FIGURE 5.** Second network for the case study with nine nodes.

and f(412) can be constructed. These four flows encircle the 1st quadrant in the upper-left corner of the network. The three other quadrants were also encircled similarly.

The constructed network has exactly two flows in a link, as shown in the network in Figure 3. As such, we can use the parameters in Table 1 to be unaltered. The only difference was in the number of hops of the flows. For example, the hop count of flow f(12569) is four, which is our flow of interest. The graphs in Figure 6 are obtained for the flow of interest using the same procedures used for the graphs in Figure 4. For the analysis of the network with no regulation, the TFA with initial cuts has to be used [16]. In our network, the cuts are made at the links (1,4), (3,6), (4,7), and (6,9).

The networks in Figure 3 and 5, and their performance graphs depicted in Figures 4 and 6 show the trend of the latency bounds as the network grows. We can observe that the no-regulation network becomes more unstable as the network grows. The latency-bound difference between the PFAR and ATS IR architectures also becomes more profound as the network grows. The advantage of the "pay burst only once" characteristic in the ATS IR system is clearer when there are more hops in the flow path. However, the latency bound of the PFAR remained stable with varying utilization. Again, considering the low complexity of PFAR, it is efficient and effective in larger networks as well. While case studies use simple and symmetric networks, they capture the essential characteristics of the regulators as cycle breakers. Such a core network with several cycles, combined with edge networks with tree topologies, would form a network comparable to a common metro area network.

## III. GENERALIZED JITTER BOUND GUARANTEE

### A. PREVIOUS WORK ON JITTER BOUND GUARANTEE WITH ASYNCHRONOUS FRAMEWORK

In previous work, a framework for guaranteeing the jitter bound in arbitrarily sized networks with any type of topology with random dynamic input traffic has been considered [21]. The jitter is defined as the latency difference between two


**FIGURE 6.** Latency bounds of three regulation strategies in the case study with the network with nine nodes, as the link utilization varies from 0 to 1.

packets within a flow, not a difference from a clock signal or from an average latency, as summarized in RFC 3393 [2].

The framework in the previous work is composed of

- a network that guarantees latency upper bounds;
- a timestamper for packets with a clock that is not necessarily synchronized with the other nodes, which resides in between, including the source and the network ingress interface;
- and a buffer that can hold the packets for a predetermined interval, which resides in between, including the destination and the network egress interface.

Figure 7 depicts the overall architecture of the BN framework for jitter-bound guarantees. Only a single flow is depicted between the source and destination, as shown in Figure 7. The arrival ($a_n$), departure ($b_n$), and buffer-out ($c_n$) instances of the $n^{th}$ packet of a flow are denoted. The end-to-end (E2E) latency and the E2E buffered latency are defined as ($b_n$-$a_n$) and ($c_n$-$a_n$), respectively.

The buffer supports as many as the number of the flows destined for the destination. In cases where the buffer is not suitable to be placed within an end station, the network can attach a buffering function at the boundary. The destination shown in Figure 7 can also be a small deterministic network. There is an entity for time-stamping arrival instances to the packets. The time-stamping function may be used in the real-time transport protocol (RTP) over the user datagram protocol or the transmission control protocol (TCP). Either the source or network ingress interface can stamp the packet. In the case where the source stamps, the timestamp value is the packet departure instance from the source, which is only a propagation time away from the packet arrival instance to the network. The source and destination do not need to share a synchronized clock. All we need to know is the differences

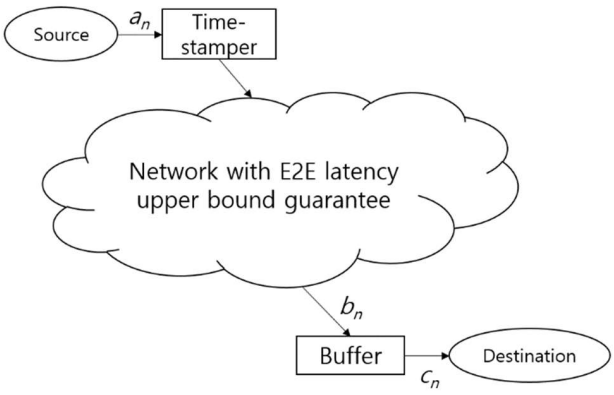


**FIGURE 7.** Buffered network (BN) framework for jitter bound guarantee [21].

between the time stamps, that is, the information about the relative arrival instances.

The latency upper bound of the flow is assumed to be guaranteed by the network. There are several candidate architectures for latency guarantee, including the PFAR architecture suggested in Section II. The IEEE ATS network can also operate. The FAIR architecture specified in ITU-T Y.3113 is another framework for the latency upper bound guarantee [19], [20]. The three solutions have their own advantages and disadvantages. While the PFAR is most scalable, the ATS has an advantage in performance and can be applied to modest-sized networks. The FAIR has a greater degree of freedom in the design parameter configuration such as the decision criteria for ADs and flow aggregates. When such parameters are carefully configured, FAIR performs best among the three solutions.

In the BN framework in Figure 7, it is recommended that the latency-lower bound information be provided by the network. The lower bound may be attributed to transmission and propagation delays within the network. The buffer holds packets in a flow according to the predefined intervals. The decision of the buffering intervals involves the timestamp within each packet.

Let the arrival instance of the $n^{\text{th}}$ packet of a flow be $a_n$. Similarly, let $b_n$ be the departure time from the network of the $n^{\text{th}}$ packet. Then, $a_1$ and $b_1$ are the arrival and departure instances of the first packet of the flow, respectively. The first packet of a flow is defined as the first packet generated by the source, among all packets that belong to the flow. Further, let $c_n$ be the buffer-out instance of the $n^{\text{th}}$ packet of the flow.

Let us define m as the jitter control parameter, which has a value larger than W and $W \leq m$. Parameter m directly controls the holding interval of the first packet. It plays a critical role in determining the jitter and latency upper bounds of a flow in a BN framework. The larger the m, the smaller the jitter bound, and the larger the latency bound. With a sufficiently large m, we can guarantee zero jitter at the cost of an increased latency bound.

The rules for the buffer-holding interval decision are given as follows:

- The buffer holds the first packet with an interval (m - W) for some m, $W \leq m$. The buffer-out instance of the first packet $c_1$ is then $(b_1 + m - W)$. In other words,

$$c_1 = (b_1 + m - W). \tag{6}$$

- The buffer holds the $n^{\text{th}}$ packet until the instance $\max\{b_n, c_1 + (a_n - a_1)\}$, for any $n > 1$. In other words,

$$c_n = \max\{b_n, c_1 + (a_n - a_1)\}, \text{ for } n > 1 \tag{7}$$

The second rule implies that a packet should be held in the buffer to make its inter-buffer-out time $(c_n - c_1)$ equal to the inter-arrival time $(a_n - a_1)$. However, when its departure from the network is too late, the inter-buffer-out time should be larger than the inter-arrival time, then just passes the buffer, that is, $c_n = b_n$. The buffer does not need to know the exact values of $a_n$ or $a_1$. It is sufficient to determine the difference between these values, which can be easily obtained by subtracting the timestamp values of the two packets.

Let us summarize the symbols used in this section, including the following subsections in Table 2.

The following theorems hold [21].

*Theorem 1 (Upper Bound of E2E Buffered Latency):* The latency from the packet arrival to the buffer-out instances $(c_n - a_n)$, is upper bounded by (m+U-W).

*Theorem 2 (Lower Bound of E2E Buffered Latency):* The latency from the packet arrival to the buffer-out instances $(c_n - a_n)$, is lower bounded by m.

*Theorem 3 (Upper Bound of Jitter):* The jitter is upper bounded by (U-m).

The jitter between packets $i$ and $j$ is defined as $|(c_i - a_i) - (c_j - a_j)|$. The proofs for the theorems can be found in [21]. The three theorems state that with the framework, any desired jitter bounds, including zero jitter, can be achieved by adjusting the parameter m, while still guaranteeing a latency bound.

**TABLE 2.** Mathematical symbols for the jitter bound framework.

| Symbol | Quantity |
|---|---|
| $a_n$ | Arrival instance of the $n_{\text{th}}$ packet of the flow to the network |
| $b_n$ | Departure instance of the $n_{\text{th}}$ packet of the flow from the network |
| $c_n$ | Buffer-out instance of the $n_{\text{th}}$ packet of the flow |
| $b_n - a_n$ | E2E latency of the $n_{\text{th}}$ packet |
| $c_n - a_n$ | E2E buffered latency of the $n_{\text{th}}$ packet |
| U | E2E latency upper bound of the flow guaranteed by the network |
| W | E2E latency lower bound of the flow guaranteed by the network |
| m | Jitter control parameter of the framework, $W \leq m$. |
| $g_n$ | Processing delay within the buffer of the $n_{\text{th}}$ packet of the flow. It includes store/lookup/forward delay. |
| $g$ | Maximum $g_n$ over $n$. |

### B. STORE AND FORWARD BUFFER

However, two rules (6) and (7) assume that the buffer is capable of cut-through, as it may be required to be $c_n = b_n$, for n $\geq$ 1, that is, the buffer-out instance should be the same as the network departure instance.

In this paper, we generalize Theorems 1 to 3 so that they still hold even with buffers without cut-through capability. Now, let $g_n$ be the processing delay within the buffer of the $n^{\text{th}}$ packet of the flow. The $g_n$ includes the time to look up the timestamp and to store/forward the packet. It does not include an intentional buffer-holding interval. By definition, $c_n - b_n \geq g_n$. Let $g_n = g$, the maximum processing delay in the buffer. It is assumed that a buffer can determine the value of $g$. We then revise the buffer-holding rules as follows:

$$c_1 = (b_1 + \mathrm{m} - \mathrm{W}). \tag{8}$$

$$c_n = \max\{g + b_n, c_1 + (a_n - a_1)\}, \text{ for n } > 1. \tag{9}$$

Furthermore, $m$ is required to meet the inequality

$$\mathrm{m} \geq \mathrm{W} + g. \tag{10}$$

By revising (7) to (9), and satisfying condition (10), we guarantee that $c_n \geq g_n + b_n$, for every $n$ including $n = 1$, because $c_1 = m - W + b_1 \geq g_1 + b_1$, and $c_n \geq g + b_n \geq g_n + b_n$, for n>1. Therefore, the cut-through capability of the buffer was not required. The following revised Theorems 1' to 3' hold:

*Theorem 1' (Upper Bound of the E2E Buffered Latency):* If m $-$ W $\geq$ $g$ then by following rules (8) and (9), the latency from the packet arrival to the buffer-out instances $(c_n - a_n)$, is upper bounded by $(U - W + m)$.

*Proof:* For any n$\geq$1, $b_n - a_n \leq U$, m $-$ W $\geq$ $g$. For n > 1,

$$\begin{aligned} c_n - a_n &= \max\{b_n + g, c_1 + (a_n - a_1)\} - a_n \\ &= \max\{b_n - a_n + g, b_1 + m - W - a_1\} \\ &\leq \max\{U + g, U + m - W\} = U - W + m. \end{aligned}$$

For $n = 1$, $c_1 - a_1 = b_1 + \mathrm{m} - \mathrm{W} - a_1 \leq U - W + m$. ■

*Theorem 2' (Lower Bound of the E2E Buffered Latency):* If m $-$ W $\geq$ $g$ then by following rules (8) and (9), the latency from the packet arrival to the buffer-out instances, $(c_n - a_n)$, is lower bounded by m.

*Proof:* For any $n \geq 1$, $b_n - a_n \geq W$, m $-$ W $\geq$ $g$. For $n > 1$,

$$\begin{aligned} c_n - a_n &= \max\{b_n + g, c_1 + (a_n - a_1)\} - a_n \\ &= \max\{b_n - a_n + g, b_1 + m - W - a_1\} \\ &\geq \max\{W + g, W + m - W\} = m. \end{aligned}$$

For $n = 1$, $c_1 - a_1 = b_1 + \mathrm{m} - \mathrm{W} - a_1 \geq W + m - W = m$. ■

*Theorem 3' (Upper Bound of Jitter):* By following rules (8) and (9), the jitter is upper bounded by $(\mathrm{U} + \mathrm{g} - \mathrm{m})^+$.

*Proof:* The jitter between the $i^{\text{th}}$ and $j^{\text{th}}$ packets is defined as $r_{ij} = |(c_i - a_i) - (c_j - a_j)|$. Let us further define the "jitter of the $i^{\text{th}}$ packet" $r_i$ such that $r_i = r_{i1} = |(c_i - a_i) - (c_1 - a_1)|$. Then for any n>1,

$$\begin{aligned} r_n &= (c_n - c_1) - (a_n - a_1) \\ &= \max\{b_n + g, c_1 + (a_n - a_1)\} - c_1 - a_n + a_1 \\ &= \max\{b_n + g - c_1 - a_n + a_1, 0\} \\ &= \max\{b_n + g - (b_1 + \mathrm{m} - \mathrm{W}) - a_n + a_1, 0\} \\ &= \max\{(b_n - a_n) - (b_1 - a_1) + (g - m + W), 0\} \\ &\leq \max\{U - W + (g - m + W), 0\} \\ &= (\mathrm{U} + \mathrm{g} - m)^+, \text{ as} \\ \mathrm{U} &\geq b_n - a_n \geq W \text{ for n } \geq 1. \end{aligned}$$

$$\begin{aligned} \text{For any } i,j \geq 1,\ r_{ij} &= \left|(c_i - a_i) - \left(c_j - a_j\right)\right| \\ &= \left|(c_i - c_1) - (a_i - a_1) - \left(c_j - c_1\right) - \left(a_j - a_1\right)\right| \\ &= \left|r_i - r_j\right| \leq (\mathrm{U} + \mathrm{g} - \mathrm{m})^+, \text{ since } 0 \leq r_i, \\ r_j &\leq (\mathrm{U} + \mathrm{g} - \mathrm{m})^+. \end{aligned}$$

■

By setting m = (U + g), we can achieve zero jitter. In this case, the E2E buffered latency bound becomes (2U + g $-$ W), which is roughly twice the E2E latency bound. In contrast, if we set $m$ to its minimum possible value W, then the jitter bound becomes (U + g $-$ W), which is roughly equal to U, while the E2E buffered latency bound becomes U, which is the same as the E2E latency bound.

### C. CLOCK DRIFT

Clock drift refers to phenomena wherein a clock does not run at exactly the same rate as a reference clock. As we do not synchronize the clocks of different nodes in a network, clock drift is unavoidable. Consequently, jitter occurs owing to the clock frequency difference or clock drift between the source and the buffer. To compensate for this type of jitter, we propose an algorithm for adjusting the instances of the buffer events such that the gap between an event's occurrence times recorded with different clocks is bounded. We will call this gap the "clock difference". The proposed algorithm is free of message exchanges common in time-synchronization methods such as the NTP and PTP [8], [9]. Therefore, the algorithm is free from requirements such as a constant network delay or equal delays for both directions.

We consider only the problem of constant clock drift, that is, the buffer clock runs constantly faster or slower than the source clock. Without loss of generality, we define the source clock as the reference clock. We distinguish the ideal buffer clock and the actual buffer clock. The former is an imaginary clock that runs at the same rate as the reference clock, while the latter is the actual clock that runs in the buffer system. In addition, the reference clock and ideal buffer clock are frequency-synchronized but not time-synchronized. We denote $b_n$ as the instance measured with the ideal buffer clock, and $b_n^{\sim}$ as the instance measured with the actual buffer clock. Using this notation, the $b_n$ in the previous subsections is considered as the instance measured with the ideal buffer clock. Without loss of generality, we let $b_1 = b_1^{\sim}$ because the

buffer system can be seen as initialized at $b_1$. The E2E latency bound is not larger than U or smaller than W. We can use this fact to detect a faster or slower buffer clock by checking whether the measured E2E latency is larger than U or lower than W. However, the buffer cannot measure the E2E latency by simply looking at $b_n^{\sim} - a_n$ because the source clock and the buffer clock are not time-synchronized. As mentioned earlier, the buffer can only infer the time difference between two source events by subtracting the timestamp values recorded in the packet header.

Let us consider the case where the buffer clock is constantly faster than the source clock. The opposite case will be argued similarly. In this case, the actually measured instance always has a larger value than the ideal one, that is, $b_n^{\sim} \geq b_n$. For the detection of a faster clock, we use the fact that the difference between the E2E latencies of any two packets should not exceed U − W. The buffer can identify the differences in the arrival instances of the two packets measured with the reference clock, $(a_n - a_1)$, by subtracting the timestamp values recorded in the $n^{\text{th}}$ and $1^{\text{st}}$ packets. Therefore, we can detect a faster buffer clock by checking the inequality:

$$\left(b_n^{\sim} - a_n\right) - \left(b_1^{\sim} - a_1\right) = \left(b_n^{\sim} - b_1^{\sim}\right) - (a_n - a_1) > U - W. \tag{11}$$

If (11) is true, then the $n^{\text{th}}$ packet's E2E latency exceeds U, since the $1^{\text{st}}$ packet's minimum E2E latency is W, which is a contradiction to the fact that the E2E latency bound is U.

As an alternative, we can compare the smallest observed E2E latency, not the first packet's E2E latency, with the $n^{\text{th}}$ packet's E2E latency, that is, to check the inequality

$$\left(b_n^{\sim} - a_n\right) - \left(b_m^{\sim} - a_m\right) = \left(b_n^{\sim} - b_m^{\sim}\right) - (a_n - a_m) > U - W, \tag{12}$$

where $\mathrm{m} = \arg\min\,(b_i^{\sim} - a_i)$, $i \leq n$. As such, $\left(b_m^{\sim} - a_m\right)$ is the least observed E2E latency until $n$. To obtain m, we have to calculate $\left(b_i^{\sim} - b_1^{\sim}\right) - (a_i - a_1)$, for $i \leq n$, and see if it is the minimum. This could be a negative value.

When a packet $n$ is found for which (12) is true, we update $b_n^{\sim}$ to $b_n^{\sim} - k_n \stackrel{\text{def}}{=} b_n^*$. $k_n$ is the most conservatively estimated clock difference, which has a value

$$k_n = \left(b_n^{\sim} - b_m^{\sim}\right) - (a_n - a_m) - U + W. \tag{13}$$

Then,

$$\begin{aligned}\left(b_n^* - b_m^{\sim}\right) - (a_n - a_m) &= \left(b_n^{\sim} - k_n - b_m^{\sim}\right) - (a_n - a_m) \\ &= U - W. \end{aligned} \tag{14}$$

Similar arguments can be applied to the case where the buffer clock is constantly slower than the source clock. The difference between the E2E latencies of any two packets should not be smaller than that of W − U. Therefore, we detect the slower clock by checking

$$\begin{aligned}\left(b_n^{\sim} - a_n\right) - \left(b_M^{\sim} - a_M\right) &= \left(b_n^{\sim} - b_M^{\sim}\right) - (a_n - a_M) \\ &< W - U, \end{aligned} \tag{15}$$

**Algorithm 1** Clock Drift Compensation Algorithm

**Input:** $a_n$, $b_n$
**Output:** $b_1$
1: if n = 1 then
2: $a_m \leftarrow a_n$
3: $b_m \leftarrow b_n$
4: $a_M \leftarrow a_n$
5: $b_M \leftarrow b_n$
6: $a_{n-1} \leftarrow a_n$
7: $b_{n-1} \leftarrow b_n$
8: $b_1 \leftarrow b_n$
9: else
10: if using the smallest or largest observed latency then
11: if $b_m - a_m > b_{n-1} - a_{n-1}$ then
12: $a_m \leftarrow a_{n-1}$
13: $b_m \leftarrow b_{n-1}$
14: else if $b_M - a_M < b_{n-1} - a_{n-1}$
15: $a_M \leftarrow a_{n-1}$
16: $b_M \leftarrow b_{n-1}$
17: end if
18: end if
19: Compute $\alpha_m = a_n - a_m$, $\beta_m = b_n - b_m$, $\alpha_M = a_n - a_M$, $\beta_M = b_n - b_M$
20: if $\beta_m - \alpha_m >$ U − W then
21: $b_1 \leftarrow b_1 + (\beta_m - \alpha_m - (U - W))$
22: else if $\beta_M - \alpha_M <$ W − U then
23: $b_1 \leftarrow b_1 + (\beta_M - \alpha_M - (W - U))$
24: else $b_1 \leftarrow b_1$
25: end if
26: $a_{n-1} \leftarrow a_n$
27: $b_{n-1} \leftarrow b_n$
28: end if
29: return $b_1$

where M= $\arg\max\,(b_i^{\sim} - a_i)$, $i \leq n$. When a packet $n$ is found for which (15) is true, we update $b_n^{\sim}$ to $b_n^{\sim} - k_n \stackrel{\text{def}}{=} b_n^*$. $k_n$ is again the most conservatively estimated clock difference, which has the value

$$k_n = \left(b_n^{\sim} - b_M^{\sim}\right) - (a_n - a_M) - W + U. \tag{16}$$

Then

$$\begin{aligned}\left(b_n^* - b_M^{\sim}\right) - (a_n - a_M) &= \left(b_n^{\sim} - k_n - b_m^{\sim}\right) - (a_n - a_m) \\ &= W - U. \end{aligned} \tag{17}$$

The process described so far is summarized in Algorithm 1.

As in line 10 of Algorithm 1, one can choose to compare the current packet's latency with the smallest or the largest observed E2E latency, or simply with the E2E latency of the first packet. The former approach is expected to yield better compensation at the cost of the additional complexity from tracking the minimum and maximum values of the E2E latency.

We suggest Algorithm 1 to be added to the buffering process and $b_n^{\sim}$ to be updated to $b_n^*$ for every $n$ if necessary,

such that the clock difference can be bounded according to Theorem 4.

*Theorem 4 (Clock Difference Bound):* Let us define the clock difference as $|b_n^* - b_n|$. If the clock drift is constant over time, according to Algorithm 1, the clock difference is upper bounded by 2(U − W).

*Proof:* The proof shows that the Theorem holds for both cases.

Case 1: The buffer clock is faster than the source clock.

From (14),

$$\left(b_n^* - a_n\right) = \left(\tilde{b_m} - a_m\right) + U - W \leq 2U - W,$$

because the measured E2E latency $\left(\tilde{b_m} - a_m\right)$ of packet m was smaller than U. As for all n, $b_n^* \geq b_n$, the clock difference

$$\begin{aligned}\left|b_n^* - b_n\right| &= \left(b_n^* - b_n\right) \\ &= \left(b_n^* - a_n\right) - \left(b_n - a_n\right) \leq \left(b_n^* - a_n\right) - \mathrm{W} \\ &\leq 2\mathrm{U} - \mathrm{W} - \mathrm{W} = 2(\mathrm{U} - \mathrm{W}).\end{aligned}$$

Case 2: The buffer clock is slower than the source clock.

From (17),

$$\left(b_n^* - a_n\right) = \left(\tilde{b_M} - a_M\right) + W - U \geq 2W - U$$

because the measured E2E latency $\left(\tilde{b_M} - a_M\right)$ is larger than that of W. As for all n, $b_n^* \leq b_n$, the clock difference

$$\begin{aligned}\left|b_n^* - b_n\right| &= \left(b_n - b_n^*\right) \\ &= \left(b_n - a_n\right) - \left(b_n^* - a_n\right) \leq \mathrm{U} - \left(b_n^* - a_n\right) \\ &\leq \mathrm{U} - 2\mathrm{W} + \mathrm{U} = 2(\mathrm{U} - \mathrm{W}).\end{aligned}$$

■

## IV. IMPLEMENTATION OF JITTER BOUND GUARANTEED NETWORK

### A. IMPLEMENTATION OVERVIEW

In this section, we consider the problem of whether zero jitter can be achieved in real implementations. A simple embedded system with the techniques described in the previous section was used. To achieve a near-zero jitter upper bound in such an implementation, the following three conditions must be considered.

First, the source and buffer should ideally be frequency-synchronized. Otherwise, jitter is added as much as the cumulative clock difference during the flow lifetime. To compensate for the clock difference, $a_n$ or $b_n$ should be updated periodically or at certain events, as suggested in Section III. The proposed method guarantees the upper bound of the clock difference.

Second, the transmission rate should ideally be constant. This is to prevent jitter due to the transmission delay difference in packets having the same length. The link connecting the buffer and the destination must satisfy this condition.

Third, the buffer should ideally be capable of simultaneous transmission and reception. As an alternative, the buffer should give a higher priority to the transmission process. To meet the buffer-out instance requirement suggested by rules (8) and (9), transmitting a packet from the buffer even while receiving another is necessary. If simultaneous transmission and reception are not possible, jitter is added as much as the reception delay. One should ensure that the reception process does not affect the transmission process. As explained in the next subsections, we take the approach to separate the transmission and reception processes by adding another buffer. Note that it is not related to the double-buffering technique standardized in IEEE 802.1Qch CQF.

We used two XMOS xCORE-200 eXplorerKITs, as shown in Figure 8. The xCORE-200 eXplorerKIT is an evaluation board of the xCORE-200 multicore MCU and is a platform used in the fields of network and audio [26]. It is a programmable device that can execute the code created on a PC through a debug adaptor on the board. The full-duplex Ethernet interface was supported. The transmission rate of the Ethernet interface can be set to 10 Mbps, 100 Mbps, or 1 Gbps. The experiments in this study were carried out at a transmission rate of 1 Gbps.

The xCORE-200 multicore MCU comprised two tiles. Each tile comprised eight logical cores. Each core is capable of executing a functional process. The parallel architecture of the MCU makes multiple concurrent tasks almost deterministically performed. The communication between tiles is implemented as a physical circuit and has a constant speed. It has been confirmed that the communication between the logical cores also has a constant speed. There is memory for each tile, so it is used to store variables or constants in the code and create a buffer. The clock frequency is 100 MHz; therefore, 10 ns is the minimum controllable time unit.

As the xCORE-200 eXplorerKIT is a programmable device, it does not support cut-through. We first implemented it to put all packets into the buffer. As we have proved in Section III.B, it is still possible to guarantee a jitter upper bound with the store-and-forward buffers by applying rules (8) and (9).

**FIGURE 8.** Experiment with two xCORE-200 eXplorerKITs.

### B. IMPLEMENTATION OF THE IDEAL SITUATION

We considered two different operation scenarios: an ideal situation and a realistic situation. The ideal situation comprises two nodes, as shown in Figure 9, and the packets move two hops: from node 1 to node 2 and then from node 2 to node 1. Each node was implemented with the xCORE-200 eXplorerKIT device, as described above. Each node comprises an Ethernet interface and processes such as the source, destination, buffer, or network. The Ethernet interface transmits or receives packets through an LAN cable. The Ethernet interface is implemented in a separate tile and has its own buffer. As the tiles are separated, they are not affected by other processes. The buffer may be implemented either at the end of the network or at the destination. Implementing the buffer at the end of the network has the advantage of reducing the burden on the destination node, which may have limited processing capability. To examine whether the buffer could be implemented at the end of the network, we measured the clock frequencies of the two devices. The clocks of the two systems drift from each other by 6 $\mu$s per second, which is considered significant. Therefore, the buffer is implemented in the same device as the source and destination to share the clock.

The source process sends a burst of 20 RTP packets, each having a fixed 250-byte length every 5 ms. The inter-arrival time between packets within a burst is set to 16 $\mu$s. RTP packet lengths could also be set at the bit level. We recorded the departure time in the timestamp field of the RTP header. The timestamp is 32 bits long in 10-ns units, which is the same as the system clock period.

The network process loopbacks the received packets. The transmitter, receiver, and Ethernet interface of the network process were implemented separately. The packets experience only a small delay in the network process. The destination process calculates the E2E buffered latency of a packet ($c_n - a_n$) by subtracting the reception time of the packet and the timestamp recorded in the packet. For an exact latency calculation, the clock of the destination should be synchronized with the clock of the source. In this ideal situation, because the source and destination are implemented in the same device, the clock is shared and thus synchronized.

The buffer process puts the received packet into the buffer, calculates the instance to transmit to the destination process, and transmits the packet at that instance. The packet transmission instance is determined according to rules (8) and (9). The upper bound (U) and lower bound (W) of the E2E latency of the network are not the objects of interest in the implementation. U + g is set to 200 $\mu$s, which is sufficiently large for the Ethernet interfaces and network process. W is set to 0 us.

As the system is on a programmable device, simultaneous transmission from and reception into the buffer is not possible. Therefore, to achieve zero jitter, the transmit and receive buffers are implemented separately. The functions of the buffer process include the reception of packets, calculation and decision, packet storage, and packet transmission. The receive buffer has reception and calculation/decision functions. After the receive buffer receives a packet, it decides the transmission instance of the packet and sends the packet and the transmission instance information to the transmit buffer. The transmit buffer stores the packet and starts the transmission at a time the calculated transmission instance minus the transmission delay of the packet. The packet transmission delay, which is several tens of nanoseconds in this experiment, is obtained based on the packet length and link capacity.

For performance comparison, the case of using only a single buffer was also implemented. In this case, additional jitter may occur because the delay at the reception process affects the transmission.

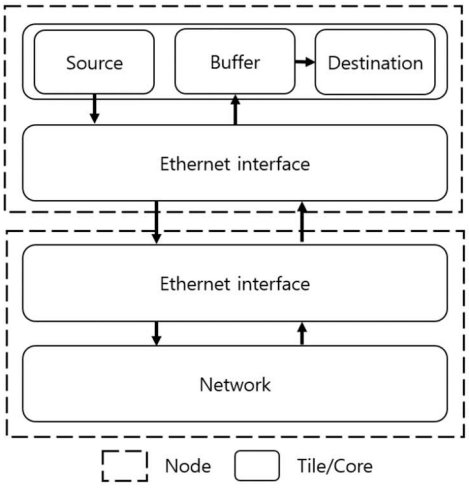


**FIGURE 9.** Processes and the topology in the ideal situation.

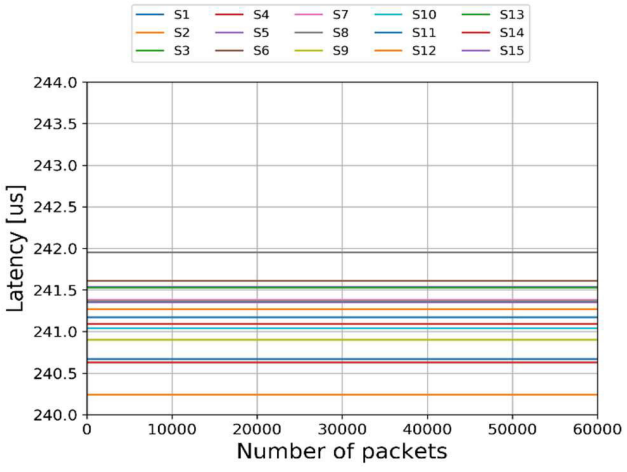


**FIGURE 10.** E2E buffered latency in the ideal situation with the dual buffer.

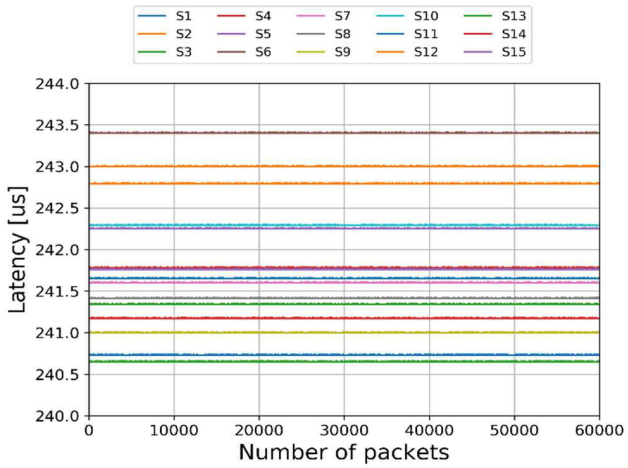


**FIGURE 11.** E2E buffered latency in the ideal situation with the single buffer.

### C. RESULTS IN THE IDEAL SITUATION

Node 1 performs the source, buffer, and destination processes, and node 2 performs the network process. The buffer process was implemented using either a dual buffer or a single buffer. A single experiment was conducted for 15 s. During 15 s, we generated 3000 bursts, each with 20 packets, thus obtaining 60,000 end-to-end latency values. The experiment was repeated 15 times. Rules (8) and (9) were applied.

With the dual buffer, we achieved a jitter of 0 $\mu$s, as shown in Figure 10, and the average latency was 240–242 $\mu$s. The single-buffer implementation has a jitter of 10 ns, as shown in Figure 11, with an average latency of 240.5–243.5 $\mu$s. A jitter of 10 ns is at the clock period level, which is the minimum possible jitter that can occur considering the 100-MHz clock frequency of xCORE-200. Even with different upper bound values, U, of the end-to-end latency of the network, the dual buffer still achieved zero jitter, and the single buffer achieved 10 ns of jitter.

### D. IMPLEMENTATION OF THE REALISTIC SITUATION

Zero jitter can be achieved in an ideal situation. In this subsection, we consider a more realistic situation. First, by implementing the source and buffer in different devices, we emulated a situation with clock drift. The clock drift compensation algorithm described in Section III.C was applied. Second, in addition to the clock drift, the packets have experienced random delays in the network; thus, they behave similarly to a real network.

A realistic situation comprises two nodes, as shown in Figure 12. The packets move a single hop from node 1 to node 2. Each node was implemented with the xCORE-200 eXplorerKIT device, as described above. Each node comprises an Ethernet interface and processes such as the source, destination, buffer, or network. Node 1 runs the source and network processes, and node 2 runs the buffer and destination processes.

The network process generated random delay values and applied them to the incoming packets in the range of 50–500 $\mu$s in steps of 50 $\mu$s. The E2E latency plus buffer processing delay upper bound (U + g) was set to 600 $\mu$s. The lower bound of the E2E latency (W) was set to 50 $\mu$s. The source and destination processes were identical to those in the ideal situation.

### E. RESULTS IN THE REALISTIC SITUATION

Node 1 performs the source and network processes, and node 2 performs the buffer and destination processes. The results from both the dual-buffer and single-buffer approaches were obtained and compared. The clock frequency of the buffer was larger than the clock frequency of the source. The difference was 6 $\mu$s per second. The experiment ran for 120 s, with sufficient time for divergence to occur. In a realistic situation, the exact E2E buffered latency could not be obtained because the source and destination were implemented in different devices. Accordingly, the estimated E2E buffered latency was obtained as the sum of the latencies taken at each node. The jitter calculated using the estimated E2E buffered latency could also have an error. However, we could still determine the trends of latency and jitter variations. A total of 480,000 estimated end-to-end latencies were obtained during a single experiment, and the experiment was repeated 10 times. Rules (8), (9), and Algorithm 1 were applied. The case of dual buffer had jitter of 137–143 $\mu$s, as shown in Figure 13, and the case of a single buffer had jitter of 108–111 $\mu$s, as shown in Figure 14. In both approaches, the estimated end-to-end latency steadily reduced over time because the buffer clock frequency was larger than the source clock frequency, which caused the latency reduction experienced by the buffer. In addition, after some time, the estimated E2E buffered latency no longer decreased because Algorithm 1 worked to guarantee the clock difference to be within a certain level. During the experiment,

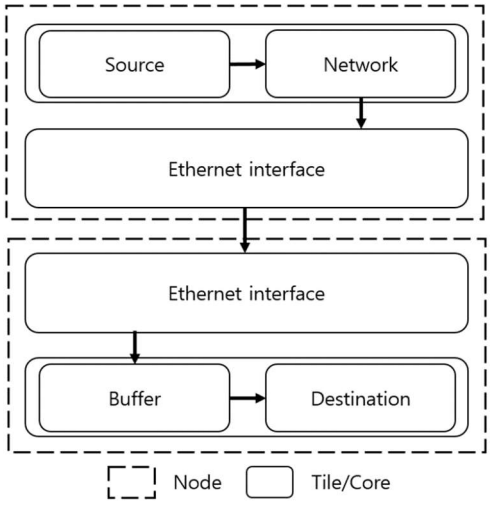


**FIGURE 12.** Processes and the topology in the realistic situation.

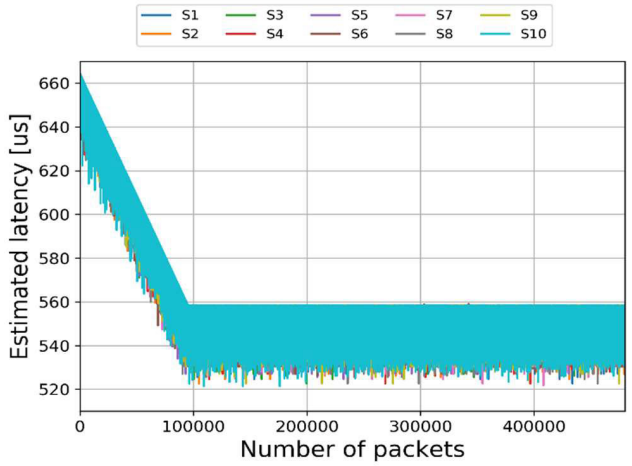


**FIGURE 13.** E2E buffered latency in the realistic situation with the dual buffer.

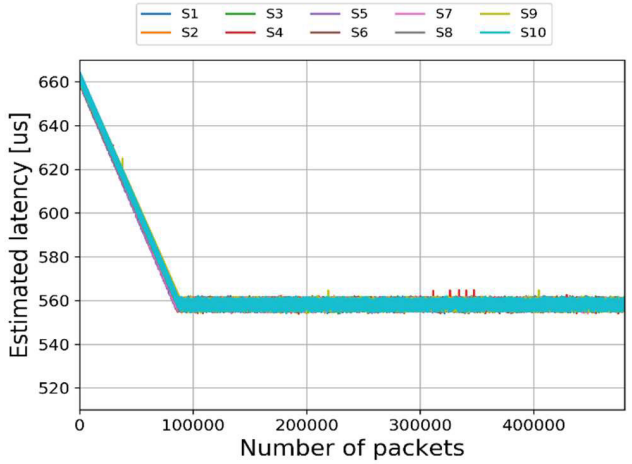


**FIGURE 14.** E2E buffered latency in the realistic situation with the single buffer.

the clock drift was actually detected, that is, the condition at line 20 of Algorithm 1 was met. The clock compensation was then enforced. As the clock drift worsened over time, the enforcement lasted until the end of the experiment. With the latency values measured only during the compensation enforcement, the dual-buffer method yielded less than 40 $\mu$s of jitter, and the single-buffer method showed less than 10 $\mu$s of jitter. Both methods had much less jitter than the upper jitter bound of a few milliseconds, which is required by many applications for high-priority flows. Theorem 4 suggests a jitter upper bound around 2U, which is around 1.2 ms in this case. This experimental result shows that Algorithm 1 can achieve much smaller jitter in real-world implementations than the theoretically guaranteed jitter upper bound.

## V. CONCLUSION

In this study, we proposed a scalable framework for both latency and jitter bounds guarantee, which are the most important performance metrics in deterministic services. Zero packet loss is guaranteed once the latency bound is given; therefore, the packet loss is beyond the scope of this study.

The key idea is to decompose the problem into two spaces: the first is to provide the latency bounds guarantee. Subsequently, by taking advantage of this guarantee, the jitter becomes a controllable parameter. Based on this approach, scalable determinism can be achieved even in large networks, including the Internet. In this study, three major contributions are given.

First, we proposed a scalable architecture to guarantee latency upper bounds, which includes regulators per class per port, the PFAR. The suggested regulators can act as cycle breakers. Regulators are scalable because they are not required to maintain flow states. It was also shown, through a numerical analysis with a simple network with a cycle and an extended network with four cycles, that the proposed architecture with the PFAR can achieve comparable latency bounds to the IEEE ATS technique. Both the PFAR and ATS frameworks showed a linear increase in latency bound as link utilization varied. In contrast, the network without regulators showed a quadratic increase, indicating instability. This instability becomes more significant as the network size increases. While the case studies used simple and symmetric networks, they captured the essential characteristics of the regulators as cycle breakers. Such a core network with several cycles, combined with edge networks with tree topologies, would form a network comparable to a common metro area network.

Second, we generalized the BN architecture such that the jitter is upper bounded even with buffers without cut-through capability and under the existence of clock drift. We proposed a clock drift compensation algorithm that does not require the exchange of messages between network nodes. Therefore, our algorithm is free from the requirements that are necessary in the NTP and PTP, such as the constant network delay or equal delays for both directions.

Third, we demonstrated with experiments on simple programmable microcontrollers that the jitter upper bound can be within a few tens of microseconds (10–40 $\mu$s) even in a realistic situation with store-and-forward buffers, clock drift of 6 $\mu$s per second, and random network delays in the range of 50–500 $\mu$s.

Consequently, an overall framework for both latency and jitter bounds was proposed, which is scalable and applicable to the Internet. The proposed regulator is of much less complexity because it maintains only a single flow-aggregate state, while any existing regulator, including the IEEE ATS framework's IR, should maintain all states of the flows. The proposed time-stamping function and buffer are located either at the edge of a network or even in the source/destination nodes. The number of flows at the edge of a network is usually within a manageable range. When compared, the synchronization, slot scheduling, and gate

control list necessary at the core of the network for the IEEE TSN synchronous approaches are much more complex and require a fundamental change to the existing Internet architecture.

We suggest that bounded E2E latencies can also be obtained by an asynchronous technique, such as ATS or DiffServ with regulators. These asynchronous techniques do not require clock or frequency synchronization. However, it can be argued that to obtain precise values of latencies, the network nodes should be at least frequency-synchronized. It is true that the processing delays and the queuing delays in nodes, and the transmission delays in links are all affected by the clock rates; therefore, even a slight clock drift may yield an inaccurate E2E latency bound. However, the error in the E2E latency bound is not cumulative and is affected only by the instantaneous clock rate difference, compared to the jitter that is affected by the accumulated clock drift over time, during the lifetime of a flow; therefore, it is negligible compared to its already conservative value of the bounds. Moreover, the maximum clock rate difference can be predicted and applied to the latency bound itself. How the clock inaccuracy affects the latency bounds in asynchronous networks is a topic for further study.

The proposed architecture for the latency bound with the PFAR requires that regulators be placed at every node. It is also plausible that placing the flow-aggregate-based regulators in only a part of the network, for example, only enough to cut the cycles in the network such that the burst does not explode owing to the cyclic dependency. However, even without cyclic dependency, the burst accumulates quickly in FIFO scheduling networks. We suggest a network with ADs wherein the flows are aggregated according to the ingress and egress interfaces of the AD, put into separate queues based on the FAs, and then scheduled. Regulators may be placed only at the edges of ADs. This is exactly the same framework as the FAIR, with the IRs replaced by the PFARs. It is for further study how much the performance would be degraded by the partial placement of the PFARs, while achieving greater scalability.

## REFERENCES

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

[2] *IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)*, document IETF RFC 3393, Nov. 2002. [Online]. Available: https://www.rfc-editor.org/info/rfc3393

[3] IEEE 802.1 Working Group. *Time-Sensitive Networking Task Group*. Accessed: Jan. 29, 2022. [Online]. Available: https://www.ieee802.org/1/pages/tsn.html

[4] *Timing and Synchronization Aspects in Packet Networks*, document Rec. ITU-T G.8261/Y.1361. Geneva, Switzerland, International Telecommunication Union, Aug. 2019.

[5] D. Chitimalla, K. Kondepu, L. Valcarenghi, M. Tornatore, and B. Mukherjee, "5G fronthaul-latency and jitter studies of CPRI over Ethernet," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 9, no. 2, pp. 172–182, Feb. 2017, doi: 10.1364/JOCN.9.000172.

[6] S. C. Ergen and P. Varaiya, "TDMA scheduling algorithms for wireless sensor networks," *Wireless Netw.*, vol. 16, no. 4, pp. 985–997, Jan. 2010, doi: 10.1007/s11276-009-0183-0.

[7] J. Mao, Z. Wu, and X. Wu, "A TDMA scheduling scheme for many-to-one communications in wireless sensor networks," *Comput. Commun.*, vol. 30, no. 4, pp. 863–872, Feb. 2007, doi: 10.1016/j.comcom.2006.10.006.

[8] *Network Time Protocol Version 4: Protocol and Algorithms Specification*, document IETF RFC 5905, Jun. 2010. [Online]. Available: https://www.rfc-editor.org/info/rfc5905

[9] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, Standard 1588, IEEE, 2019.

[10] O. Al-Kofahi, "Evaluating time synchronization using application-layer time-stamping," in *Proc. IEEE Wireless Commun. Netw. Conf.*, Apr. 2016, pp. 1–6, doi: 10.1109/wcnc.2016.7564909.

[11] T. Kovacshazy and B. Ferencz, "Performance evaluation of PTPd, a IEEE 1588 implementation, on the X86 Linux platform for typical application scenarios," in *Proc. IEEE Int. Instrum. Meas. Technol. Conf. Proc.*, May 2012, pp. 2548–2552, doi: 10.1109/I2MTC.2012.6229387.

[12] G. Cena, M. Cereia, I. C. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "A software implementation of IEEE 1588 on RTAI/RTnet platforms," in *Proc. IEEE 15th Conf. Emerg. Technol. Factory Autom. (ETFA)*, Bilbao, Spain, Sep. 2010, pp. 1–8, doi: 10.1109/ETFA.2010.5640955.

[13] B. Liu, S. Ren, C. Wang, V. Angilella, P. Medagliani, S. Martin, and J. Leguay, "Towards large-scale deterministic IP networks," in *Proc. IFIP Netw. Conf. (IFIP Networking)*, Jun. 2021, pp. 1–9, doi: 10.23919/IFIPNetworking52078.2021.9472798.

[14] S. Wang, "Large-scale deterministic IP networks on CENI," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, May 2021, pp. 1–6, doi: 10.1109/INFOCOMWKSHPS51825.2021.9484627.

[15] J. Krolikowski, S. Martin, P. Medagliani, J. Leguay, S. Chen, X. Chang, and X. Geng, "Joint routing and scheduling for large-scale deterministic IP networks," *Comput. Commun.*, vol. 165, pp. 33–42, Jan. 2021, doi: 10.1016/j.comcom.2020.10.016.

[16] 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.

[17] M. Andrews, "Instability of FIFO in the permanent sessions model at arbitrarily small network loads," *ACM Trans. Algorithms*, vol. 5, no. 3, pp. 1–29, Jul. 2009, doi: 10.1145/1541885.1541894.

[18] A. Bouillard, M. Boyer, and E. Le Corronc, "Deterministic network calculus: From theory to practical implementation," in *Networks and Telecommunications*. Hoboken, NJ, USA: Wiley, 2018, doi: 10.1002/9781119440284.

[19] 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.

[20] *Framework for Latency Guarantee in Large Scale Networks Including IMT-2020 Network*, document Rec. ITU-T Y.3113, Geneva, Switzerland, International Telecommunication Union, Feb. 2021.

[21] J. Joung and J. Kwon, "Zero jitter for deterministic networks without time-synchronization," *IEEE Access*, vol. 9, pp. 49398–49414, 2021, doi: 10.1109/ACCESS.2021.3068515.

[22] J.-Y. Le Boudec, "A theory of traffic regulators for deterministic networks with application to interleaved regulators," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2721–2733, Dec. 2018, doi: 10.1109/TNET.2018.2875191.

[23] 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.

[24] J. Migge. (Jan. 2018). *Insights on the Performance and Configuration of AVB and TSN in Automotive Ethernet Networks. Embedded Real-Time Software and Systems*. Toulouse, France: ERTS. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01746132.

[25] N. Navet, J. Villanueva, J. Migge, and M. Boyer, "Experimental assessment of QoS protocols for in-car Ethernet networks," in *Proc. IEEE Standards Association (IEEE-SA) Ethernet & IP*, San-Jose, CA, USA, Oct. 2017.

[26] *XMOS*. Accessed: Jan. 29, 2022. [Online]. Available: https://www.xmos.ai/xcore-200/

**JINOO JOUNG** received the B.S. degree in electronics engineering from the Korea Advanced Institute for Science and Technology, Taejon, 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 worked at Samsung Electronics and the Samsung Advanced Institute for Technology. His work in Samsung includes 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 was active in international standardization activities. He is the main editor of various ITU-T recommendations and currently holds the TTA ICT International Standard Expert title.

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

**JEONG-DONG RYOO** received the B.S. degree in EE from Kyungpook National University, Daegu, Republic of Korea, and the M.S. and Ph.D. degrees in EE from New York University, New York, NY, USA. After completing his Ph.D. in the area of telecommunication networks and optimization, he started working with Bell Labs, Lucent Technologies, Murray Hill, NJ, USA, in 1999. While he was with Bell Labs, he was mainly involved in performance analysis, evaluation, and enhancement studies for various wireless and wired network systems. Since he joined ETRI, in 2004, his work has been focused on next-generation networks, carrier class Ethernet, and MPLS-TP technology research, especially participating in OAM and protection standardization activities in ITU-T. He is currently a Principal Researcher at the ETRI, Daejeon, Republic of Korea, and the Head of network engineering major at the ETRI School, University of Science and Technology, Daejeon. He is a Vice-Chairperson of the ITU-T Study Group 15. He coauthored *TCP/IP Essentials: A Lab-Based Approach* (Cambridge University Press, 2004). He is a member of the Eta Kappa Nu Association. He is the 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), and G.808.1 (Generic protection-Linear).

**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 working as a Principal Researcher with the 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 coauthor of IETF RFC 7271 and RFC 8234. His current work focuses on time-sensitive packet networking technologies, such as IEEE 802.1 TSN and IETF DetNet. He is the Co-Editor of the ITU-T Rec. G.873.2 and G.808.2.